Hosting Environment (Daemon)

Generated by Doxygen 1.5.7.1

Tue Apr 28 11:10:24 2009

Contents

Chapter 1

Namespace Index

1.1 Namespace List

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

Chapter 2

Data Structure Index

2.1 Class Hierarchy

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

Chapter 3

Data Structure Index

3.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 4

Namespace Documentation

4.1 Arc Namespace Reference

Some utility methods for using xml security library (`http://www.aleksey.com/xmlsec/`).

Data Structures

- class **ACC**
- class **ACCConfig**
- class **ACCPluginArgument**
- class **ACCLoader**
- class **Broker**
- class **ClientInterface**
- class **ClientTCP**
- struct **HTTPClientInfo**
- class **ClientHTTP**
- class **ClientSOAP**
- class **SecHandlerConfig**
- class **DNListHandlerConfig**
- class **ARCPolicyHandlerConfig**
- class **ClientHTTPwithSAML2SSO**
- class **ClientSOAPwithSAML2SSO**
- class **ClientX509Delegation**
- struct **ApplicationEnvironment**
- class **ExecutionTarget**
- class **JDLParser**
- class **Job**
- class **JobController**
- struct **ReferenceTimeType**
- struct **EnvironmentType**
- struct **XLogueType**
- struct **RunTimeEnvironmentType**
- struct **NotificationType**
- struct **SourceType**
- struct **TargetType**

- struct **FileType**
- struct **DirectoryType**
- struct **OptionalElementType**
- class **JobDescription**
- class **JobDescriptionParser**
- class **JobSupervisor**
- class **JSDLParser**
- class **PosixJSDLParser**
- class **RSLValue**
- class **RSLLiteral**
- class **RSLVariable**
- class **RSLConcat**
- class **RSLList**
- class **RSLSequence**
- class **RSL**
- class **RSLBoolean**
- class **RSLCondition**
- class **RSLParser**
- class **Sandbox**
- class **Submitter**
- class **TargetGenerator**
- class **TargetRetriever**
- class **UserConfig**
- class **XRSLParser**
- class **Config**

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

- class **BaseConfig**
- class **ArcLocation**

  Determines ARC installation location.

- class **RegularExpression**

  A regular expression class.

- class **Base64**
- class **MemoryAllocationException**
- class **ByteArray**
- class **Counter**

  A class defining a common interface for counters.

- class **CounterTicket**

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

- class **ExpirationReminder**

  A class intended for internal use within counters.

- class **Period**
- class **Time**

  A class for storing and manipulating times.

- class **Database**

  Interface for calling database client library.

- class **Query**
- class **DItem**
- class **DBranch**
- class **DItemString**
- class **FileLock**
- class **IntraProcessCounter**

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

- 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 **MySQLDatabase**
- class **MySQLQuery**
- class **OptionParser**
- class **Run**
- class **SimpleCondition**

  Simple triggered condition.

- class **URL**

  Class to hold general URL's.

- class **URLLocation**

  *Class to hold a resolved **URL** (p. 286) location.*

- class **PathIterator**
- class **User**
- class **UserSwitch**
- class **AutoPointer**

  Wrapper for pointer with automatic destruction.

- class **CountedPointer**

  Wrapper for pointer with automatic destruction and mutiple references.

- class **NS**
- class **XMLNode**

Wrapper for LibXML library Tree interface.

- class **XMLNodeContainer**
- class **CredentialError**
- class **Credential**
- class **VOMSTrustList**
- 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** (p. 53) class.*

- class **DataBuffer**

  Represents set of buffers.

- class **DataCallback**
- class **DataHandle**

  *This class is a wrapper around the **DataPoint** (p. 91) class.*

- class **DataMover**
- class **DataPoint**

  *This base class is an abstraction of **URL** (p. 286).*

- class **DataPointDirect**

  This is a kind of generalized file handle.

- class **DataPointIndex**

  *Complements **DataPoint** (p. 91) with attributes common for Indexing **Service** (p. 270) URLs.*

- class **DataSpeed**

  Keeps track of average and instantaneous transfer speed.

- class **DataStatus**
- class **DMC**
- class **DMCConfig**
- class **DMCPluginArgument**
- class **DMCLoader**
- struct **CacheParameters**
- class **FileCache**
- class **FileInfo**

  ***FileInfo** (p. 151) stores information about files (metadata).*

- class **URLMap**
- class **XmlContainer**

- class **XmlDatabase**
- class **DelegationConsumer**
- class **DelegationProvider**
- class **DelegationConsumerSOAP**
- class **DelegationProviderSOAP**
- class **DelegationContainerSOAP**
- class **GlobusResult**
- class **GSSCredential**
- class **InfoCache**

  Stores XML document in filesystem split into parts.

- class **InfoCacheInterface**
- class **InfoFilter**

  Filters information document according to identity of requestor.

- class **InfoRegister**

  Registration to ISIS interface.

- class **InfoRegisters**

  Handling multiple registrations to ISISes.

- struct **Register_Info_Type**
- struct **ISIS_description**
- class **InfoRegistrar**

  Registration process associated with particular ISIS.

- class **InfoRegisterContainer**
- class **InformationInterface**

  Information System message processor.

- class **InformationContainer**

  Information System document container and processor.

- class **InformationRequest**

  Request for information in InfoSystem.

- class **InformationResponse**

  Informational response from InfoSystem.

- class **RegisteredService**

  Service (p. *270*) *- last component in a* ***Message*** (p. *192*) *Chain.*

- class **Loader**

  Plugins loader.

- class **LoadableModuleDesciption**
- class **ModuleManager**

  Manager of shared libraries.

- class **Plugin**

Base class for loadable ARC components.

- class **PluginArgument**

  Base class for passing arguments to loadable ARC components.

- struct **PluginDescriptor**

  Description of ARC lodable component.

- class **PluginsFactory**

  Generic ARC plugins loader.

- class **MCCInterface**

  *Interface for communication between **MCC** (p. 181), **Service** (p. 270) and **Plexer** (p. 228) objects.*

- class **MCC**

  ***Message** (p. 192) Chain Component - base class for every **MCC** (p. 181) plugin.*

- class **MCCConfig**
- class **MCCPluginArgument**
- class **MCC_Status**

  *A class for communication of **MCC** (p. 181) processing results.*

- class **MCCLoader**

  *Creator of **Message** (p. 192) Component Chains (**MCC** (p. 181)).*

- class **ChainContext**

  Interface to chain specific functionality.

- class **MessagePayload**

  Base class for content of message passed through chain.

- class **MessageContextElement**

  Top class for elements contained in message context.

- class **MessageContext**

  Handler for content of message context.

- class **MessageAuthContext**

  Handler for content of message auth∗ context.

- class **Message**

  Object being passed through chain of MCCs.

- class **AttributeIterator**

  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** (p. 192) objects.*

- struct **PayloadRawBuf**
- class **PayloadRaw**

  Raw byte multi-buffer.

- class **PayloadSOAP**

  *Payload of **Message** (p. 192) with SOAP content.*

- class **PayloadStreamInterface**

  *Stream-like Payload for **Message** (p. 192) object.*

- class **PayloadStream**

  POSIX handle as Payload.

- class **PlexerEntry**

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

- class **Plexer**

  *The **Plexer** (p. 228) class, used for routing messages to services.*

- class **CIStringValue**

  This class implements case insensitive strings as security attributes.

- class **SecAttrValue**

  This is an abstract interface to a security attribute.

- class **SecAttrFormat**

  Export/import format.

- class **SecAttr**

  This is an abstract interface to a security attribute.

- class **MultiSecAttr**

  *Container of multiple **SecAttr** (p. 261) attributes.*

- class **Service**

  ***Service** (p. 270) - last component in a **Message** (p. 192) Chain.*

- class **ServicePluginArgument**
- class **SOAPMessage**

  ***Message** (p. 192) restricted to SOAP payload.*

- class **ClassLoader**
- class **ClassLoaderPluginArgument**
- class **WSAEndpointReference**

  Interface for manipulation of WS-Adressing Endpoint Reference.

- class **WSAHeader**

  Interface for manipulation WS-Addressing information in SOAP header.

- class **SAMLToken**

  Class for manipulating SAML Token Profile.

- class **UsernameToken**

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

- class **X509Token**

  Class for manipulating X.509 Token Profile.

- class **PayloadWSRF**

  *This class combines **MessagePayload** (p. 203) with **WSRF** (p. 306).*

- 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** (p. 306) message.*

- class **WSRFBaseFault**

  *Base class for **WSRF** (p. 306) fault messages.*

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

  *Extends **XMLNode** (p. 316) class to support XML security operation.*

Typedefs

- typedef **Plugin** *(* **get_plugin_instance**)(**PluginArgument** *arg)
- typedef std::multimap< std::string, std::string > **AttrMap**
- typedef AttrMap::const_iterator **AttrConstIter**
- typedef AttrMap::iterator **AttrIter**

Enumerations

- enum **TimeFormat**
- enum **LogLevel**
- enum **StatusKind** { ,

  STATUS_OK = 1, **GENERIC_ERROR** = 2, **PARSING_ERROR** = 4, **PROTOCOL_-RECOGNIZED_ERROR** = 8,

  UNKNOWN_SERVICE_ERROR = 16, **BUSY_ERROR** = 32, **SESSION_CLOSE** = 64 }
- enum **WSAFault** { , **WSAFaultUnknown**, **WSAFaultInvalidAddressingHeader** }

Functions

- std::ostream & **operator**<< (std::ostream &, const Period &)
- std::ostream & **operator**<< (std::ostream &, const **Time** &)
- std::string **TimeStamp** (const **TimeFormat** &=Time::GetFormat())
- std::string **TimeStamp** (**Time**, const **TimeFormat** &=Time::GetFormat())
- void **GUID** (std::string &guid)
- std::string **UUID** (void)
- std::ostream & **operator**<< (std::ostream &os, **LogLevel** level)
- 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 **lower** (const std::string &s)
- std::string **upper** (const std::string &s)

- void **tokenize** (const std::string &str, std::vector< std::string > &tokens, const std::string &delimiters=" ")
- std::string **trim** (const std::string &str, const char *sep=NULL)
- std::string **uri_unescape** (const std::string &str)
- bool **CreateThreadFunction** (void(*func)(void *), void *arg)
- std::list< **URL** > **ReadURLList** (const **URL** &urllist)
- std::string **GetEnv** (const std::string &var)
- bool **SetEnv** (const std::string &var, const std::string &value)
- void **UnsetEnv** (const std::string &var)
- std::string **StrError** (int errnum=errno)
- bool **MatchXMLName** (const **XMLNode** &node1, const **XMLNode** &node2)
- bool **MatchXMLName** (const **XMLNode** &node, const char *name)
- bool **MatchXMLName** (const **XMLNode** &node, const std::string &name)
- bool **MatchXMLNamespace** (const **XMLNode** &node1, const **XMLNode** &node2)
- bool **MatchXMLNamespace** (const **XMLNode** &node, const char *uri)
- bool **MatchXMLNamespace** (const **XMLNode** &node, const std::string &uri)
- bool **createVOMSAC** (std::string &codedac, Credential &issuer_cred, Credential &holder_cred, std::vector< std::string > &fqan, std::vector< std::string > &targets, std::vector< std::string > &attributes, std::string &voname, std::string &uri, int lifetime)
- bool **addVOMSAC** (ArcCredential::AC **&aclist, std::string &decodedac)
- bool **parseVOMSAC** (X509 *holder, const std::string &ca_cert_dir, const std::string &ca_cert_file, const **VOMSTrustList** &vomscert_trust_dn, std::vector< std::string > &output, bool verify=true)
- bool **parseVOMSAC** (Credential &holder_cred, const std::string &ca_cert_dir, const std::string &ca_cert_file, const **VOMSTrustList** &vomscert_trust_dn, std::vector< std::string > &output, bool verify=true)
- char * **VOMSDecode** (const char *data, int size, int *j)
- std::string **string** (**StatusKind** kind)
- const char * **ContentFromPayload** (const **MessagePayload** &payload)
- void **WSAFaultAssign** (SOAPEnvelope &mesage, **WSAFault** fid)
- **WSAFault WSAFaultExtract** (SOAPEnvelope &message)
- int **passphrase_callback** (char *buf, int size, int rwflag, void *)
- bool **init_xmlsec** (void)
- bool **final_xmlsec** (void)
- std::string **get_cert_str** (const char *certfile)
- xmlSecKey * **get_key_from_keystr** (const std::string &value)
- xmlSecKey * **get_key_from_keyfile** (const char *keyfile)
- std::string **get_key_from_certfile** (const char *certfile)
- xmlSecKey * **get_key_from_certstr** (const std::string &value)
- xmlSecKeysMngrPtr **load_key_from_keyfile** (xmlSecKeysMngrPtr *keys_manager, const char *keyfile)
- xmlSecKeysMngrPtr **load_key_from_certfile** (xmlSecKeysMngrPtr *keys_manager, const char *certfile)
- xmlSecKeysMngrPtr **load_key_from_certstr** (xmlSecKeysMngrPtr *keys_manager, const std::string &certstr)
- xmlSecKeysMngrPtr **load_trusted_cert_file** (xmlSecKeysMngrPtr *keys_manager, const char *cert_file)
- xmlSecKeysMngrPtr **load_trusted_cert_str** (xmlSecKeysMngrPtr *keys_manager, const std::string &cert_str)
- xmlSecKeysMngrPtr **load_trusted_certs** (xmlSecKeysMngrPtr *keys_manager, const char *cafile, const char *capath)
- **XMLNode get_node** (**XMLNode** &parent, const char *name)

Variables

- const Glib::TimeVal **ETERNAL**
- const Glib::TimeVal **HISTORIC**
- const size_t **thread_stacksize** = (16 ∗ 1024 ∗ 1024)
- **Logger CredentialLogger**
- const char ∗ **plugins_table_name**

4.1.1 Detailed Description

Some utility methods for using xml security library (`http://www.aleksey.com/xmlsec/`).

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

Some implicit idea in the ClassLoader/ModuleManager stuff: share_lib_name (e.g. mccsoap) should be global identical plugin_name (e.g. __arc_attrfactory_modules__) should be global identical desc->name (e.g. attr.factory) should also be global identical

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** (p. 195) 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** (p. 195) 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 Plugin∗(∗ Arc::get_plugin_instance)(PluginArgument ∗arg)

Constructor function of ARC lodable component.

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

4.1.3 Enumeration Type Documentation

4.1.3.1 enum Arc::LogLevel

Logging levels.

Logging levels for tagging and filtering log messages. FATAL level designates very severe error events that will presumably lead the application to abort. ERROR level designates error events that might still allow the application to continue running. WARNING level designates potentially harmful situations. INFO level designates informational messages that highlight the progress of the application at coarse-grained level. DEBUG level designates fine-grained informational events that are most useful to debug an application. VERBOSE level designates finer-grained informational events than the DEBUG

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** (p. 192) does not fit into expected protocol.

BUSY_ERROR There is no destination configured for this message.

SESSION_CLOSE **Message** (p. 192) 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 bool Arc::addVOMSAC (ArcCredential::AC **& *aclist*, std::string & *decodedac*)

Add decoded AC string into a list of AC objects

Parameters:

aclist The list of AC objects

decodedac The AC string that is decoded from the string returned from voms server

4.1.4.2 const char* Arc::ContentFromPayload (const MessagePayload & *payload*)

Returns pointer to main memory chunk of **Message** (p. 192) payload.

If no buffer is present or if payload is not of **PayloadRawInterface** (p. 212) type NULL is returned.

4.1.4.3 bool Arc::CreateThreadFunction (void(*)(void *) *func*, void * *arg*)

Helper function to create simple thread.

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

4.1.4.4 bool Arc::createVOMSAC (std::string & *codedac*, Credential & *issuer_cred*, Credential & *holder_cred*, std::vector< std::string > & *fqan*, std::vector< std::string > & *targets*, std::vector< std::string > & *attributes*, std::string & *voname*, std::string & *uri*, int *lifetime*)

Create AC(Attribute Certificate) with voms specific format.

Parameters:

codedac The coded AC as output of this method

issuer_cred The issuer credential which is used to sign the AC

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

4.1.4.5 bool Arc::final_xmlsec (void)

Finalize the xml security library

4.1.4.6 std::string Arc::get_cert_str (const char * *certfile*)

Get certificate in string format from certificate file

4.1.4.7 std::string Arc::get_key_from_certfile (const char * *certfile*)

Get public key in string format from certificate file

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

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

4.1.4.9 xmlSecKey* Arc::get_key_from_keyfile (const char * *keyfile*)

Get key in xmlSecKey structure from key file

4.1.4.10 xmlSecKey* Arc::get_key_from_keystr (const std::string & *value*)

Get key in xmlSecKey structure from key in string format

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

Generate a new child **XMLNode** (p. 316) with specified name

4.1.4.12 std::string Arc::GetEnv (const std::string & *var*)

Portable function for getting environment variables.

4.1.4.13 void Arc::GUID (std::string & *guid*)

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

4.1.4.14 bool Arc::init_xmlsec (void)

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

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

Load public key from a certificate file into key manager

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

Load public key from a certificate string into key manager

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

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

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

Load trusted certificate from certificate file into key manager

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

Load trusted certificate from cetrtificate string into key manager

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

Load trusted cetificates from a file or directory into key manager

4.1.4.21 std::string Arc::lower (const std::string & *s*)

This method converts to lower case of the string.

4.1.4.22 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

4.1.4.23 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

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

Returns true if underlying XML elements have same names

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

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

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

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

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

Returns true if underlying XML elements belong to same namespaces

4.1.4.28 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.

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

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

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

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

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

Parse the certificate. The same as the above one

4.1.4.32 bool Arc::parseVOMSAC (X509 * *holder*, const std::string & *ca_cert_dir*, const std::string & *ca_cert_file*, const VOMSTrustList & *vomscert_trust_dn*, std::vector< std::string > & *output*, bool *verify* = `true`)

Parse the certificate, and output the attributes.

Parameters:

holder The proxy certificate which includes the voms specific formated AC.

ca_cert_dir The trusted certificates which are used to verify the certificate which is used to sign the AC

ca_cert_file The same as ca_cert_dir except it is a file instead of a directory. Only one of them need to be set

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

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

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

Parameters:

verify true: Verify the voms certificate is trusted based on the ca_cert_dir/ca_cert_file which specifies the CA certificates, and the vomscert_trust_dn which specifies the trusted DN chain from voms server certificate to CA certificate.

false: Not verify, which means the issuer of AC (voms server certificate is supposed to be trusted by default). In this case the parameters 'ca_cert_dir', 'ca_cert_file' and 'vomscert_trust_dn' will not effect, and should be set as empty. This case is specifically used by 'arcproxy –info' to list all of the attributes in AC, and not to need to verify if the AC's issuer is trusted.

4.1.4.33 int Arc::passphrase_callback (char * *buf*, int *size*, int *rwflag*, void *)

callback method for inputing passphrase of key file

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

Reads a list of URLs from a file.

4.1.4.35 bool Arc::SetEnv (const std::string & *var*, const std::string & *value*)

Portable function for setting environment variables.

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

Portable function for obtaining description of last system error.

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

Conversion to string.

Conversion from StatusKind to string.

Parameters:

kind The StatusKind to convert.

4.1.4.38 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.

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

This method converts a string to any type.

References Arc::Logger::msg().

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

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

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

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

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

This method tokenize string.

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

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

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

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

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

Portable function for unsetting environment variables.

4.1.4.46 std::string Arc::upper (const std::string & *s*)

This method converts to upper case of the string.

4.1.4.47 std::string Arc::uri_unescape (const std::string & *str*)

This method unescape the URI encoded string.

4.1.4.48 std::string Arc::UUID (void)

This function generates uuid.

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

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

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

Makes WS-Addressing fault.

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

4.1.4.51 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 Logger Arc::CredentialLogger

Logger (p. 173) to be used by all modules of credentials library

4.1.5.2 const Glib::TimeVal Arc::ETERNAL

A time very far in the future.

4.1.5.3 const Glib::TimeVal Arc::HISTORIC

A time very far in the past.

4.1.5.4 const char* Arc::plugins_table_name

Name of symbol refering to table of plugins.

This C null terminated string specifies name of symbol which shared library should export to give an access to an array of **PluginDescriptor** (p. 233) elements. The array is terminated by element with all components set to NULL.

4.1.5.5 const size_t Arc::thread_stacksize = (16 * 1024 * 1024)

This module provides convenient helpers for Glibmm interface for thread management.

So far it takes care of automatic initialization of threading environment and creation of simple detached threads. Defines size of stack assigned to every new thread.

4.2 ArcCredential Namespace Reference

Data Structures

- struct **cert_verify_context**
- struct **PROXYPOLICY_st**
- struct **PROXYCERTINFO_st**
- struct **ACDIGEST**
- struct **ACIS**
- struct **ACFORM**
- struct **ACACI**
- struct **ACHOLDER**
- struct **ACVAL**
- struct **ACIETFATTR**
- struct **ACTARGET**
- struct **ACTARGETS**
- struct **ACATTR**
- struct **ACINFO**
- struct **ACC**
- struct **ACSEQ**
- struct **ACCERTS**
- struct **ACATTRIBUTE**
- struct **ACATTHOLDER**
- struct **ACFULLATTRIBUTES**

Enumerations

- enum **certType** {

  CERT_TYPE_EEC, **CERT_TYPE_CA**, **CERT_TYPE_GSI_3_IMPERSONATION_PROXY**, **CERT_TYPE_GSI_3_INDEPENDENT_PROXY**,

  CERT_TYPE_GSI_3_LIMITED_PROXY, **CERT_TYPE_GSI_3_RESTRICTED_PROXY**, **CERT_TYPE_GSI_2_PROXY**, **CERT_TYPE_GSI_2_LIMITED_PROXY**,

  CERT_TYPE_RFC_IMPERSONATION_PROXY, **CERT_TYPE_RFC_INDEPENDENT_PROXY**, **CERT_TYPE_RFC_LIMITED_PROXY**, **CERT_TYPE_RFC_RESTRICTED_PROXY**,

  CERT_TYPE_RFC_ANYLANGUAGE_PROXY }

4.2.1 Detailed Description

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

Borrow the code about Attribute Certificate from VOMS The **VOMSAttribute.h** (p. **??**) and VOMSAttribute.cpp are integration about code written by VOMS project, so here the original license follows.

4.2.2 Enumeration Type Documentation

4.2.2.1 enum ArcCredential::certType

Enumerator:

CERT_TYPE_EEC A end entity certificate

CERT_TYPE_CA A CA certificate

CERT_TYPE_GSI_3_IMPERSONATION_PROXY A X.509 Proxy Certificate Profile (pre-RFC) compliant impersonation proxy

CERT_TYPE_GSI_3_INDEPENDENT_PROXY A X.509 Proxy Certificate Profile (pre-RFC) compliant independent proxy

CERT_TYPE_GSI_3_LIMITED_PROXY A X.509 Proxy Certificate Profile (pre-RFC) compliant limited proxy

CERT_TYPE_GSI_3_RESTRICTED_PROXY A X.509 Proxy Certificate Profile (pre-RFC) compliant restricted proxy

CERT_TYPE_GSI_2_PROXY A legacy Globus impersonation proxy

CERT_TYPE_GSI_2_LIMITED_PROXY A legacy Globus limited impersonation proxy

CERT_TYPE_RFC_IMPERSONATION_PROXY A X.509 Proxy Certificate Profile RFC compliant impersonation proxy; RFC inheritAll proxy

CERT_TYPE_RFC_INDEPENDENT_PROXY A X.509 Proxy Certificate Profile RFC compliant independent proxy; RFC independent proxy

CERT_TYPE_RFC_LIMITED_PROXY A X.509 Proxy Certificate Profile RFC compliant limited proxy

CERT_TYPE_RFC_RESTRICTED_PROXY A X.509 Proxy Certificate Profile RFC compliant restricted proxy

CERT_TYPE_RFC_ANYLANGUAGE_PROXY RFC anyLanguage proxy

Chapter 5

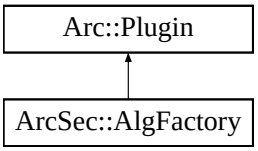
Data Structure Documentation

5.1 ArcSec::AlgFactory Class Reference

Interface for algorithm factory class.

`#include <AlgFactory.h>`

Inheritance diagram for ArcSec::AlgFactory::



Public Member Functions

- virtual **CombiningAlg** ∗ **createAlg** (const std::string &type)=0

5.1.1 Detailed Description

Interface for algorithm factory class.

AlgFactory (p. 33) is in charge of creating **CombiningAlg** (p. 59) according to the algorithm type given as argument of method createAlg. This class can be inherited for implementing a factory class which can create some specific combining algorithm objects.

5.1.2 Member Function Documentation

5.1.2.1 virtual CombiningAlg∗ ArcSec::AlgFactory::createAlg (const std::string & *type*) [pure virtual]

creat algorithm object based on the type algorithm type

Parameters:

type The type of combining algorithm

Returns:

The object of **CombiningAlg** (p. 59)

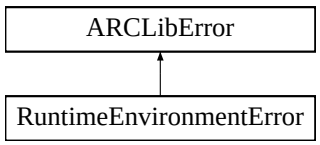
The documentation for this class was generated from the following file:

- AlgFactory.h

5.2 ARCLibError Class Reference

`#include <error.h>`

Inheritance diagram for ARCLibError::



Public Member Functions

- **ARCLibError** (std::string message)
- virtual ~**ARCLibError** () throw ()
- virtual const char * **what** () const throw ()

5.2.1 Detailed Description

Contains common error message functionality This is the top exception for ARCLib. Every exeption in ARCLib should inherit from this. The exception inherits from the top C++ exception: std::exception.

5.2.2 Constructor & Destructor Documentation

5.2.2.1 ARCLibError::ARCLibError (std::string *message*) [inline]

Creates a new exception, with the mesage given as argument

5.2.2.2 virtual ARCLibError::~ARCLibError () throw () [inline, virtual]

Destructor. Not that much to say.

5.2.3 Member Function Documentation

5.2.3.1 virtual const char* ARCLibError::what () const throw () [inline, virtual]

Returns the message given in the constructer.

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

- error.h

5.3 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** ()

5.3.1 Detailed Description

Determines ARC installation location.

5.3.2 Member Function Documentation

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

Returns ARC installation location.

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

Returns ARC plugins directory location.

Main source is value of variable ARC_PLUGIN_PATH, otherwise path is derived from installation location.

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

Initializes location information.

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

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

- ArcLocation.h

5.4 ArcSec::Attr Struct Reference

Attr (p. 37) contains a tuple of attribute type and value.

`#include <Request.h>`

5.4.1 Detailed Description

Attr (p. 37) contains a tuple of attribute type and value.

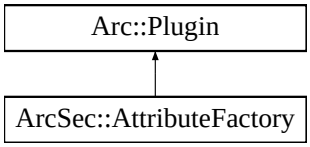
The documentation for this struct was generated from the following file:

- Request.h

5.5 ArcSec::AttributeFactory Class Reference

`#include <AttributeFactory.h>`

Inheritance diagram for ArcSec::AttributeFactory::



5.5.1 Detailed Description

Base attribute factory class

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

- AttributeFactory.h

5.6 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**

5.6.1 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** (p. 195) class returns an **AttributeIterator** (p. 39) object that can be used to access the values of the attribute.

Typical usage is:

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

5.6.2 Constructor & Destructor Documentation

5.6.2.1 Arc::AttributeIterator::AttributeIterator ()

Default constructor.

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

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

Protected constructor used by the **MessageAttributes** (p. 195) 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** (p. 195) class.

Parameters:

begin A const_iterator pointing to the first matching key-value pair in the internal multimap of the **MessageAttributes** (p. 195) class.

end A const_iterator pointing to the first key-value pair in the internal multimap of the **MessageAttributes** (p. 195) class where the key is larger than the key searched for.

5.6.3 Member Function Documentation

5.6.3.1 bool Arc::AttributeIterator::hasMore () const

Predicate method for iteration termination.

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

Returns:

Returns true if there are more values, otherwise false.

5.6.3.2 const std::string& Arc::AttributeIterator::key (void) const

The key of attribute.

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

5.6.3.3 const std::string& Arc::AttributeIterator::operator* () const

The dereference operator.

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

Returns:

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

5.6.3.4 AttributeIterator Arc::AttributeIterator::operator++ (int)

The postfix advance operator.

Advances the iterator to the next value. Works intuitively.

Returns:

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

5.6.3.5 const AttributeIterator& Arc::AttributeIterator::operator++ ()

The prefix advance operator.

Advances the iterator to the next value. Works intuitively.

Returns:

A const reference to this iterator.

5.6.3.6 const std::string* Arc::AttributeIterator::operator → () const

The arrow operator.

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

5.6.4 Friends And Related Function Documentation

5.6.4.1 friend class MessageAttributes [friend]

The **MessageAttributes** (p. 195) class is a friend.

The constructor that creates an **AttributeIterator** (p. 39) that is connected to the internal multimap of the **MessageAttributes** (p. 195) class should not be exposed to the outside, but it still needs to be accessible from the getAll() method of the **MessageAttributes** (p. 195) class. Therefore, that class is a friend.

5.6.5 Field Documentation

5.6.5.1 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** (p. 195) class.

5.6.5.2 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** (p. 195) class where the key is larger than the key searched for.

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

- MessageAttributes.h

5.7 ArcSec::AttributeProxy Class Reference

Interface for creating the **AttributeValue** (p. 43) object, it will be used by **AttributeFactory** (p. 38).

`#include <AttributeProxy.h>`

Public Member Functions

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

5.7.1 Detailed Description

Interface for creating the **AttributeValue** (p. 43) object, it will be used by **AttributeFactory** (p. 38).

The **AttributeProxy** (p. 42) object will be insert into AttributeFactoty; and the getAttribute(node) method will be called inside AttributeFacroty.createvalue(node), in order to create a specific **AttributeValue** (p. 43)

5.7.2 Member Function Documentation

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

Create a **AttributeValue** (p. 43) object according to the information inside the XMLNode as parameter.

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

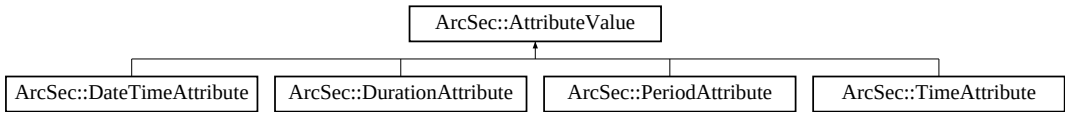
- AttributeProxy.h

5.8 ArcSec::AttributeValue Class Reference

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

`#include <AttributeValue.h>`

Inheritance diagram for ArcSec::AttributeValue::



Public Member Functions

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

5.8.1 Detailed Description

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

<Attribute> contains different "Type" definition; Each type of <Attribute> needs different approach to compare the value. Any specific class which is for processing specific "Type" shoud inherit this class. The "Type" supported so far is: StringAttribute, DateAttribute, **TimeAttribute** (p. 284), **DurationAttribute** (p. 132), **PeriodAttribute** (p. 224), AnyURIAttribute, X500NameAttribute

5.8.2 Member Function Documentation

5.8.2.1 virtual std::string ArcSec::AttributeValue::encode () `[pure virtual]`

encode the value in a string format

Implemented in **ArcSec::DateTimeAttribute** (p. 118), **ArcSec::TimeAttribute** (p. 284), **ArcSec::DurationAttribute** (p. 132), and **ArcSec::PeriodAttribute** (p. 224).

5.8.2.2 virtual bool ArcSec::AttributeValue::equal (AttributeValue ∗ *value*) `[pure virtual]`

Evluate whether "this" equale to the parameter value

Implemented in **ArcSec::DateTimeAttribute** (p. 118), **ArcSec::TimeAttribute** (p. 284), **ArcSec::DurationAttribute** (p. 132), and **ArcSec::PeriodAttribute** (p. 224).

5.8.2.3 virtual std::string ArcSec::AttributeValue::getId () `[pure virtual]`

Get the identifier of the <Attribute>

Implemented in **ArcSec::DateTimeAttribute** (p. 118), **ArcSec::TimeAttribute** (p. 284), **ArcSec::DurationAttribute** (p. 132), and **ArcSec::PeriodAttribute** (p. 224).

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

Get the type of the <Attribute>

Implemented in **ArcSec::DateTimeAttribute** (p. 118), **ArcSec::TimeAttribute** (p. 284), **ArcSec::DurationAttribute** (p. 132), and **ArcSec::PeriodAttribute** (p. 224).

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

- AttributeValue.h

5.9 ArcSec::Attrs Class Reference

Attrs (p. 45) is a container for one or more **Attr** (p. 37).

`#include <Request.h>`

5.9.1 Detailed Description

Attrs (p. 45) is a container for one or more **Attr** (p. 37).

Attrs (p. 45) 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

5.10 ArcSec::AuthzRequestSection Struct Reference

`#include <PDP.h>`

5.10.1 Detailed Description

These structure are based on the request schema for **PDP** (p. 223), 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

5.11 Arc::AutoPointer< T > Class Template Reference

Wrapper for pointer with automatic destruction.

`#include <Utils.h>`

Public Member Functions

- **AutoPointer** (void)
- **AutoPointer** (T *o)
- ~**AutoPointer** (void)
- T & **operator*** (void) const
- T * **operator** -> (void) const
- **operator bool** (void) const
- bool **operator!** (void) const
- **operator T** * (void) const

5.11.1 Detailed Description

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

Wrapper for pointer with automatic destruction.

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

5.11.2 Constructor & Destructor Documentation

5.11.2.1 template<typename T > Arc::AutoPointer< T >::AutoPointer (void) [inline]

NULL pointer constructor.

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

Constructor which wraps pointer.

5.11.2.3 template<typename T > Arc::AutoPointer< T >::~AutoPointer (void) [inline]

Destructor destroys wrapped object using delete().

5.11.3 Member Function Documentation

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

Returns false if pointer is NULL and true otherwise.

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

Cast to original pointer.

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

Returns true if pointer is NULL and false otherwise.

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

For refering wrapped object.

5.11.3.5 template<typename T > T* Arc::AutoPointer< T >::operator -> (void) const [inline]

For refering wrapped object.

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

- Utils.h

5.12 Arc::BaseConfig Class Reference

`#include <ArcConfig.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

5.12.1 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.

5.12.2 Member Function Documentation

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

Add CA directory

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

Add CA file

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

Add certificate

5.12.2.4 void Arc::BaseConfig::AddOverlay (XMLNode *cfg*)

Add configuration overlay

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

Adds non-standard location of plugins

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

Add private key

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

Add credentials proxy

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

Read overlay from file

5.12.2.9 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:

- ArcConfig.h

5.13 Arc::CacheParameters Struct Reference

`#include <FileCache.h>`

5.13.1 Detailed Description

Contains data on the parameters of a cache.

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

- FileCache.h

5.14 Arc::ChainContext Class Reference

Interface to chain specific functionality.

`#include <MCCLoader.h>`

Public Member Functions

- **operator PluginsFactory** ∗ ()

5.14.1 Detailed Description

Interface to chain specific functionality.

Object of this class is associated with every **MCCLoader** (p. 189) object. It is accessible for **MCC** (p. 181) and **Service** (p. 270) components and provides an interface to manipulate chains stored in **Loader** (p. 170). 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 PluginsFactory ∗ () [inline]

Returns associated **PluginsFactory** (p. 234) object

References Arc::Loader::factory_.

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

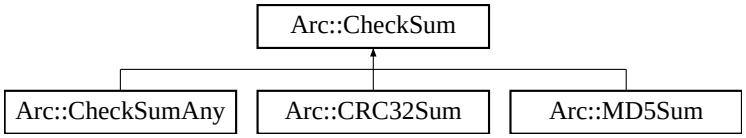
- MCCLoader.h

5.15 Arc::CheckSum Class Reference

Defines interface for variuos checksum manipulations.

`#include <CheckSum.h>`

Inheritance diagram for Arc::CheckSum::



5.15.1 Detailed Description

Defines interface for variuos checksum manipulations.

This class is used during data transfers through **DataBuffer** (p. 78) class

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

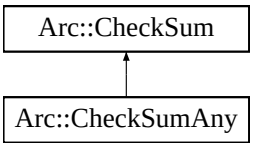
- CheckSum.h

5.16 Arc::CheckSumAny Class Reference

Wraper for **CheckSum** (p. 53) class.

`#include <CheckSum.h>`

Inheritance diagram for Arc::CheckSumAny::



5.16.1 Detailed Description

Wraper for **CheckSum** (p. 53) 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:

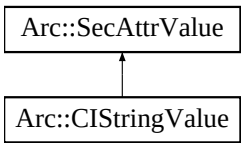
- CheckSum.h

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)

5.17.1 Detailed Description

This class implements case insensitive strings as security attributes.

This is an example of how to inherit **SecAttrValue** (p. 265). 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 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** (p. 265).

5.17.3.2 virtual Arc::CIStringValue::operator bool () [virtual]

This function returns false if the string is empty or uninitialized

Reimplemented from **Arc::SecAttrValue** (p. 265).

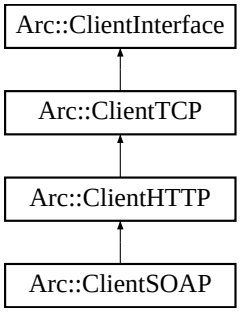
The documentation for this class was generated from the following file:

- CIStringValue.h

5.18 Arc::ClientSOAP Class Reference

`#include <ClientInterface.h>`

Inheritance diagram for Arc::ClientSOAP::



Public Member Functions

- **ClientSOAP** ()
- **MCC_Status process** (**PayloadSOAP** ∗request, **PayloadSOAP** ∗∗response)
- **MCC_Status process** (const std::string &action, **PayloadSOAP** ∗request, **PayloadSOAP** ∗∗response)
- **MCC** ∗ **GetEntry** ()
- void **AddSecHandler** (**XMLNode** handlercfg, const std::string &libanme="", const std::string &libpath="")
- virtual void **Load** ()

5.18.1 Detailed Description

Class with easy interface for sending/receiving SOAP messages over HTTP(S/G). It takes care of configuring **MCC** (p. 181) chain and making an entry point.

5.18.2 Constructor & Destructor Documentation

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

Constructor creates **MCC** (p. 181) chain and connects to server.

5.18.3 Member Function Documentation

5.18.3.1 void Arc::ClientSOAP::AddSecHandler (XMLNode *handlercfg*, const std::string & *libanme* = "", const std::string & *libpath* = "")

Adds security handler to configuration of SOAP **MCC** (p. 181)

5.18.3.2 MCC* Arc::ClientSOAP::GetEntry () [inline]

Returns entry point to SOAP **MCC** (p. 181) in configured chain. To initialize entry point **Load()** (p. 58) method must be called.

5.18.3.3 virtual void Arc::ClientSOAP::Load () [virtual]

Instantiates pluggable elements according to generated configuration

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

Send SOAP request with specified SOAP action and receive response.

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

Send SOAP request and receive response.

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

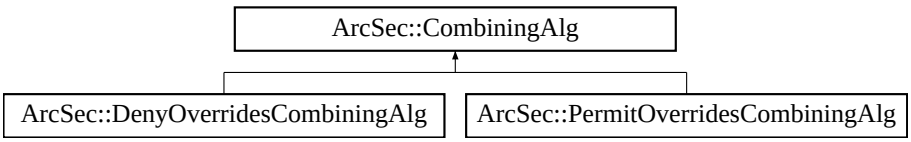
- ClientInterface.h

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

5.19.1 Detailed Description

Interface for combining algrithm.

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

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 each policy will be called, and then those results from each policy will be combined according to the combining algorithm inside CombingAlg class.

Implemented in **ArcSec::DenyOverridesCombiningAlg** (p. 130), and **ArcSec::PermitOverridesCombiningAlg** (p. 226).

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

Get the identifier of the combining algorithm class

Returns:

The identity of the algorithm

Implemented in **ArcSec::DenyOverridesCombiningAlg** (p. 130), and **ArcSec::PermitOverridesCombiningAlg** (p. 226).

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

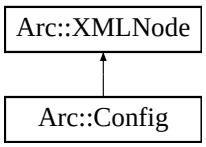
- CombiningAlg.h

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 char *filename)
- **Config** (const std::string &xml_str)
- **Config** (**XMLNode** xml)
- **Config** (long cfg_ptr_addr)
- **Config** (const **Config** &cfg)
- void **print** (void)
- void **parse** (const char *filename)
- const std::string & **getFileName** (void)
- void **setFileName** (const std::string &filename)
- void **save** (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]`

Creates empty XML tree

5.20.2.2 Arc::Config::Config (const char * *filename*)

Loads configuration document from file 'filename'

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

Parse configuration document from memory

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

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

5.20.2.5 Arc::Config::Config (long *cfg_ptr_addr*)

Copy constructor used by language bindings

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

Copy constructor used by language bindings

5.20.3 Member Function Documentation

5.20.3.1 const std::string& Arc::Config::getFileName (void) [inline]

Gives back file name of config file or empty string if it was generared from the **XMLNode** (p. 316) subtree

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

Parse configuration document from file 'filename'

5.20.3.3 void Arc::Config::print (void)

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

5.20.3.4 void Arc::Config::save (const char * *filename*)

Save to file

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

Set the file name of config file

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

- ArcConfig.h

5.21 Arc::CountedPointer< T > Class Template Reference

Wrapper for pointer with automatic destruction and mutiple references.

`#include <Utils.h>`

Data Structures

- class **Base**

Public Member Functions

- T & **operator*** (void) const
- T * **operator ->** (void) const
- **operator bool** (void) const
- bool **operator!** (void) const
- **operator T *** (void) const

5.21.1 Detailed Description

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

Wrapper for pointer with automatic destruction and mutiple references.

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

5.21.2 Member Function Documentation

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

Returns false if pointer is NULL and true otherwise.

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

Cast to original pointer.

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

Returns true if pointer is NULL and false otherwise.

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

For refering wrapped object.

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

For refering wrapped object.

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

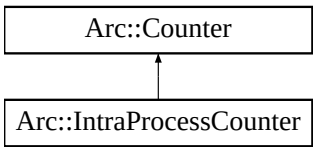
- Utils.h

5.22 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**

5.22.1 Detailed Description

A class defining a common interface for counters.

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

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

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

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

Typical usage is:

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

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

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

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

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

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

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

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

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

5.22.2 Member Typedef Documentation

5.22.2.1 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** (p. 72) class in order to be able to cancel and extend reservations.

5.22.3 Constructor & Destructor Documentation

5.22.3.1 Arc::Counter::Counter () [protected]

Default constructor.

This is the default constructor. Since **Counter** (p. 65) is an abstract class, it should only be used by subclasses. Therefore it is protected. Furthermore, since the **Counter** (p. 65) class has no attributes, nothing needs to be initialized and thus this constructor is empty.

5.22.3.2 virtual Arc::Counter::~Counter () [virtual]

The destructor.

This is the destructor of the **Counter** (p. 65) class. Since the **Counter** (p. 65) class has no attributes, nothing needs to be cleaned up and thus the destructor is empty.

5.22.4 Member Function Documentation

5.22.4.1 virtual void Arc::Counter::cancel (IDType *reservationID*) [protected, pure virtual]

Cancellation of a reservation.

This method cancels a reservation. It is called by the **CounterTicket** (p. 72) that corresponds to the reservation.

Parameters:

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

5.22.4.2 virtual int Arc::Counter::changeExcess (int *amount*) [pure virtual]

Changes the excess limit of the counter.

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

Parameters:

amount The amount by which to change the excess limit.

Returns:

The new excess limit.

Implemented in **Arc::IntraProcessCounter** (p. 167).

5.22.4.3 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** (p. 167).

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

Extension of a reservation.

This method extends a reservation. It is called by the **CounterTicket** (p. 72) 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.

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

A "relay method" for a constructor of the **CounterTicket** (p. 72) class.

This method acts as a relay for one of the constructors of the **CounterTicket** (p. 72) class. That constructor is private, but needs to be accessible from the subclasses of **Counter** (p. 65) (bot not from anywhere else). In order not to have to declare every possible subclass of **Counter** (p. 65) as a friend of **CounterTicket** (p. 72), only the base class **Counter** (p. 65) 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** (p. 72).

expiryTime the expiry time of the reservation corresponding to the **CounterTicket** (p. 72).

counter The **Counter** (p. 65) from which the reservation has been made.

Returns:

The counter ticket that has been created.

5.22.4.6 Glib::TimeVal Arc::Counter::getCurrentTime () [protected]

Get the current time.

Returns the current time. An "adapter method" for the assign_current_time() method in the Glib::TimeVal class. return The current time.

5.22.4.7 virtual int Arc::Counter::getExcess () [pure virtual]

Returns the excess limit of the counter.

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

Returns:

The excess limit.

Implemented in **Arc::IntraProcessCounter** (p. 168).

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

A "relay method" for the constructor of **ExpirationReminder** (p. 143).

This method acts as a relay for one of the constructors of the **ExpirationReminder** (p. 143) class. That constructor is private, but needs to be accessible from the subclasses of **Counter** (p. 65) (bot not from anywhere else). In order not to have to declare every possible subclass of **Counter** (p. 65) as a friend of **ExpirationReminder** (p. 143), only the base class **Counter** (p. 65) 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** (p. 143).

resID The identity number of the reservation corresponding to the **ExpirationReminder** (p. 143).

Returns:

The **ExpirationReminder** (p. 143) that has been created.

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

Computes an expiry time.

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

Parameters:

duration The duration.

Returns:

The expiry time.

5.22.4.10 virtual int Arc::Counter::getLimit () [pure virtual]

Returns the current limit of the counter.

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

Returns:

The current limit of the counter.

Implemented in **Arc::IntraProcessCounter** (p. 168).

5.22.4.11 virtual int Arc::Counter::getValue () [pure virtual]

Returns the current value of the counter.

Returns the current value of the counter, i.e. the number of unreserved units. Initially, the value is equal to the limit of the counter. When a reservation is made, the 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** (p. 168).

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

Makes a reservation from the counter.

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

Parameters:

amount The amount to reserve, default value is 1.

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

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

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

Returns:

A **CounterTicket** (p. 72) that can be queried about the status of the reservation as well as for cancellations and extensions.

Implemented in **Arc::IntraProcessCounter** (p. 169).

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

Sets the excess limit of the counter.

This method sets a new excess limit for the counter.

Parameters:

newExcess The new excess limit, an absolute number.

Returns:

The new excess limit.

Implemented in **Arc::IntraProcessCounter** (p. 169).

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

Sets the limit of the counter.

This method sets a new limit for the counter.

Parameters:

newLimit The new limit, an absolute number.

Returns:

The new limit.

Implemented in **Arc::IntraProcessCounter** (p. 169).

5.22.5 Friends And Related Function Documentation

5.22.5.1 friend class CounterTicket [friend]

The **CounterTicket** (p. 72) class needs to be a friend.

5.22.5.2 friend class ExpirationReminder [friend]

The **ExpirationReminder** (p. 143) class needs to be a friend.

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

- Counter.h

5.23 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**

5.23.1 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** (p. 65), a ReservationTicket is returned. This ticket can then be queried about the validity of a reservation. It can also be used for cancelation and extension of reservations.

Typical usage is:

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

5.23.2 Constructor & Destructor Documentation

5.23.2.1 Arc::CounterTicket::CounterTicket ()

The default constructor.

This is the default constructor. It creates a **CounterTicket** (p. 72) 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** (p. 65).

5.23.3 Member Function Documentation

5.23.3.1 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.

5.23.3.2 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.

5.23.3.3 bool Arc::CounterTicket::isValid ()

Returns the validity of a **CounterTicket** (p. 72).

This method checks whether a **CounterTicket** (p. 72) is valid. The ticket was probably returned earlier by the reserve() method of a **Counter** (p. 65) but the corresponding reservation may have expired.

Returns:

The validity of the ticket.

5.23.4 Friends And Related Function Documentation

5.23.4.1 friend class Counter [friend]

The **Counter** (p. 65) class needs to be a friend.

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

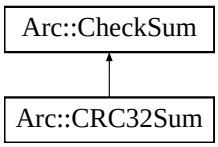
- Counter.h

5.24 Arc::CRC32Sum Class Reference

Implementation of CRC32 checksum.

`#include <CheckSum.h>`

Inheritance diagram for Arc::CRC32Sum::



5.24.1 Detailed Description

Implementation of CRC32 checksum.

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

- CheckSum.h

5.25 Arc::CredentialError Class Reference

`#include <Credential.h>`

Public Member Functions

- **CredentialError** (const std::string &what="")

5.25.1 Detailed Description

This is an exception class that is used to handle runtime errors discovered in the Credential class.

5.25.2 Constructor & Destructor Documentation

5.25.2.1 Arc::CredentialError::CredentialError (const std::string & *what* = "")

This is the constructor of the **CredentialError** (p. 75) class.

Parameters:

what An explanation of the error.

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

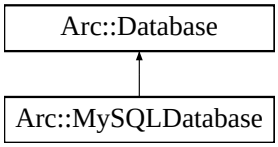
- Credential.h

5.26 Arc::Database Class Reference

Interface for calling database client library.

`#include <DBInterface.h>`

Inheritance diagram for Arc::Database::



Public Member Functions

- **Database** ()
- **Database** (std::string &server, int port)
- **Database** (const **Database** &other)
- virtual ~**Database** ()
- virtual bool **connect** (std::string &dbname, std::string &user, std::string &password)=0
- virtual bool **isconnected** () const =0
- virtual void **close** ()=0
- virtual bool **enable_ssl** (const std::string keyfile="", const std::string certfile="", const std::string cafile="", const std::string capath="")=0
- virtual bool **shutdown** ()=0

5.26.1 Detailed Description

Interface for calling database client library.

For different types of database client library, different classes should be implemented by implementing this interface.

5.26.2 Constructor & Destructor Documentation

5.26.2.1 Arc::Database::Database () `[inline]`

Default constructor

5.26.2.2 Arc::Database::Database (std::string & *server*, int *port*) `[inline]`

Constructor which uses the server's name(or IP address) and port as parametes

5.26.2.3 Arc::Database::Database (const Database & *other*) `[inline]`

Copy constructor

5.26.2.4 virtual Arc::Database::~Database () [inline, virtual]

Deconstructor

5.26.3 Member Function Documentation

5.26.3.1 virtual void Arc::Database::close () [pure virtual]

Close the connection with database server

Implemented in **Arc::MySQLDatabase** (p. 207).

5.26.3.2 virtual bool Arc::Database::connect (std::string & *dbname*, std::string & *user*, std::string & *password*) [pure virtual]

Do connection with database server

Parameters:

dbname The database name which will be used.
user The username which will be used to access database.
password The password which will be used to access database.

Implemented in **Arc::MySQLDatabase** (p. 207).

5.26.3.3 virtual bool Arc::Database::enable_ssl (const std::string *keyfile* = "", const std::string *certfile* = "", const std::string *cafile* = "", const std::string *capath* = "") [pure virtual]

Enable ssl communication for the connection

Parameters:

keyfile The location of key file.
certfile The location of certificate file.
cafile The location of ca file.
capath The location of ca directory

Implemented in **Arc::MySQLDatabase** (p. 208).

5.26.3.4 virtual bool Arc::Database::isconnected () const [pure virtual]

Get the connection status

Implemented in **Arc::MySQLDatabase** (p. 208).

5.26.3.5 virtual bool Arc::Database::shutdown () [pure virtual]

Ask database server to shutdown

Implemented in **Arc::MySQLDatabase** (p. 208).

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

- DBInterface.h

5.27 Arc::DataBuffer Class Reference

Represents set of buffers.

`#include <DataBuffer.h>`

Data Structures

- struct **buf_desc**
- struct **checksum_desc**

Public Member Functions

- **operator bool** () const
- **DataBuffer** (unsigned int size=65536, int blocks=3)
- **DataBuffer** (**CheckSum** *cksum, unsigned int size=65536, int blocks=3)
- ~**DataBuffer** ()
- bool **set** (**CheckSum** *cksum=NULL, unsigned int size=65536, int blocks=3)
- int **add** (**CheckSum** *cksum)
- 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_any** ()
- bool **wait_used** ()
- bool **checksum_valid** () const
- const **CheckSum** * **checksum_object** () const
- 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** () const

Data Fields

- **DataSpeed speed**

5.27.1 Detailed Description

Represents set of buffers.

This class is used used during data transfer using **DataPoint** (p. 91) classes.

5.27.2 Constructor & Destructor Documentation

5.27.2.1 Arc::DataBuffer::DataBuffer (unsigned int *size* = 65536, int *blocks* = 3)

Contructor

Parameters:

size size of every buffer in bytes.

blocks number of buffers.

5.27.2.2 Arc::DataBuffer::DataBuffer (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 **DataBuffer** (p. 78) itself.

5.27.2.3 Arc::DataBuffer::~DataBuffer ()

Destructor.

5.27.3 Member Function Documentation

5.27.3.1 int Arc::DataBuffer::add (CheckSum * *cksum*)

Add a checksum object which will compute checksum of buffer.

Parameters:

cksum object which will compute checksum. Should not be destroyed till **DataBuffer** (p. 78) itself.

Returns:

integer position in the list of checksum objects.

5.27.3.2 unsigned int Arc::DataBuffer::buffer_size () const

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

5.27.3.3 const CheckSum* Arc::DataBuffer::checksum_object () const

Returns **CheckSum** (p. 53) object specified in constructor, returns NULL if index is not in list.

Parameters:

index of the checksum in question.

5.27.3.4 bool Arc::DataBuffer::checksum_valid () const

Returns true if checksum was successfully computed, returns false if index is not in list.

Parameters:

index of the checksum in question.

5.27.3.5 unsigned long long int Arc::DataBuffer::eof_position () const [inline]

Returns offset following last piece of data transfered.

5.27.3.6 bool Arc::DataBuffer::eof_read ()

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

5.27.3.7 void Arc::DataBuffer::eof_read (bool *v*)

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

5.27.3.8 bool Arc::DataBuffer::eof_write ()

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

5.27.3.9 void Arc::DataBuffer::eof_write (bool *v*)

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

5.27.3.10 bool Arc::DataBuffer::error ()

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

5.27.3.11 bool Arc::DataBuffer::error_read ()

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

5.27.3.12 void Arc::DataBuffer::error_read (bool *v*)

Informs object if error accured on 'read' side.

Parameters:

v true if error.

5.27.3.13 bool Arc::DataBuffer::error_transfer ()

Returns true if eror occured inside object.

5.27.3.14 bool Arc::DataBuffer::error_write ()

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

5.27.3.15 void Arc::DataBuffer::error_write (bool *v*)

Informs object if error accured on 'write' side.

Parameters:

v true if error.

5.27.3.16 bool Arc::DataBuffer::for_read ()

Check if there are buffers which can be taken by **for_read()** (p. 81). This function checks only for buffers and does not take eof and error conditions into account.

5.27.3.17 bool Arc::DataBuffer::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

5.27.3.18 bool Arc::DataBuffer::for_write ()

Check if there are buffers which can be taken by **for_write()** (p. 81). This function checks only for buffers and does not take eof and error conditions into account.

5.27.3.19 bool Arc::DataBuffer::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.

5.27.3.20 bool Arc::DataBuffer::is_notwritten (char * *buf*)

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

Parameters:

buf - address of buffer

5.27.3.21 bool Arc::DataBuffer::is_notwritten (int *handle*)

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

Parameters:

handle buffer's number.

5.27.3.22 bool Arc::DataBuffer::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.

5.27.3.23 bool Arc::DataBuffer::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.

5.27.3.24 bool Arc::DataBuffer::is_written (char * *buf*)

Informs object that data was written from buffer.

Parameters:

buf - address of buffer

5.27.3.25 bool Arc::DataBuffer::is_written (int *handle*)

Informs object that data was written from buffer.

Parameters:

handle buffer's number.

5.27.3.26 Arc::DataBuffer::operator bool (void) const [inline]

Check if **DataBuffer** (p. 78) object is initialized.

5.27.3.27 char* Arc::DataBuffer::operator[ ] (int *n*)

Direct access to buffer by number.

5.27.3.28 bool Arc::DataBuffer::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 **DataBuffer** (p. 78) itself.

5.27.3.29 bool Arc::DataBuffer::wait_any ()

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

5.27.3.30 bool Arc::DataBuffer::wait_eof ()

Wait till end of transfer happens on any side.

5.27.3.31 bool Arc::DataBuffer::wait_eof_read ()

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

5.27.3.32 bool Arc::DataBuffer::wait_eof_write ()

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

5.27.3.33 bool Arc::DataBuffer::wait_read ()

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

5.27.3.34 bool Arc::DataBuffer::wait_used ()

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

5.27.3.35 bool Arc::DataBuffer::wait_write ()

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

5.27.4 Field Documentation

5.27.4.1 DataSpeed Arc::DataBuffer::speed

This object controls transfer speed.

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

- DataBuffer.h

5.28 Arc::DataCallback Class Reference

`#include <DataCallback.h>`

5.28.1 Detailed Description

This class is used by **DataHandle** (p. 86) 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

5.29 Arc::DataHandle Class Reference

This class is a wrapper around the **DataPoint** (p. 91) class.

`#include <DataHandle.h>`

5.29.1 Detailed Description

This class is a wrapper around the **DataPoint** (p. 91) class.

It simplifies the construction, use and destruction of **DataPoint** (p. 91) objects.

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

- DataHandle.h

5.30 Arc::DataMover Class Reference

`#include <DataMover.h>`

Public Member Functions

- **DataMover** ()
- ~**DataMover** ()
- DataStatus **Transfer** (**DataPoint** &source, **DataPoint** &destination, **FileCache** &cache, const URLMap &map, std::string &failure_description, callback cb=NULL, void *arg=NULL, const char *prefix=NULL)
- DataStatus **Transfer** (**DataPoint** &source, **DataPoint** &destination, **FileCache** &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)
- 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)

5.30.1 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** (p. 89). Instance represents only attributes used during transfer.

5.30.2 Constructor & Destructor Documentation

5.30.2.1 Arc::DataMover::DataMover ()

Constructor.

5.30.2.2 Arc::DataMover::~DataMover ()

Destructor.

5.30.3 Member Function Documentation

5.30.3.1 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).

5.30.3.2 bool Arc::DataMover::checks ()

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

5.30.3.3 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.

5.30.3.4 void Arc::DataMover::passive (bool)

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

5.30.3.5 void Arc::DataMover::retry (bool)

Set if transfer will be retried in case of failure.

5.30.3.6 bool Arc::DataMover::retry ()

Check if transfer will be retried in case of failure.

5.30.3.7 void Arc::DataMover::secure (bool)

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

5.30.3.8 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** (p. 114) class.

5.30.3.9 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** (p. 114) class.

5.30.3.10 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** (p. 114) class.

5.30.3.11 DataStatus Arc::DataMover::Transfer (DataPoint & *source*, DataPoint & *destination*, FileCache & *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.

5.30.3.12 DataStatus Arc::DataMover::Transfer (DataPoint & *source*, DataPoint & *destination*, FileCache & *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** (p. 286).

destination destination **URL** (p. 286).

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

map **URL** (p. 286) mapping/convertion table (for 'source' **URL** (p. 286)).

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.

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

Activate printing information about transfer status.

Parameters:

prefix use this string if 'prefix' in **DataMover::Transfer** (p. 89) is NULL.

5.30.3.14 void Arc::DataMover::verbose (bool)

Activate printing information about transfer status.

5.30.3.15 bool Arc::DataMover::verbose ()

Check if printing information about transfer status is activated.

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

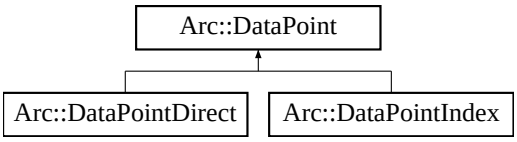
- DataMover.h

5.31 Arc::DataPoint Class Reference

This base class is an abstraction of **URL** (p. 286).

`#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** (**DataBuffer** &buffer)=0
- virtual DataStatus **StartWriting** (**DataBuffer** &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 long_list=false, bool resolve=false, bool metadata=false)=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 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
- void **AssignCredentials** (const std::string &proxyPath, const std::string &certifcatePath, const std::string &keyPath, const std::string &caCertificatesDir)
- void **AssignCredentials** (const **XMLNode** &node)
- void **ApplySecurity** (MCCConfig &cfg) const

5.31.1 Detailed Description

This base class is an abstraction of **URL** (p. 286).

Specializations should be provided for different kind of direct access URLs (`file://`, `ftp://`, gsiftp://, `http://`, `https://`, httpg://, ...) or indexing service URLs (rls://, lfc://, ...). **DataPoint** (p. 91) provides means to resolve an indexing service **URL** (p. 286) into multiple URLs and to loop through them.

5.31.2 Constructor & Destructor Documentation

5.31.2.1 Arc::DataPoint::DataPoint (const URL & *url*)

Constructor requires **URL** (p. 286) to be provided.

5.31.2.2 virtual Arc::DataPoint::~DataPoint () `[virtual]`

Destructor.

5.31.3 Member Function Documentation

5.31.3.1 virtual bool Arc::DataPoint::AcceptsMeta () [pure virtual]

If endpoint can have any use from meta information.

Implemented in **Arc::DataPointDirect** (p. 103), and **Arc::DataPointIndex** (p. 109).

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

Add **URL** (p. 286) to list.

Parameters:

url Location **URL** (p. 286) to add.

meta Location meta information.

Implemented in **Arc::DataPointDirect** (p. 103), and **Arc::DataPointIndex** (p. 109).

5.31.3.3 void Arc::DataPoint::ApplySecurity (MCCConfig & *cfg*) const [inline]

Apply authentication credentials.

This method applies the member credentials to the passed MCCConfig object reference.

Parameters:

cfg The member credentials are applied to this object reference.

5.31.3.4 void Arc::DataPoint::AssignCredentials (const XMLNode & *node*)

Assing credentials used for authentication (using XML node).

5.31.3.5 void Arc::DataPoint::AssignCredentials (const std::string & *proxyPath*, const std::string & *certifcatePath*, const std::string & *keyPath*, const std::string & *caCertificatesDir*)

Assing credentials used for authentication.

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

Get suggested number of buffers for transfers.

Implemented in **Arc::DataPointDirect** (p. 103), and **Arc::DataPointIndex** (p. 109).

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

Get suggested buffer size for transfers.

Implemented in **Arc::DataPointDirect** (p. 103), and **Arc::DataPointIndex** (p. 109).

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

Returns true if file is cacheable.

Implemented in **Arc::DataPointDirect** (p. 103), and **Arc::DataPointIndex** (p. 109).

5.31.3.9 virtual DataStatus Arc::DataPoint::Check () [pure virtual]

Query the **DataPoint** (p. 91) to check if object is accessible.

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

Implemented in **Arc::DataPointIndex** (p. 109).

5.31.3.10 virtual bool Arc::DataPoint::CheckCheckSum () const [virtual]

Check if meta-information 'checksum' is available.

5.31.3.11 virtual bool Arc::DataPoint::CheckCreated () const [virtual]

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

5.31.3.12 virtual bool Arc::DataPoint::CheckSize () const [virtual]

Check if meta-information 'size' is available.

5.31.3.13 virtual bool Arc::DataPoint::CheckValid () const [virtual]

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

5.31.3.14 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.

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

Returns current (resolved) **URL** (p. 286).

Implemented in **Arc::DataPointDirect** (p. 103), and **Arc::DataPointIndex** (p. 110).

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

Returns meta information used to create current **URL** (p. 286).

Usage differs between different indexing services.

Implemented in **Arc::DataPointDirect** (p. 104), and **Arc::DataPointIndex** (p. 110).

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

Check if additional checks before will be performed.

Implemented in **Arc::DataPointDirect** (p. 104), and **Arc::DataPointIndex** (p. 110).

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

Get value of meta-information 'checksum'.

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

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

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

Check if heavy security during data transfer is allowed.

Implemented in **Arc::DataPointDirect** (p. 104), and **Arc::DataPointIndex** (p. 110).

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

Get value of meta-information 'size'.

5.31.3.22 virtual int Arc::DataPoint::GetTries () const [virtual]

Returns number of retries left.

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

Returns the **URL** (p. 286) that was passed to the constructor.

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

Get value of meta-information 'validity time'.

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

Returns true if number of resolved URLs is not 0.

Implemented in **Arc::DataPointDirect** (p. 104), and **Arc::DataPointIndex** (p. 110).

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

Check if **URL** (p. 286) is an Indexing **Service** (p. 270).

Implemented in **Arc::DataPointDirect** (p. 104), and **Arc::DataPointIndex** (p. 110).

5.31.3.27 virtual DataStatus Arc::DataPoint::ListFiles (std::list< FileInfo > & *files*, bool *long_list* = false, bool *resolve* = false, bool *metadata* = false) [pure virtual]

List file(s).

If the **DataPoint** (p. 91) represents a directory its contents will be listed.

Parameters:

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

long)list if true, list additional properties of each file.

resolve if true, resolve physical locations (relevant for indexing services only).

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

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

Implemented in **Arc::DataPointDirect** (p. 104), and **Arc::DataPointIndex** (p. 110).

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

Returns false if out of retries.

Implemented in **Arc::DataPointDirect** (p. 104), and **Arc::DataPointIndex** (p. 110).

5.31.3.30 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** (p. 104), and **Arc::DataPointIndex** (p. 111).

5.31.3.31 virtual Arc::DataPoint::operator bool () const [virtual]

Is **DataPoint** (p. 91) valid?

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

Is **DataPoint** (p. 91) valid?

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

Request passive transfers for FTP-like protocols.

Parameters:

true to request.

Implemented in **Arc::DataPointDirect** (p. 105), and **Arc::DataPointIndex** (p. 111).

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

Index **Service** (p. 270) postregistration.

Used for same purpose as 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** (p. 270) under same name.

Implemented in **Arc::DataPointDirect** (p. 105).

5.31.3.35 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** (p. 270).

Implemented in **Arc::DataPointDirect** (p. 105).

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

Index **Service** (p. 270) 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** (p. 270) under same name.

Implemented in **Arc::DataPointDirect** (p. 105).

5.31.3.37 virtual bool Arc::DataPoint::ProvidesMeta () [pure virtual]

If endpoint can provide at least some meta information directly.

Implemented in **Arc::DataPointDirect** (p. 106), and **Arc::DataPointIndex** (p. 111).

5.31.3.38 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** (p. 106), and **Arc::DataPointIndex** (p. 111).

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

Allow/disallow **DataPoint** (p. 91) to produce scattered data during reading∗ operation.

Parameters:

v true if allowed (default is false).

Implemented in **Arc::DataPointDirect** (p. 106), and **Arc::DataPointIndex** (p. 111).

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

Check if file is registered in Indexing **Service** (p. 270).

Proper value is obtainable only after Resolve.

Implemented in **Arc::DataPointDirect** (p. 106), and **Arc::DataPointIndex** (p. 111).

5.31.3.41 virtual DataStatus Arc::DataPoint::Remove () [pure virtual]

Remove/delete object at **URL** (p. 286).

Implemented in **Arc::DataPointIndex** (p. 111).

5.31.3.42 virtual DataStatus Arc::DataPoint::RemoveLocation () [pure virtual]

Remove current **URL** (p. 286) from list.

Implemented in **Arc::DataPointDirect** (p. 106), and **Arc::DataPointIndex** (p. 112).

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

Remove locations present in another **DataPoint** (p. 91) object.

Implemented in **Arc::DataPointDirect** (p. 106), and **Arc::DataPointIndex** (p. 112).

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

Resolves index service **URL** (p. 286) into list of ordinary URLs.

Also obtains meta information about the file.

Parameters:

source true if **DataPoint** (p. 91) object represents source of information.

Implemented in **Arc::DataPointDirect** (p. 106).

5.31.3.45 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** (p. 107), and **Arc::DataPointIndex** (p. 112).

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

Set value of meta-information 'checksum'.

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

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

5.31.3.48 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.

5.31.3.49 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** (p. 107), and **Arc::DataPointIndex** (p. 112).

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

Set value of meta-information 'size'.

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

Set number of retries.

Reimplemented in **Arc::DataPointIndex** (p. 112).

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

Set value of meta-information 'validity time'.

5.31.3.53 virtual DataStatus Arc::DataPoint::StartReading (DataBuffer & *buffer*) [pure virtual]

Start reading data from **URL** (p. 286).

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** (p. 112).

5.31.3.54 virtual DataStatus Arc::DataPoint::StartWriting (DataBuffer & *buffer*, DataCallback * *space_cb* = NULL) [pure virtual]

Start writing data to **URL** (p. 286).

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** (p. 113).

5.31.3.55 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** (p. 113).

5.31.3.56 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** (p. 113).

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

Returns a string representation of the **DataPoint** (p. 91).

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

Index **Service** (p. 270) unregistration.

Remove information about file registered in Indexing **Service** (p. 270).

Parameters:

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

Implemented in **Arc::DataPointDirect** (p. 107).

5.31.3.59 virtual bool Arc::DataPoint::WriteOutOfOrder () [pure virtual]

Returns true if **URL** (p. 286) can accept scattered data for ∗writing∗ operation.

Implemented in **Arc::DataPointDirect** (p. 107), and **Arc::DataPointIndex** (p. 113).

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

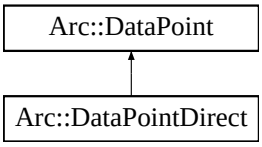
- DataPoint.h

5.32 Arc::DataPointDirect Class Reference

This is a kind of generalized file handle.

`#include <DataPointDirect.h>`

Inheritance diagram for Arc::DataPointDirect::



Public Member Functions

- virtual bool **IsIndex** () const
- virtual unsigned long long int **BufSize** () const
- virtual int **BufNum** () const
- virtual bool **Cache** () const
- virtual bool **Local** () 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)

5.32.1 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 **DataBuffer** (p. 78) 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.

5.32.2 Member Function Documentation

5.32.2.1 virtual bool Arc::DataPointDirect::AcceptsMeta () [virtual]

If endpoint can have any use from meta information.

Implements **Arc::DataPoint** (p. 93).

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

Add **URL** (p. 286) to list.

Parameters:

url Location **URL** (p. 286) to add.

meta Location meta information.

Implements **Arc::DataPoint** (p. 93).

5.32.2.3 virtual int Arc::DataPointDirect::BufNum () const [virtual]

Get suggested number of buffers for transfers.

Implements **Arc::DataPoint** (p. 93).

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

Get suggested buffer size for transfers.

Implements **Arc::DataPoint** (p. 93).

5.32.2.5 virtual bool Arc::DataPointDirect::Cache () const [virtual]

Returns true if file is cacheable.

Implements **Arc::DataPoint** (p. 94).

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

Returns current (resolved) **URL** (p. 286).

Implements **Arc::DataPoint** (p. 94).

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

Returns meta information used to create current **URL** (p. 286).

Usage differs between different indexing services.

Implements **Arc::DataPoint** (p. 94).

5.32.2.8 virtual bool Arc::DataPointDirect::GetAdditionalChecks () const [virtual]

Check if additional checks before will be performed.

Implements **Arc::DataPoint** (p. 95).

5.32.2.9 virtual bool Arc::DataPointDirect::GetSecure () const [virtual]

Check if heavy security during data transfer is allowed.

Implements **Arc::DataPoint** (p. 95).

5.32.2.10 virtual bool Arc::DataPointDirect::HaveLocations () const [virtual]

Returns true if number of resolved URLs is not 0.

Implements **Arc::DataPoint** (p. 95).

5.32.2.11 virtual bool Arc::DataPointDirect::IsIndex () const [virtual]

Check if **URL** (p. 286) is an Indexing **Service** (p. 270).

Implements **Arc::DataPoint** (p. 96).

5.32.2.12 virtual bool Arc::DataPointDirect::Local () const [virtual]

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

Implements **Arc::DataPoint** (p. 96).

5.32.2.13 virtual bool Arc::DataPointDirect::LocationValid () const [virtual]

Returns false if out of retries.

Implements **Arc::DataPoint** (p. 96).

5.32.2.14 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** (p. 96).

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

Request passive transfers for FTP-like protocols.

Parameters:

true to request.

Implements **Arc::DataPoint** (p. 96).

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

Index **Service** (p. 270) postregistration.

Used for same purpose as 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** (p. 270) under same name.

Implements **Arc::DataPoint** (p. 97).

5.32.2.17 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** (p. 270).

Implements **Arc::DataPoint** (p. 97).

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

Index **Service** (p. 270) 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** (p. 270) under same name.

Implements **Arc::DataPoint** (p. 97).

5.32.2.19 virtual bool Arc::DataPointDirect::ProvidesMeta () [virtual]

If endpoint can provide at least some meta information directly.

Implements **Arc::DataPoint** (p. 97).

5.32.2.20 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** (p. 98).

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

Allow/disallow **DataPoint** (p. 91) to produce scattered data during reading∗ operation.

Parameters:

v true if allowed (default is false).

Implements **Arc::DataPoint** (p. 98).

5.32.2.22 virtual bool Arc::DataPointDirect::Registered () const [virtual]

Check if file is registered in Indexing **Service** (p. 270).

Proper value is obtainable only after Resolve.

Implements **Arc::DataPoint** (p. 98).

5.32.2.23 virtual DataStatus Arc::DataPointDirect::RemoveLocation () [virtual]

Remove current **URL** (p. 286) from list.

Implements **Arc::DataPoint** (p. 98).

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

Remove locations present in another **DataPoint** (p. 91) object.

Implements **Arc::DataPoint** (p. 98).

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

Resolves index service **URL** (p. 286) into list of ordinary URLs.

Also obtains meta information about the file.

Parameters:

source true if **DataPoint** (p. 91) object represents source of information.

Implements **Arc::DataPoint** (p. 98).

5.32.2.26 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** (p. 99).

5.32.2.27 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** (p. 99).

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

Index **Service** (p. 270) unregistration.

Remove information about file registered in Indexing **Service** (p. 270).

Parameters:

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

Implements **Arc::DataPoint** (p. 101).

5.32.2.29 virtual bool Arc::DataPointDirect::WriteOutOfOrder () [virtual]

Returns true if **URL** (p. 286) can accept scattered data for ∗writing∗ operation.

Implements **Arc::DataPoint** (p. 101).

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

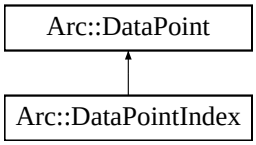
- DataPointDirect.h

5.33 Arc::DataPointIndex Class Reference

Complements **DataPoint** (p. 91) with attributes common for Indexing **Service** (p. 270) URLs.

`#include <DataPointIndex.h>`

Inheritance diagram for Arc::DataPointIndex::



Public Member Functions

- 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 DataStatus **StartReading** (**DataBuffer** &buffer)
- virtual DataStatus **StartWriting** (**DataBuffer** &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**

5.33.1 Detailed Description

Complements **DataPoint** (p. 91) with attributes common for Indexing **Service** (p. 270) URLs.

It should never be used directly. Instead inherit from it to provide a class for specific a Indexing **Service** (p. 270).

5.33.2 Member Function Documentation

5.33.2.1 virtual bool Arc::DataPointIndex::AcceptsMeta () [virtual]

If endpoint can have any use from meta information.

Implements **Arc::DataPoint** (p. 93).

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

Add **URL** (p. 286) to list.

Parameters:

url Location **URL** (p. 286) to add.

meta Location meta information.

Implements **Arc::DataPoint** (p. 93).

5.33.2.3 virtual int Arc::DataPointIndex::BufNum () const [virtual]

Get suggested number of buffers for transfers.

Implements **Arc::DataPoint** (p. 93).

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

Get suggested buffer size for transfers.

Implements **Arc::DataPoint** (p. 93).

5.33.2.5 virtual bool Arc::DataPointIndex::Cache () const [virtual]

Returns true if file is cacheable.

Implements **Arc::DataPoint** (p. 94).

5.33.2.6 virtual DataStatus Arc::DataPointIndex::Check () [virtual]

Query the **DataPoint** (p. 91) to check if object is accessible.

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

Implements **Arc::DataPoint** (p. 94).

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

Returns current (resolved) **URL** (p. 286).

Implements **Arc::DataPoint** (p. 94).

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

Returns meta information used to create current **URL** (p. 286).

Usage differs between different indexing services.

Implements **Arc::DataPoint** (p. 94).

5.33.2.9 virtual bool Arc::DataPointIndex::GetAdditionalChecks () const [virtual]

Check if additional checks before will be performed.

Implements **Arc::DataPoint** (p. 95).

5.33.2.10 virtual bool Arc::DataPointIndex::GetSecure () const [virtual]

Check if heavy security during data transfer is allowed.

Implements **Arc::DataPoint** (p. 95).

5.33.2.11 virtual bool Arc::DataPointIndex::HaveLocations () const [virtual]

Returns true if number of resolved URLs is not 0.

Implements **Arc::DataPoint** (p. 95).

5.33.2.12 virtual bool Arc::DataPointIndex::IsIndex () const [virtual]

Check if **URL** (p. 286) is an Indexing **Service** (p. 270).

Implements **Arc::DataPoint** (p. 96).

5.33.2.13 virtual bool Arc::DataPointIndex::Local () const [virtual]

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

Implements **Arc::DataPoint** (p. 96).

5.33.2.14 virtual bool Arc::DataPointIndex::LocationValid () const [virtual]

Returns false if out of retries.

Implements **Arc::DataPoint** (p. 96).

5.33.2.15 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** (p. 96).

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

Request passive transfers for FTP-like protocols.

Parameters:

true to request.

Implements **Arc::DataPoint** (p. 96).

5.33.2.17 virtual bool Arc::DataPointIndex::ProvidesMeta () [virtual]

If endpoint can provide at least some meta information directly.

Implements **Arc::DataPoint** (p. 97).

5.33.2.18 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** (p. 98).

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

Allow/disallow **DataPoint** (p. 91) to produce scattered data during reading∗ operation.

Parameters:

v true if allowed (default is false).

Implements **Arc::DataPoint** (p. 98).

5.33.2.20 virtual bool Arc::DataPointIndex::Registered () const [virtual]

Check if file is registered in Indexing **Service** (p. 270).

Proper value is obtainable only after Resolve.

Implements **Arc::DataPoint** (p. 98).

5.33.2.21 virtual DataStatus Arc::DataPointIndex::Remove () [virtual]

Remove/delete object at **URL** (p. 286).

Implements **Arc::DataPoint** (p. 98).

5.33.2.22 virtual DataStatus Arc::DataPointIndex::RemoveLocation () [virtual]

Remove current **URL** (p. 286) from list.

Implements **Arc::DataPoint** (p. 98).

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

Remove locations present in another **DataPoint** (p. 91) object.

Implements **Arc::DataPoint** (p. 98).

5.33.2.24 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** (p. 99).

5.33.2.25 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** (p. 99).

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

Set number of retries.

Reimplemented from **Arc::DataPoint** (p. 99).

5.33.2.27 virtual DataStatus Arc::DataPointIndex::StartReading (DataBuffer & *buffer*) [virtual]

Start reading data from **URL** (p. 286).

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** (p. 100).

5.33.2.28 virtual DataStatus Arc::DataPointIndex::StartWriting (DataBuffer & *buffer*, DataCallback * *space_cb* = NULL) [virtual]

Start writing data to **URL** (p. 286).

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** (p. 100).

5.33.2.29 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** (p. 100).

5.33.2.30 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** (p. 100).

5.33.2.31 virtual bool Arc::DataPointIndex::WriteOutOfOrder () [virtual]

Returns true if **URL** (p. 286) can accept scattered data for *writing* operation.

Implements **Arc::DataPoint** (p. 101).

5.33.3 Field Documentation

5.33.3.1 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

5.34 Arc::DataSpeed Class Reference

Keeps track of average and instantaneous transfer speed.

`#include <DataSpeed.h>`

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)

5.34.1 Detailed Description

Keeps track of average and instantaneous transfer speed.

Also detects data transfer inactivity and other transfer timeouts.

5.34.2 Constructor & Destructor Documentation

5.34.2.1 Arc::DataSpeed::DataSpeed (time_t *base* = `DATASPEED_AVERAGING_PERIOD`)

Constructor

Parameters:

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

5.34.2.2 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.

5.34.2.3 Arc::DataSpeed::~DataSpeed (void)

Destructor.

5.34.3 Member Function Documentation

5.34.3.1 void Arc::DataSpeed::hold (bool *disable*)

Turn off speed control.

Parameters:

disable true to turn off.

5.34.3.2 bool Arc::DataSpeed::max_inactivity_time_failure () [inline]

Check if maximal inactivity time error was triggered.

5.34.3.3 bool Arc::DataSpeed::min_average_speed_failure () [inline]

Check if minimal average speed error was triggered.

5.34.3.4 bool Arc::DataSpeed::min_speed_failure () [inline]

Check if minimal speed error was triggered.

5.34.3.5 void Arc::DataSpeed::reset (void)

Reset all counters and triggers.

5.34.3.6 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).

5.34.3.7 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.

5.34.3.8 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.

5.34.3.9 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.

5.34.3.10 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

5.34.3.11 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.

5.34.3.12 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).

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

Returns amount of data this object knows about.

5.34.3.14 bool Arc::DataSpeed::verbose (void)

Check if speed information is going to be printed.

5.34.3.15 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.

5.34.3.16 void Arc::DataSpeed::verbose (bool *val*)

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

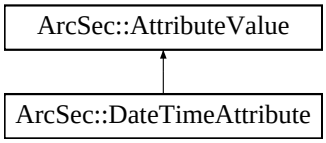
The documentation for this class was generated from the following file:

- DataSpeed.h

5.35 ArcSec::DateTimeAttribute Class Reference

`#include <DateTimeAttribute.h>`

Inheritance diagram for ArcSec::DateTimeAttribute::



Public Member Functions

- virtual bool **equal** (**AttributeValue** *other)
- virtual std::string **encode** ()
- virtual std::string **getType** ()
- virtual std::string **getId** ()

5.35.1 Detailed Description

Format: YYYYMMDDHHMMSSZ Day Month DD HH:MM:SS YYYY YYYY-MM-DD HH:MM:SS YYYY-MM-DDTHH:MM:SS+HH:MM YYYY-MM-DDTHH:MM:SSZ

5.35.2 Member Function Documentation

5.35.2.1 virtual std::string ArcSec::DateTimeAttribute::encode () [virtual]

encode the value in a string format

Implements **ArcSec::AttributeValue** (p. 43).

5.35.2.2 virtual bool ArcSec::DateTimeAttribute::equal (AttributeValue * *value*) [virtual]

Evluate whether "this" equale to the parameter value

Implements **ArcSec::AttributeValue** (p. 43).

5.35.2.3 virtual std::string ArcSec::DateTimeAttribute::getId () [inline, virtual]

Get the identifier of the <Attribute>

Implements **ArcSec::AttributeValue** (p. 43).

5.35.2.4 virtual std::string ArcSec::DateTimeAttribute::getType () [inline, virtual]

Get the type of the <Attribute>

Implements **ArcSec::AttributeValue** (p. 44).

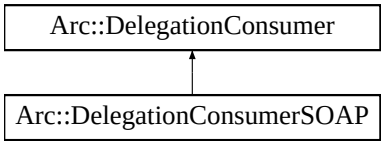
The documentation for this class was generated from the following file:

- DateTimeAttribute.h

5.36 Arc::DelegationConsumer Class Reference

`#include <DelegationInterface.h>`

Inheritance diagram for Arc::DelegationConsumer::



Public Member Functions

- **DelegationConsumer** (void)
- **DelegationConsumer** (const std::string &content)
- const std::string & **ID** (void)
- bool **Backup** (std::string &content)
- bool **Restore** (const std::string &content)
- bool **Request** (std::string &content)
- bool **Acquire** (std::string &content)
- bool **Acquire** (std::string &content, std::string &identity)

Protected Member Functions

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

5.36.1 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()** (p. 121) method for generating certificate request followed by call to **Acquire()** (p. 121) method for making complete credentials from certificate chain.

5.36.2 Constructor & Destructor Documentation

5.36.2.1 Arc::DelegationConsumer::DelegationConsumer (void)

Creates object with new private key

5.36.2.2 Arc::DelegationConsumer::DelegationConsumer (const std::string & *content*)

Creates object with provided private key

5.36.3 Member Function Documentation

5.36.3.1 bool Arc::DelegationConsumer::Acquire (std::string & *content*, std::string & *identity*)

Includes the functionality in Acquire(content); pluse extracting the credential identity

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

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

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

Stores content of this object into a string

5.36.3.4 bool Arc::DelegationConsumer::Generate (void) [protected]

Private key

5.36.3.5 const std::string& Arc::DelegationConsumer::ID (void)

Return identifier of this object - not implemented

5.36.3.6 void Arc::DelegationConsumer::LogError (void) [protected]

Creates private key

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

Make X509 certificate request from internal private key

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

Restores content of object from string

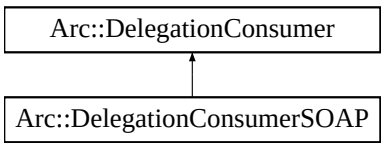
The documentation for this class was generated from the following file:

- DelegationInterface.h

5.37 Arc::DelegationConsumerSOAP Class Reference

`#include <DelegationInterface.h>`

Inheritance diagram for Arc::DelegationConsumerSOAP::



Public Member Functions

- **DelegationConsumerSOAP** (void)
- **DelegationConsumerSOAP** (const std::string &content)
- bool **DelegateCredentialsInit** (const std::string &id, const SOAPEnvelope &in, SOAPEnvelope &out)
- bool **UpdateCredentials** (std::string &credentials, const SOAPEnvelope &in, SOAPEnvelope &out)
- bool **UpdateCredentials** (std::string &credentials, std::string &identity, const SOAPEnvelope &in, SOAPEnvelope &out)
- bool **DelegatedToken** (std::string &credentials, const **XMLNode** &token)

5.37.1 Detailed Description

This class extends **DelegationConsumer** (p. 120) to support SOAP message exchange. Implements WS interface `http://www.nordugrid.org/schemas/delegation` described in delegation.wsdl.

5.37.2 Constructor & Destructor Documentation

5.37.2.1 Arc::DelegationConsumerSOAP::DelegationConsumerSOAP (void)

Creates object with new private key

5.37.2.2 Arc::DelegationConsumerSOAP::DelegationConsumerSOAP (const std::string & *content*)

Creates object with specified private key

5.37.3 Member Function Documentation

5.37.3.1 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.

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

Similar to UpdateCredentials but takes only DelegatedToken XML element

5.37.3.3 bool Arc::DelegationConsumerSOAP::UpdateCredentials (std::string & *credentials*, std::string & *identity*, const SOAPEnvelope & *in*, SOAPEnvelope & *out*)

Includes the functionality in above UpdateCredentials method; plus extracting the credential identity

5.37.3.4 bool Arc::DelegationConsumerSOAP::UpdateCredentials (std::string & *credentials*, const SOAPEnvelope & *in*, SOAPEnvelope & *out*)

Accepts delegated credentials. Process 'in' SOAP message and stores full proxy credentials in 'credentials'. 'out' message is genarated for sending to DelagationProviderSOAP.

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

- DelegationInterface.h

5.38 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

- int **max_size_**
- int **max_duration_**
- int **max_usage_**
- bool **context_lock_**
- bool **restricted_**

5.38.1 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** (p. 122) with identifier included in SOAP message used to route execution to one of managed **DelegationConsumerSOAP** (p. 122) instances.

5.38.2 Member Function Documentation

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

See **DelegationConsumerSOAP::DelegateCredentialsInit** (p. 122)

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

See **DelegationConsumerSOAP::DelegatedToken** (p. 123)

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

See **DelegationConsumerSOAP::UpdateCredentials** (p. 123)

5.38.3 Field Documentation

5.38.3.1 bool Arc::DelegationContainerSOAP::context_lock_ [protected]

If true delegation consumer is deleted when connection context is destroyed

5.38.3.2 int Arc::DelegationContainerSOAP::max_duration_ [protected]

Lifetime of unused delegation consumer

5.38.3.3 int Arc::DelegationContainerSOAP::max_size_ [protected]

Max. number of delegation consumers

5.38.3.4 int Arc::DelegationContainerSOAP::max_usage_ [protected]

Max. times same delegation consumer may accept credentials

5.38.3.5 bool Arc::DelegationContainerSOAP::restricted_ [protected]

If true all delegation phases must be performed by same identity

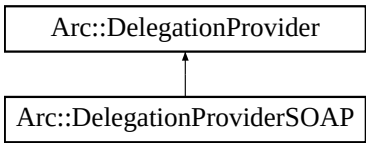
The documentation for this class was generated from the following file:

- DelegationInterface.h

5.39 Arc::DelegationProvider Class Reference

`#include <DelegationInterface.h>`

Inheritance diagram for Arc::DelegationProvider::



Public Member Functions

- **DelegationProvider** (const std::string &credentials)
- **DelegationProvider** (const std::string &cert_file, const std::string &key_file, std::istream *inpwd=NULL)
- std::string **Delegate** (const std::string &request, const DelegationRestrictions &restrictions=DelegationRestrictions())

5.39.1 Detailed Description

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

5.39.2 Constructor & Destructor Documentation

5.39.2.1 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.

5.39.2.2 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.

5.39.3 Member Function Documentation

5.39.3.1 std::string Arc::DelegationProvider::Delegate (const std::string & *request*, const DelegationRestrictions & *restrictions* = `DelegationRestrictions()`)

Perform delegation. Takes X509 certificate request and creates proxy credentials excluding private key. Result is then to be fed into **DelegationConsumer::Acquire** (p. 121)

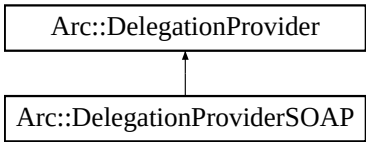
The documentation for this class was generated from the following file:

- DelegationInterface.h

5.40 Arc::DelegationProviderSOAP Class Reference

`#include <DelegationInterface.h>`

Inheritance diagram for Arc::DelegationProviderSOAP::



Public Member Functions

- **DelegationProviderSOAP** (const std::string &credentials)
- **DelegationProviderSOAP** (const std::string &cert_file, const std::string &key_file, std::istream *inpwd=NULL)
- bool **DelegateCredentialsInit** (**MCCInterface** &mcc_interface, **MessageContext** *context)
- bool **DelegateCredentialsInit** (**MCCInterface** &mcc_interface, **MessageAttributes** *attributes_in, **MessageAttributes** *attributes_out, **MessageContext** *context)
- bool **UpdateCredentials** (**MCCInterface** &mcc_interface, **MessageContext** *context, const DelegationRestrictions &restrictions=DelegationRestrictions())
- bool **UpdateCredentials** (**MCCInterface** &mcc_interface, **MessageAttributes** *attributes_in, **MessageAttributes** *attributes_out, **MessageContext** *context, const DelegationRestrictions &restrictions=DelegationRestrictions())
- bool **DelegatedToken** (**XMLNode** &parent)
- const std::string & **ID** (void)

5.40.1 Detailed Description

Extension of **DelegationProvider** (p. 126) with SOAP exchange interface. This class is also a temporary container for intermediate information used during delegation procedure.

5.40.2 Constructor & Destructor Documentation

5.40.2.1 Arc::DelegationProviderSOAP::DelegationProviderSOAP (const std::string & *credentials*)

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

5.40.2.2 Arc::DelegationProviderSOAP::DelegationProviderSOAP (const std::string & *cert_file*, const std::string & *key_file*, std::istream * *inpwd* = NULL)

Creates instance from provided credentials. Credentials are used to sign delegated credentials. Arguments should contain filesystem path to PEM-encoded certificate and private key. Optionally cert_file may contain certificates chain.

5.40.3 Member Function Documentation

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

Extended version of **DelegateCredentialsInit(MCCInterface&,MessageContext*)** (p. 129). Additionally takes attributes for request and response message to make fine control on message processing possible.

5.40.3.2 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.

5.40.3.3 bool Arc::DelegationProviderSOAP::DelegatedToken (XMLNode & *parent*)

Generates DelegatedToken element. Element is created as child of provided XML element and contains structure described in delegation.wsdl.

5.40.3.4 const std::string& Arc::DelegationProviderSOAP::ID (void) [inline]

Returns the identifier by service accepting delegated credentials. This identifier may then be used to refer to credentials stored at service.

5.40.3.5 bool Arc::DelegationProviderSOAP::UpdateCredentials (MCCInterface & *mcc_interface*, MessageAttributes * *attributes_in*, MessageAttributes * *attributes_out*, MessageContext * *context*, const DelegationRestrictions & *restrictions* = DelegationRestrictions())

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

5.40.3.6 bool Arc::DelegationProviderSOAP::UpdateCredentials (MCCInterface & *mcc_interface*, MessageContext * *context*, const DelegationRestrictions & *restrictions* = DelegationRestrictions())

Performs UpdateCredentials SOAP operation. This concludes delegation procedure and passes delagated credentials to **DelegationConsumerSOAP** (p. 122) instance.

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

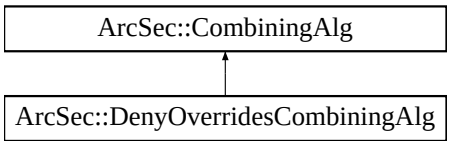
- DelegationInterface.h

5.41 ArcSec::DenyOverridesCombiningAlg Class Reference

Implement the "Deny-Overrides" algorithm.

`#include <DenyOverridesAlg.h>`

Inheritance diagram for ArcSec::DenyOverridesCombiningAlg::



Public Member Functions

- virtual Result **combine** (**EvaluationCtx** *ctx, std::list< **Policy** * > policies)
- virtual const std::string & **getalgId** (void) const

5.41.1 Detailed Description

Implement the "Deny-Overrides" algorithm.

Deny-Overrides, scans the policy set which is given as the parameters of "combine" method, if gets "deny" result from any policy, then stops scanning and gives "deny" as result, otherwise gives "permit".

5.41.2 Member Function Documentation

5.41.2.1 virtual Result ArcSec::DenyOverridesCombiningAlg::combine (EvaluationCtx * *ctx*, std::list< Policy * > *policies*) [virtual]

If there is one policy which return negative evaluation result, then omit the other policies and return DECISION_DENY

Parameters:

ctx This object contains request information which will be used to evaluated against policy.

policlies This is a container which contains policy objects.

Returns:

The combined result according to the algorithm.

Implements **ArcSec::CombiningAlg** (p. 59).

5.41.2.2 virtual const std::string& ArcSec::DenyOverridesCombiningAlg::getalgId (void) const [inline, virtual]

Get the identifier

Implements **ArcSec::CombiningAlg** (p. 59).

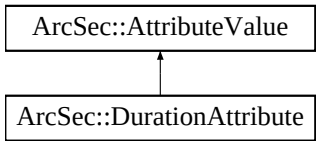
The documentation for this class was generated from the following file:

- DenyOverridesAlg.h

5.42 ArcSec::DurationAttribute Class Reference

`#include <DateTimeAttribute.h>`

Inheritance diagram for ArcSec::DurationAttribute::



Public Member Functions

- virtual bool **equal** (**AttributeValue** *other)
- virtual std::string **encode** ()
- virtual std::string **getType** ()
- virtual std::string **getId** ()

5.42.1 Detailed Description

Formate: P??Y??M??DT??H??M??S

5.42.2 Member Function Documentation

5.42.2.1 virtual std::string ArcSec::DurationAttribute::encode () [virtual]

encode the value in a string format

Implements **ArcSec::AttributeValue** (p. 43).

5.42.2.2 virtual bool ArcSec::DurationAttribute::equal (AttributeValue * *value*) [virtual]

Evluate whether "this" equale to the parameter value

Implements **ArcSec::AttributeValue** (p. 43).

5.42.2.3 virtual std::string ArcSec::DurationAttribute::getId () [inline, virtual]

Get the identifier of the <Attribute>

Implements **ArcSec::AttributeValue** (p. 43).

5.42.2.4 virtual std::string ArcSec::DurationAttribute::getType () [inline, virtual]

Get the type of the <Attribute>

Implements **ArcSec::AttributeValue** (p. 44).

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

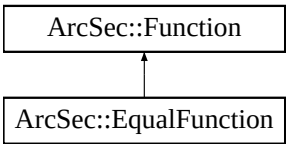
- DateTimeAttribute.h

5.43 ArcSec::EqualFunction Class Reference

Evaluate whether the two values are equal.

`#include <EqualFunction.h>`

Inheritance diagram for ArcSec::EqualFunction::



Public Member Functions

- virtual bool **evaluate** (**AttributeValue** ∗arg0, **AttributeValue** ∗arg1)

Static Public Member Functions

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

5.43.1 Detailed Description

Evaluate whether the two values are equal.

5.43.2 Member Function Documentation

5.43.2.1 virtual bool ArcSec::EqualFunction::evaluate (AttributeValue ∗ *arg0*, AttributeValue ∗ *arg1*) [virtual]

Evaluate two **AttributeValue** (p. 43) objects

Implements **ArcSec::Function** (p. 153).

5.43.2.2 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

5.44 ArcSec::EvalResult Struct Reference

Struct to record the xml node and effect, which will be used by **Evaluator** (p. 137) to get the information about which rule/policy(in xmlnode) is satisfied.

`#include <Result.h>`

5.44.1 Detailed Description

Struct to record the xml node and effect, which will be used by **Evaluator** (p. 137) 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

5.45 ArcSec::EvaluationCtx Class Reference

EvaluationCtx (p. 136), in charge of storing some context information for evaluation, including **Request** (p. 244), current time, etc.

`#include <EvaluationCtx.h>`

Public Member Functions

- **EvaluationCtx** (**Request** *request)
- virtual void **split** ()

5.45.1 Detailed Description

EvaluationCtx (p. 136), in charge of storing some context information for evaluation, including **Request** (p. 244), current time, etc.

5.45.2 Constructor & Destructor Documentation

5.45.2.1 ArcSec::EvaluationCtx::EvaluationCtx (Request * *request*)

Construct a new **EvaluationCtx** (p. 136) based on the given request

5.45.3 Member Function Documentation

5.45.3.1 virtual void ArcSec::EvaluationCtx::split () `[virtual]`

Convert/split one **RequestItem** (p. 247) (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** (p. 248) one by one, not the **RequestItem** (p. 247) 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:

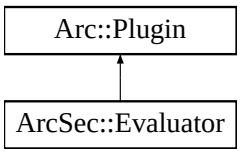
- EvaluationCtx.h

5.46 ArcSec::Evaluator Class Reference

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

`#include <Evaluator.h>`

Inheritance diagram for ArcSec::Evaluator::



Public Member Functions

- virtual **Response** ∗ **evaluate** (**Request** ∗request)=0
- virtual **Response** ∗ **evaluate** (const **Source** &request)=0
- virtual **Response** ∗ **evaluate** (**Request** ∗request, const **Source** &policy)=0
- virtual **Response** ∗ **evaluate** (const **Source** &request, const **Source** &policy)=0
- virtual **Response** ∗ **evaluate** (**Request** ∗request, **Policy** ∗policyobj)=0
- virtual **Response** ∗ **evaluate** (const **Source** &request, **Policy** ∗policyobj)=0
- virtual **AttributeFactory** ∗ **getAttrFactory** ()=0
- virtual **FnFactory** ∗ **getFnFactory** ()=0
- virtual **AlgFactory** ∗ **getAlgFactory** ()=0
- virtual void **addPolicy** (const **Source** &policy, const std::string &id="")=0
- virtual void **addPolicy** (**Policy** ∗policy, const std::string &id="")=0
- virtual void **setCombiningAlg** (EvaluatorCombiningAlg alg)=0
- virtual void **setCombiningAlg** (**CombiningAlg** ∗alg=NULL)=0
- virtual const char ∗ **getName** (void) const =0

Protected Member Functions

- virtual **Response** ∗ **evaluate** (**EvaluationCtx** ∗ctx)=0

5.46.1 Detailed Description

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

5.46.2 Member Function Documentation

5.46.2.1 virtual void ArcSec::Evaluator::addPolicy (Policy ∗ *policy*, const std::string & *id* = "") [pure virtual]

Add policy to the evaluator. **Policy** (p. 236) will be marked with id. The policy object is taken over by this instance and will be destroyed in destructor.

5.46.2.2 virtual void ArcSec::Evaluator::addPolicy (const Source & *policy*, const std::string & *id* = "") [pure virtual]

Add policy from specified source to the evaluator. **Policy** (p. 236) will be marked with id.

5.46.2.3 virtual Response* ArcSec::Evaluator::evaluate (EvaluationCtx * *ctx*) [protected, pure virtual]

Evaluate the request by using the **EvaluationCtx** (p. 136) object (which includes the information about request). The ctx is destroyed inside this method (why?!?!?).

5.46.2.4 virtual Response* ArcSec::Evaluator::evaluate (const Source & *request*, Policy * *policyobj*) [pure virtual]

Evaluate the request from specified source against the specified policy. In some implementations all of the existing policie inside the evaluator may be destroyed by this method.

5.46.2.5 virtual Response* ArcSec::Evaluator::evaluate (Request * *request*, Policy * *policyobj*) [pure virtual]

Evaluate the specified request against the specified policy. In some implementations all of the existing policy inside the evaluator may be destroyed by this method.

5.46.2.6 virtual Response* ArcSec::Evaluator::evaluate (const Source & *request*, const Source & *policy*) [pure virtual]

Evaluate the request from specified source against the policy from specified source. In some implementations all of the existing policie inside the evaluator may be destroyed by this method.

5.46.2.7 virtual Response* ArcSec::Evaluator::evaluate (Request * *request*, const Source & *policy*) [pure virtual]

Evaluate the specified request against the policy from specified source. In some implementations all of the existing policies inside the evaluator may be destroyed by this method.

5.46.2.8 virtual Response* ArcSec::Evaluator::evaluate (const Source & *request*) [pure virtual]

Evaluates the request by using a specified source

5.46.2.9 virtual Response* ArcSec::Evaluator::evaluate (Request * *request*) [pure virtual]

Evaluates the request by using a **Request** (p. 244) object. Evaluation is done till at least one of policies is satisfied.

5.46.2.10 virtual AlgFactory* ArcSec::Evaluator::getAlgFactory () [pure virtual]

Get the **AlgFactory** (p. 33) object

Referenced by ArcSec::EvaluatorContext::operator AlgFactory ∗().

5.46.2.11 virtual AttributeFactory∗ ArcSec::Evaluator::getAttrFactory () [pure virtual]

Get the **AttributeFactory** (p. 38) object

Referenced by ArcSec::EvaluatorContext::operator AttributeFactory ∗().

5.46.2.12 virtual FnFactory∗ ArcSec::Evaluator::getFnFactory () [pure virtual]

Get the **FnFactory** (p. 152) object

Referenced by ArcSec::EvaluatorContext::operator FnFactory ∗().

5.46.2.13 virtual const char∗ ArcSec::Evaluator::getName (void) const [pure virtual]

Get the name of this evaluator

5.46.2.14 virtual void ArcSec::Evaluator::setCombiningAlg (CombiningAlg ∗ *alg* = NULL) [pure virtual]

Specifies loadable combining algorithms. In case of multiple policies their results will be combined using this algorithm. To switch to simple algorithm specify NULL argument.

5.46.2.15 virtual void ArcSec::Evaluator::setCombiningAlg (EvaluatorCombiningAlg *alg*) [pure virtual]

Specifies one of simple combining algorithms. In case of multiple policies their results will be combined using this algorithm.

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

- Evaluator.h

5.47 ArcSec::EvaluatorContext Class Reference

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

`#include <Evaluator.h>`

Public Member Functions

- **operator AttributeFactory** ∗ ()
- **operator FnFactory** ∗ ()
- **operator AlgFactory** ∗ ()

5.47.1 Detailed Description

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

5.47.2 Member Function Documentation

5.47.2.1 ArcSec::EvaluatorContext::operator AlgFactory ∗ () [inline]

Returns associated **AlgFactory** (p. 33) object

References ArcSec::Evaluator::getAlgFactory().

5.47.2.2 ArcSec::EvaluatorContext::operator AttributeFactory ∗ () [inline]

Returns associated **AttributeFactory** (p. 38) object

References ArcSec::Evaluator::getAttrFactory().

5.47.2.3 ArcSec::EvaluatorContext::operator FnFactory ∗ () [inline]

Returns associated **FnFactory** (p. 152) object

References ArcSec::Evaluator::getFnFactory().

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

- Evaluator.h

5.48 ArcSec::EvaluatorLoader Class Reference

EvaluatorLoader (p. 141) is implemented as a helper class for loading different **Evaluator** (p. 137) objects, like ArcEvaluator.

`#include <EvaluatorLoader.h>`

Public Member Functions

- **Evaluator** ∗ **getEvaluator** (const std::string &classname)
- **Evaluator** ∗ **getEvaluator** (const **Policy** ∗policy)
- **Evaluator** ∗ **getEvaluator** (const **Request** ∗request)
- **Request** ∗ **getRequest** (const std::string &classname, const **Source** &requestsource)
- **Request** ∗ **getRequest** (const **Source** &requestsource)
- **Policy** ∗ **getPolicy** (const std::string &classname, const **Source** &policysource)
- **Policy** ∗ **getPolicy** (const **Source** &policysource)

5.48.1 Detailed Description

EvaluatorLoader (p. 141) is implemented as a helper class for loading different **Evaluator** (p. 137) objects, like ArcEvaluator.

The object loading is based on the configuration information about evaluator, including information for factory class, request, policy and evaluator itself

5.48.2 Member Function Documentation

5.48.2.1 Evaluator∗ ArcSec::EvaluatorLoader::getEvaluator (const Request ∗ *request*)

Get evaluator object suitable for presented request

5.48.2.2 Evaluator∗ ArcSec::EvaluatorLoader::getEvaluator (const Policy ∗ *policy*)

Get evaluator object suitable for presented policy

5.48.2.3 Evaluator∗ ArcSec::EvaluatorLoader::getEvaluator (const std::string & *classname*)

Get evaluator object according to the class name

5.48.2.4 Policy∗ ArcSec::EvaluatorLoader::getPolicy (const Source & *policysource*)

Get proper policy object according to the policy source

5.48.2.5 Policy∗ ArcSec::EvaluatorLoader::getPolicy (const std::string & *classname*, const Source & *policysource*)

Get policy object according to the class name, based on the policy source

5.48.2.6 Request* ArcSec::EvaluatorLoader::getRequest (const Source & *requestsource*)

Get request object according to the request source

5.48.2.7 Request* ArcSec::EvaluatorLoader::getRequest (const std::string & *classname*, const Source & *requestsource*)

Get request object according to the class name, based on the request source

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

- EvaluatorLoader.h

5.49 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**

5.49.1 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.

5.49.2 Member Function Documentation

5.49.2.1 Glib::TimeVal Arc::ExpirationReminder::getExpiryTime () const

Returns the expiry time.

This method returns the expiry time of the reservation that this **ExpirationReminder** (p. 143) is associated with.

Returns:

The expiry time.

5.49.2.2 Counter::IDType Arc::ExpirationReminder::getReservationID () const

Returns the identification number of the reservation.

This method returns the identification number of the self-expiring reservation that this **ExpirationReminder** (p. 143) is associated with.

Returns:

The identification number.

5.49.2.3 bool Arc::ExpirationReminder::operator< (const ExpirationReminder & *other*) const

Less than operator, compares "soonness".

This is the less than operator for the **ExpirationReminder** (p. 143) 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.

5.49.3 Friends And Related Function Documentation

5.49.3.1 friend class Counter [friend]

The **Counter** (p. 65) class needs to be a friend.

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

- Counter.h

5.50 Arc::FileCache Class Reference

`#include <FileCache.h>`

Public Member Functions

- **FileCache** (std::string cache_path, std::string id, uid_t job_uid, gid_t job_gid)
- **FileCache** (std::vector< std::string > caches, std::string id, uid_t job_uid, gid_t job_gid)
- **FileCache** (const **FileCache** &cache)
- **FileCache** ()
- virtual ~**FileCache** (void)
- bool **Start** (std::string url, bool &available, bool &is_locked)
- bool **Stop** (std::string url)
- bool **StopAndDelete** (std::string url)
- std::string **File** (std::string url)
- bool **Link** (std::string link_path, std::string url)
- bool **Copy** (std::string dest_path, std::string url, bool executable=false)
- bool **Clean** (unsigned long long int size=1)
- bool **Release** ()
- bool **AddDN** (std::string url, std::string DN, **Time** expiry_time)
- bool **CheckDN** (std::string url, std::string DN)
- bool **CheckCreated** (std::string url)
- **Time GetCreated** (std::string url)
- bool **CheckValid** (std::string url)
- **Time GetValid** (std::string url)
- bool **SetValid** (std::string url, **Time** val)
- **operator bool** ()
- bool **operator==** (const **FileCache** &a)

5.50.1 Detailed Description

FileCache (p. 145) provides an interface to all cache operations to be used by external classes. An instance should be created per job, and all files within the job are managed by that instance. When it is decided a file should be downloaded to the cache, **Start()** (p. 149) should be called, so that the cache file can be prepared and locked. When a transfer has finished successfully, **Link()** (p. 148) or **Copy()** (p. 147) should be called to create a hard link to a per-job directory in the cache and then soft link, or copy the file directly to the session directory so it can be accessed from the user's job. **Stop()** (p. 149) must then be called to release any locks on the cache file.

The cache directory(ies) and the optional directory to link to when the soft-links are made are set in the global configuration file. The names of cache files are formed from a hash of the **URL** (p. 286) specified as input to the job. To ease the load on the file system, the cache files are split into subdirectories based on the first two characters in the hash. For example the file with hash 76f11edda169848038efbd9fa3df5693 is stored in 76/f11edda169848038efbd9fa3df5693. A cache filename can be found by passing the **URL** (p. 286) to Find(). For more information on the structure of the cache, see the Grid Manager Administration Guide.

A metadata file with the '.meta' suffix is stored next to each cache file. This contains the **URL** (p. 286) corresponding to the cache file and the expiry time, if it is available. For example lfc://lfc1.ndgf.org//grid/atlas/test/test1 20081007151045Z

While cache files are downloaded, they are locked by creating a lock file with the '.lock' suffix next to the cache file. Calling **Start()** (p. 149) creates this lock and **Stop()** (p. 149) releases it. All processes calling **Start()** (p. 149) must wait until they successfully obtain the lock before downloading can begin.

5.50.2 Constructor & Destructor Documentation

5.50.2.1 Arc::FileCache::FileCache (std::string *cache_path*, std::string *id*, uid_t *job_uid*, gid_t *job_gid*)

Create a new **FileCache** (p. 145) instance.

Parameters:

cache_path The format is "cache_dir[link_path]". path is the path to the cache directory and the optional link_path is used to create a link in case the cache directory is visible under a different name during actual usage. When linking from the session dir this path is used instead of cache_-path.

id the job id. This is used to create the per-job dir which the job's cache files will be hard linked from

job_uid owner of job. The per-job dir will only be readable by this user

job_gid owner group of job

5.50.2.2 Arc::FileCache::FileCache (std::vector< std::string > *caches*, std::string *id*, uid_t *job_uid*, gid_t *job_gid*)

Create a new **FileCache** (p. 145) instance with multiple cache dirs

Parameters:

caches a vector of strings describing caches. The format of each string is "cache_dir[link_path]".

id the job id. This is used to create the per-job dir which the job's cache files will be hard linked from

job_uid owner of job. The per-job dir will only be readable by this user

job_gid owner group of job

5.50.2.3 Arc::FileCache::FileCache (const FileCache & *cache*)

Copy constructor

5.50.2.4 Arc::FileCache::FileCache () [inline]

Default constructor. Invalid cache.

5.50.2.5 virtual Arc::FileCache::~FileCache (void) [virtual]

Destructor

5.50.3 Member Function Documentation

5.50.3.1 bool Arc::FileCache::AddDN (std::string *url*, std::string *DN*, Time *expiry_time*)

Add the given DN to the list of cached DNs with the given expiry time

Parameters:

url the url corresponding to the cache file to which we want to add a cached DN

DN the DN of the user

expiry_time the expiry time of this DN in the DN cache

5.50.3.2 bool Arc::FileCache::CheckCreated (std::string *url*)

Check if there is an information about creation time. Returns true if the file exists in the cache, since the creation time is the creation time of the cache file.

Parameters:

url the url corresponding to the cache file for which we want to know if the creation date exists

5.50.3.3 bool Arc::FileCache::CheckDN (std::string *url*, std::string *DN*)

Check if the given DN is cached for authorisation.

Parameters:

url the url corresponding to the cache file for which we want to check the cached DN

DN the DN of the user

5.50.3.4 bool Arc::FileCache::CheckValid (std::string *url*)

Check if there is an information about expiry time.

Parameters:

url the url corresponding to the cache file for which we want to know if the expiration time exists

5.50.3.5 bool Arc::FileCache::Clean (unsigned long long int *size* = `1`) `[inline]`

Remove some amount of oldest information from cache. Returns true on success. Not implemented.

Parameters:

size amount to be removed (bytes)

5.50.3.6 bool Arc::FileCache::Copy (std::string *dest_path*, std::string *url*, bool *executable* = `false`)

Copy the cache file corresponding to url to the dest_path

5.50.3.7 std::string Arc::FileCache::File (std::string *url*)

Returns the full pathname of the file in the cache which corresponds to the given url.

5.50.3.8 Time Arc::FileCache::GetCreated (std::string *url*)

Get the creation time of a cached file. If the cache file does not exist, 0 is returned.

Parameters:

url the url corresponding to the cache file for which we want to know the creation date

5.50.3.9 Time Arc::FileCache::GetValid (std::string *url*)

Get expiry time of a cached file. If the time is not available, a time equivalent to 0 is returned.

Parameters:

url the url corresponding to the cache file for which we want to know the expiry time

5.50.3.10 bool Arc::FileCache::Link (std::string *link_path*, std::string *url*)

Create a hard-link to the per-job dir from the cache dir, and then a soft-link from here to the session directory. This is effectively 'claiming' the file for the job, so even if the original cache file is deleted, eg by some external process, the hard link still exists until it is explicitly released by calling **Release()** (p. 148).

If cache_link_path is set to "." then files will be copied directly to the session directory rather than via the hard link.

Parameters:

link_path path to the session dir for soft-link or new file

url url of file to link to or copy

5.50.3.11 Arc::FileCache::operator bool (void) [inline]

Returns true if object is useable.

5.50.3.12 bool Arc::FileCache::operator== (const FileCache & *a*)

Return true if all attributes are equal

5.50.3.13 bool Arc::FileCache::Release ()

Release claims on input files for the job specified by id. For each cache directory the per-job directory with the hard-links will be deleted.

5.50.3.14 bool Arc::FileCache::SetValid (std::string *url*, Time *val*)

Set expiry time.

Parameters:

url the url corresponding to the cache file for which we want to set the expiry time

val expiry time

5.50.3.15 bool Arc::FileCache::Start (std::string *url*, bool & *available*, bool & *is_locked*)

Prepare cache for downloading file, and lock the cached file. On success returns true. If there is another process downloading the same url, false is returned and is_locked is set to true. In this case the client should wait and retry later. If the lock has expired this process will take over the lock and the method will return as if no lock was present, ie available and is_locked are false.

Parameters:

url url that is being downloaded

available true on exit if the file is already in cache

is_locked true on exit if the file is already locked, ie cannot be used by this process

5.50.3.16 bool Arc::FileCache::Stop (std::string *url*)

This method (or stopAndDelete) must be called after file was downloaded or download failed, to release the lock on the cache file. **Stop()** (p. 149) does not delete the cache file. It returns false if the lock file does not exist, or another pid was found inside the lock file (this means another process took over the lock so this process must go back to **Start()** (p. 149)), or if it fails to delete the lock file.

Parameters:

url the url of the file that was downloaded

5.50.3.17 bool Arc::FileCache::StopAndDelete (std::string *url*)

Release the cache file and delete it, because for example a failed download left an incomplete copy, or it has expired. This method also deletes the meta file which contains the url corresponding to the cache file. The logic of the return value is the same as **Stop()** (p. 149).

Parameters:

url the url corresponding to the cache file that has to be released and deleted

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

- FileCache.h

5.51 FileCacheHash Class Reference

`#include <FileCacheHash.h>`

Static Public Member Functions

- static std::string **getHash** (std::string url)
- static int **maxLength** ()

5.51.1 Detailed Description

FileCacheHash (p. 150) provides methods to make hashes from strings. Currently the md5 hash from the openssl library is used.

5.51.2 Member Function Documentation

5.51.2.1 static std::string FileCacheHash::getHash (std::string *url*) [static]

Return a hash of the given URL, according to the current hash scheme.

5.51.2.2 static int FileCacheHash::maxLength () [inline, static]

Return the maximum length of a hash string.

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

- FileCacheHash.h

5.52 Arc::FileInfo Class Reference

FileInfo (p. 151) stores information about files (metadata).

`#include <FileInfo.h>`

5.52.1 Detailed Description

FileInfo (p. 151) stores information about files (metadata).

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

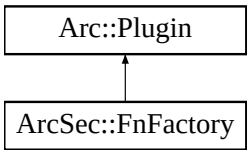
- FileInfo.h

5.53 ArcSec::FnFactory Class Reference

Interface for function factory class.

`#include <FnFactory.h>`

Inheritance diagram for ArcSec::FnFactory::



Public Member Functions

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

5.53.1 Detailed Description

Interface for function factory class.

FnFactory (p. 152) is in charge of creating **Function** (p. 153) object according to the algorithm type given as argument of method createFn. This class can be inherited for implementing a factory class which can create some specific **Function** (p. 153) objects.

5.53.2 Member Function Documentation

5.53.2.1 virtual Function* ArcSec::FnFactory::createFn (const std::string & *type*) [pure virtual]

creat algorithm object based on the type algorithm type

Parameters:

type The type of **Function** (p. 153)

Returns:

The object of **Function** (p. 153)

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

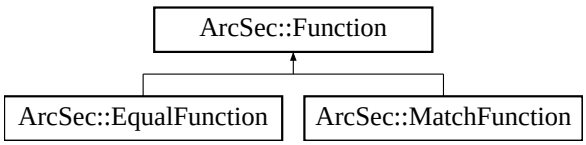
- FnFactory.h

5.54 ArcSec::Function Class Reference

Interface for function, which is in charge of evaluating two **AttributeValue** (p. 43).

`#include <Function.h>`

Inheritance diagram for ArcSec::Function::



Public Member Functions

- virtual bool **evaluate** (**AttributeValue** ∗arg0, **AttributeValue** ∗arg1)=0

5.54.1 Detailed Description

Interface for function, which is in charge of evaluating two **AttributeValue** (p. 43).

5.54.2 Member Function Documentation

5.54.2.1 virtual bool ArcSec::Function::evaluate (AttributeValue ∗ *arg0*, AttributeValue ∗ *arg1*) [pure virtual]

Evaluate two **AttributeValue** (p. 43) objects

Implemented in **ArcSec::EqualFunction** (p. 134), and **ArcSec::MatchFunction** (p. 180).

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

- Function.h

5.55 Arc::InfoCache Class Reference

Stores XML document in filesystem split into parts.

`#include <InfoCache.h>`

Public Member Functions

- **InfoCache** (const **Config** &cfg, const std::string &service_id)

5.55.1 Detailed Description

Stores XML document in filesystem split into parts.

5.55.2 Constructor & Destructor Documentation

5.55.2.1 Arc::InfoCache::InfoCache (const Config & *cfg*, const std::string & *service_id*)

Creates object according to configuration (see InfoCacheConfig.xsd).

XML configuration is passed in cfg. Argument service_id is used to distiguish between various documents stored under same path - corresponding files will be stored in subdirectory with service_id name.

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

- InfoCache.h

5.56 Arc::InfoFilter Class Reference

Filters information document according to identity of requestor.

`#include <InfoFilter.h>`

Public Member Functions

- **InfoFilter** (**MessageAuth** &id)
- bool **Filter** (**XMLNode** doc) const
- bool **Filter** (**XMLNode** doc, const InfoFilterPolicies &policies, const NS &ns) const

5.56.1 Detailed Description

Filters information document according to identity of requestor.

Identity is compared to policies stored inside information document and external ones. Parts of document which do not pass policy evaluation are removed.

5.56.2 Constructor & Destructor Documentation

5.56.2.1 Arc::InfoFilter::InfoFilter (MessageAuth & *id*)

Creates object and associates identity.

Associated identity is not copied, hence passed argument must not be destroyed while this method is used.

5.56.3 Member Function Documentation

5.56.3.1 bool Arc::InfoFilter::Filter (XMLNode *doc*, const InfoFilterPolicies & *policies*, const NS & *ns*) const

Filter information document according to internal and external policies.

In provided document all policies and nodes which have their policies evaluated to negative result are removed. External policies are provided in policies argument. First element of every pair is XPath defining to which XML node policy must be applied. Second element is policy itself. Argument ns defines XML namespaces for XPath evaluation.

5.56.3.2 bool Arc::InfoFilter::Filter (XMLNode *doc*) const

Filter information document according to internal policies.

In provided document all policies and nodes which have their policies evaluated to negative result are removed.

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

- InfoFilter.h

5.57 Arc::InfoRegister Class Reference

Registration to ISIS interface.

`#include <InfoRegister.h>`

5.57.1 Detailed Description

Registration to ISIS interface.

This class represents service registering to Information Indexing **Service** (p. 270). It does not perform registration itself. It only collects configuration information. Configuration is as described in InfoRegister-Config.xsd for element InfoRegistration.

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

- InfoRegister.h

5.58 Arc::InfoRegisterContainer Class Reference

`#include <InfoRegister.h>`

Public Member Functions

- void **addRegistrars** (**XMLNode** doc)
- void **addService** (**InfoRegister** *reg, const std::list< std::string > &ids, **XMLNode** cfg=**XMLNode**())
- void **removeService** (**InfoRegister** *reg)

5.58.1 Detailed Description

Singleton class for scanning configuration and storing refernces to registration elements.

5.58.2 Member Function Documentation

5.58.2.1 void Arc::InfoRegisterContainer::addRegistrars (XMLNode *doc*)

Adds ISISes to list of handled services.

Supplied configuration document is scanned for **InfoRegistrar** (p. 159) elements and those are turned into **InfoRegistrar** (p. 159) classes for handling connection to ISIS service each.

5.58.2.2 void Arc::InfoRegisterContainer::addService (InfoRegister * *reg*, const std::list< std::string > & *ids*, XMLNode *cfg* = XMLNode())

Adds service to list of handled.

This method must be called first time after last addRegistrar was called - services will be only associated with ISISes which are already added. Argument ids contains list of ISIS identifiers to which service is associated. If ids is empty then service is associated to all ISISes currently added. If argument cfg is available and no ISISes are configured then addRegistrars is called with cfg used as configuration document.

5.58.2.3 void Arc::InfoRegisterContainer::removeService (InfoRegister * *reg*)

This method must be called if service being destroyed.

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

- InfoRegister.h

5.59 Arc::InfoRegisters Class Reference

Handling multiple registrations to ISISes.

`#include <InfoRegister.h>`

Public Member Functions

- **InfoRegisters** (**XMLNode** &cfg, **Service** *service_)

5.59.1 Detailed Description

Handling multiple registrations to ISISes.

5.59.2 Constructor & Destructor Documentation

5.59.2.1 Arc::InfoRegisters::InfoRegisters (XMLNode & *cfg*, Service * *service_*)

Constructor creates **InfoRegister** (p. 156) objects according to configuration.

Inside cfg elements InfoRegistration are found and for each corresponding **InfoRegister** (p. 156) object is created. Those objects are destroyed in destructor of this class.

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

- InfoRegister.h

5.60 Arc::InfoRegistrar Class Reference

Registration process associated with particular ISIS.

`#include <InfoRegister.h>`

Public Member Functions

- void **registration** (void)
- bool **addService** (**InfoRegister** *, **XMLNode** &)
- bool **removeService** (**InfoRegister** *)

5.60.1 Detailed Description

Registration process associated with particular ISIS.

Instance of this class starts thread which takes care passing information about associated services to ISIS service defined in configuration. Configuration is as described in InfoRegister.xsd for element **InfoRegistrar** (p. 159).

5.60.2 Member Function Documentation

5.60.2.1 bool Arc::InfoRegistrar::addService (InfoRegister *, XMLNode &)

Adds new service to list of handled services.

Service (p. 270) is described by it's **InfoRegister** (p. 156) object which must be valid as long as this object is functional.

5.60.2.2 void Arc::InfoRegistrar::registration (void)

Performs registartion in a loop.

Never exits unless there is a critical error or requested by destructor.

5.60.2.3 bool Arc::InfoRegistrar::removeService (InfoRegister *)

Removes service from list of handled services.

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

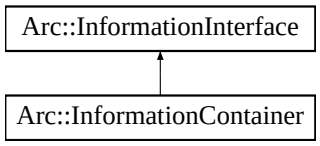
- InfoRegister.h

5.61 Arc::InformationContainer Class Reference

Information System document container and processor.

`#include <InformationInterface.h>`

Inheritance diagram for Arc::InformationContainer::



Public Member Functions

- **InformationContainer** (**XMLNode** doc, bool copy=false)
- **XMLNode Acquire** (void)
- void **Assign** (**XMLNode** doc, bool copy=false)

Protected Member Functions

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

Protected Attributes

- **XMLNode doc_**

5.61.1 Detailed Description

Information System document container and processor.

This class inherits form **InformationInterface** (p. 162) and offers container for storing informational XML document.

5.61.2 Constructor & Destructor Documentation

5.61.2.1 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.

5.61.3 Member Function Documentation

5.61.3.1 XMLNode Arc::InformationContainer::Acquire (void)

Get a lock on contained XML document. To be used in multi-threaded environment. Do not forget to release it with Release()

5.61.3.2 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.

5.61.3.3 virtual void Arc::InformationContainer::Get (const std::list< std::string > & *path*, XMLNodeContainer & *result*) [protected, virtual]

This method is called by this object's Process method. Real implementation of this class should return (sub)tree of XML document. This method may be called multiple times per single Process call. Here is a set on XML element names specifying how to reach requested node(s).

Reimplemented from **Arc::InformationInterface** (p. 162).

5.61.4 Field Documentation

5.61.4.1 XMLNode Arc::InformationContainer::doc_ [protected]

Either link or container of XML document

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

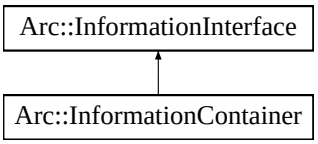
- InformationInterface.h

5.62 Arc::InformationInterface Class Reference

Information System message processor.

`#include <InformationInterface.h>`

Inheritance diagram for Arc::InformationInterface::



Public Member Functions

- **InformationInterface** (bool safe=true)

Protected Member Functions

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

Protected Attributes

- Glib::Mutex **lock_**

5.62.1 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.

5.62.2 Constructor & Destructor Documentation

5.62.2.1 Arc::InformationInterface::InformationInterface (bool *safe* = `true`)

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

5.62.3 Member Function Documentation

5.62.3.1 virtual void Arc::InformationInterface::Get (const std::list< std::string > & *path*, XMLNodeContainer & *result*) `[protected, virtual]`

This method is called by this object's Process method. Real implementation of this class should return (sub)tree of XML document. This method may be called multiple times per single Process call. Here is a set on XML element names specifying how to reach requested node(s).

Reimplemented in **Arc::InformationContainer** (p. 161).

5.62.4 Field Documentation

5.62.4.1 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

5.63 Arc::InformationRequest Class Reference

Request for information in InfoSystem.

`#include <InformationInterface.h>`

Public Member Functions

- **InformationRequest** (void)
- **InformationRequest** (const std::list< std::string > &path)
- **InformationRequest** (const std::list< std::list< std::string > > &paths)
- **InformationRequest** (**XMLNode** query)
- SOAPEnvelope * **SOAP** (void)

5.63.1 Detailed Description

Request for information in InfoSystem.

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

5.63.2 Constructor & Destructor Documentation

5.63.2.1 Arc::InformationRequest::InformationRequest (void)

Dummy constructor

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

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

5.63.2.3 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.

5.63.2.4 Arc::InformationRequest::InformationRequest (XMLNode *query*)

Request for attributes specified by XPath query.

5.63.3 Member Function Documentation

5.63.3.1 SOAPEnvelope* Arc::InformationRequest::SOAP (void)

Returns generated SOAP message

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

- InformationInterface.h

5.64 Arc::InformationResponse Class Reference

Informational response from InfoSystem.

`#include <InformationInterface.h>`

Public Member Functions

- **InformationResponse** (SOAPEnvelope &soap)
- std::list< **XMLNode** > **Result** (void)

5.64.1 Detailed Description

Informational response from InfoSystem.

This is a convenience wrapper analyzing WS-ResourceProperties response from InfoSystem interface of service.

5.64.2 Constructor & Destructor Documentation

5.64.2.1 Arc::InformationResponse::InformationResponse (SOAPEnvelope & *soap*)

Constructor parses WS-ResourceProperties ressponse. Provided SOAPEnvelope object must be valid as long as this object is in use.

5.64.3 Member Function Documentation

5.64.3.1 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:

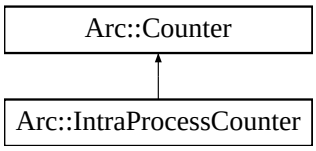
- InformationInterface.h

5.65 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**)

5.65.1 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** (p. 65) class for further information about counters and examples of usage.

5.65.2 Constructor & Destructor Documentation

5.65.2.1 Arc::IntraProcessCounter::IntraProcessCounter (int *limit*, int *excess*)

Creates an **IntraProcessCounter** (p. 166) 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.

5.65.2.2 virtual Arc::IntraProcessCounter::~IntraProcessCounter () [virtual]

Destructor.

This is the destructor of the **IntraProcessCounter** (p. 166) class. Does not need to do anything.

5.65.3 Member Function Documentation

5.65.3.1 virtual void Arc::IntraProcessCounter::cancel (IDType *reservationID*) [protected, virtual]

Cancellation of a reservation.

This method cancels a reservation. It is called by the **CounterTicket** (p. 72) that corresponds to the reservation.

Parameters:

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

5.65.3.2 virtual int Arc::IntraProcessCounter::changeExcess (int *amount*) [virtual]

Changes the excess limit of the counter.

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

Parameters:

amount The amount by which to change the excess limit.

Returns:

The new excess limit.

Implements **Arc::Counter** (p. 67).

5.65.3.3 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** (p. 68).

5.65.3.4 virtual void Arc::IntraProcessCounter::extend (IDType & *reservationID*, Glib::TimeVal & *expiryTime*, Glib::TimeVal *duration* = ETERNAL) [protected, virtual]

Extension of a reservation.

This method extends a reservation. It is called by the **CounterTicket** (p. 72) 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.

5.65.3.5 virtual int Arc::IntraProcessCounter::getExcess () [virtual]

Returns the excess limit of the counter.

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

Returns:

The excess limit.

Implements **Arc::Counter** (p. 69).

5.65.3.6 virtual int Arc::IntraProcessCounter::getLimit () [virtual]

Returns the current limit of the counter.

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

Returns:

The current limit of the counter.

Implements **Arc::Counter** (p. 70).

5.65.3.7 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** (p. 70).

5.65.3.8 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** (p. 72) that can be queried about the status of the reservation as well as for cancellations and extensions.

Implements **Arc::Counter** (p. 70).

5.65.3.9 virtual int Arc::IntraProcessCounter::setExcess (int *newExcess*) [virtual]

Sets the excess limit of the counter.

This method sets a new excess limit for the counter.

Parameters:

newExcess The new excess limit, an absolute number.

Returns:

The new excess limit.

Implements **Arc::Counter** (p. 71).

5.65.3.10 virtual int Arc::IntraProcessCounter::setLimit (int *newLimit*) [virtual]

Sets the limit of the counter.

This method sets a new limit for the counter.

Parameters:

newLimit The new limit, an absolute number.

Returns:

The new limit.

Implements **Arc::Counter** (p. 71).

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

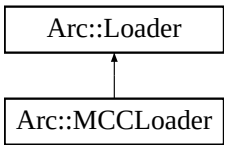
- IntraProcessCounter.h

5.66 Arc::Loader Class Reference

Plugins loader.

`#include <Loader.h>`

Inheritance diagram for Arc::Loader::



Public Member Functions

- **Loader** (**Config** &cfg)
- ~**Loader** ()

Protected Attributes

- **PluginsFactory** * **factory_**

5.66.1 Detailed Description

Plugins loader.

This class processes XML configration and loads specified plugins. Accepted configuration is defined by XML schema mcc.xsd. "Plugins" elements are parsed by this class and corresponding libraries are loaded.

5.66.2 Constructor & Destructor Documentation

5.66.2.1 Arc::Loader::Loader (Config & *cfg*)

Constructor that takes whole XML configuration and performs common configuration part

5.66.2.2 Arc::Loader::~Loader ()

Destructor destroys all components created by constructor

5.66.3 Field Documentation

5.66.3.1 PluginsFactory* Arc::Loader::factory_ [protected]

Link to Factory responsible for loading and creation of **Plugin** (p. 231) and derived objects

Referenced by Arc::ChainContext::operator PluginsFactory *().

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

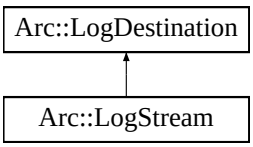
- Loader.h

5.67 Arc::LogDestination Class Reference

A base class for log destinations.

`#include <Logger.h>`

Inheritance diagram for Arc::LogDestination::



Public Member Functions

- virtual void **log** (const **LogMessage** &message)=0

Protected Member Functions

- **LogDestination** ()
- **LogDestination** (const std::string &locale)

5.67.1 Detailed Description

A base class for log destinations.

This class defines an interface for LogDestinations. **LogDestination** (p. 171) objects will typically contain synchronization mechanisms and should therefore never be copied.

5.67.2 Constructor & Destructor Documentation

5.67.2.1 Arc::LogDestination::LogDestination () [protected]

Default constructor.

This destination will use the default locale.

5.67.2.2 Arc::LogDestination::LogDestination (const std::string & *locale*) [protected]

Constructor with specific locale.

This destination will use the specified locale.

5.67.3 Member Function Documentation

5.67.3.1 virtual void Arc::LogDestination::log (const LogMessage & *message*) [pure virtual]

Logs a **LogMessage** (p. 176) to this **LogDestination** (p. 171).

Implemented in **Arc::LogStream** (p. 179).

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

- Logger.h

5.68 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)

Static Public Member Functions

- static **Logger** & **getRootLogger** ()

5.68.1 Detailed Description

A logger class.

This class defines a **Logger** (p. 173) to which LogMessages can be sent.

Every **Logger** (p. 173) (except for the rootLogger) has a parent **Logger** (p. 173). The domain of a **Logger** (p. 173) (a string that indicates the origin of LogMessages) is composed by adding a subdomain to the domain of its parent **Logger** (p. 173).

A **Logger** (p. 173) also has a threshold. Every **LogMessage** (p. 176) that have a level that is greater than or equal to the threshold is forwarded to any **LogDestination** (p. 171) connected to this **Logger** (p. 173) as well as to the parent **Logger** (p. 173).

Typical usage of the **Logger** (p. 173) class is to declare a global **Logger** (p. 173) object for each library/module/component to be used by all classes and methods there.

5.68.2 Constructor & Destructor Documentation

5.68.2.1 Arc::Logger::Logger (Logger & *parent*, const std::string & *subdomain*)

Creates a logger.

Creates a logger. The threshold is inherited from its parent **Logger** (p. 173).

Parameters:

parent The parent **Logger** (p. 173) of the new **Logger** (p. 173).

subdomain The subdomain of the new logger.

5.68.2.2 Arc::Logger::Logger (Logger & *parent*, const std::string & *subdomain*, LogLevel *threshold*)

Creates a logger.

Creates a logger.

Parameters:

parent The parent **Logger** (p. 173) of the new **Logger** (p. 173).

subdomain The subdomain of the new logger.

threshold The threshold of the new logger.

5.68.3 Member Function Documentation

5.68.3.1 void Arc::Logger::addDestination (LogDestination & *destination*)

Adds a **LogDestination** (p. 171).

Adds a **LogDestination** (p. 171) to which to forward LogMessages sent to this logger (if they pass the threshold). Since LogDestinatoins should not be copied, the new **LogDestination** (p. 171) is passed by reference and a pointer to it is kept for later use. It is therefore important that the **LogDestination** (p. 171) passed to this **Logger** (p. 173) exists at least as long as the **Logger** (p. 173) iteslf.

5.68.3.2 static Logger& Arc::Logger::getRootLogger () [static]

The root **Logger** (p. 173).

This is the root **Logger** (p. 173). It is an ancestor of any other **Logger** (p. 173) and allways exists.

5.68.3.3 LogLevel Arc::Logger::getThreshold () const

Returns the threshold.

Returns the threshold.

Returns:

The threshold of this **Logger** (p. 173).

5.68.3.4 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** (p. 176) and sends it to the other **msg()** (p. 175) method.

Parameters:

level The level of the message.

str The message text.

References msg().

5.68.3.5 void Arc::Logger::msg (LogMessage *message*)

Sends a **LogMessage** (p. 176).

Sends a **LogMessage** (p. 176).

Parameters:

The **LogMessage** (p. 176) to send.

Referenced by msg(), and Arc::stringto().

5.68.3.6 void Arc::Logger::removeDestinations (void)

Removes all LogDestinations.

5.68.3.7 void Arc::Logger::setThreshold (LogLevel *threshold*)

Sets the threshold.

This method sets the threshold of the **Logger** (p. 173). Any message sent to this **Logger** (p. 173) that has a level below this threshold will be discarded.

Parameters:

The threshold

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

- Logger.h

5.69 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)

5.69.1 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.

5.69.2 Constructor & Destructor Documentation

5.69.2.1 Arc::LogMessage::LogMessage (LogLevel *level*, const IString & *message*)

Creates a **LogMessage** (p. 176) with the specified level and message text.

This constructor creates a **LogMessage** (p. 176) with the specified level and message text. The time is set automatically, the domain is set by the **Logger** (p. 173) to which the **LogMessage** (p. 176) 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** (p. 176).

message The message text.

5.69.2.2 Arc::LogMessage::LogMessage (LogLevel *level*, const IString & *message*, const std::string & *identifier*)

Creates a **LogMessage** (p. 176) with the specified attributes.

This constructor creates a **LogMessage** (p. 176) with the specified level, message text and identifier. The time is set automatically and the domain is set by the **Logger** (p. 173) to which the **LogMessage** (p. 176) is sent.

Parameters:

level The level of the **LogMessage** (p. 176).

message The message text.

ident The identifier of the **LogMessage** (p. 176).

5.69.3 Member Function Documentation

5.69.3.1 LogLevel Arc::LogMessage::getLevel () const

Returns the level of the **LogMessage** (p. 176).

Returns the level of the **LogMessage** (p. 176).

Returns:

The level of the **LogMessage** (p. 176).

5.69.3.2 void Arc::LogMessage::setIdentifier (std::string *identifier*) [protected]

Sets the identifier of the **LogMessage** (p. 176).

The purpose of this method is to allow subclasses (in case there are any) to set the identifier of a **LogMessage** (p. 176).

Parameters:

The identifier.

5.69.4 Friends And Related Function Documentation

5.69.4.1 friend class Logger [friend]

The **Logger** (p. 173) class is a friend.

The **Logger** (p. 173) class must have some privileges (e.g. ability to call the setDomain() method), therefore it is a friend.

5.69.4.2 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:

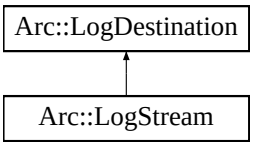
- Logger.h

5.70 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)

5.70.1 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** (p. 178) object as long as the **Logger** (p. 173) to which it has been registered.

5.70.2 Constructor & Destructor Documentation

5.70.2.1 Arc::LogStream::LogStream (std::ostream & *destination*)

Creates a **LogStream** (p. 178) connected to an ostream.

Creates a **LogStream** (p. 178) connected to the specified ostream. In order not to break synchronization, it is important not to connect more than one **LogStream** (p. 178) object to a certain stream.

Parameters:

destination The ostream to which to erite LogMessages.

5.70.2.2 Arc::LogStream::LogStream (std::ostream & *destination*, const std::string & *locale*)

Creates a **LogStream** (p. 178) connected to an ostream.

Creates a **LogStream** (p. 178) connected to the specified ostream. The output will be localised to the specified locale.

5.70.3 Member Function Documentation

5.70.3.1 virtual void Arc::LogStream::log (const LogMessage & *message*) [virtual]

Writes a **LogMessage** (p. 176) to the stream.

This method writes a **LogMessage** (p. 176) to the ostream that is connected to this **LogStream** (p. 178) object. It is synchronized so that not more than one **LogMessage** (p. 176) can be written at a time.

Parameters:

message The **LogMessage** (p. 176) to write.

Implements **Arc::LogDestination** (p. 171).

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

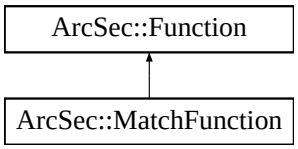
- Logger.h

5.71 ArcSec::MatchFunction Class Reference

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

`#include <MatchFunction.h>`

Inheritance diagram for ArcSec::MatchFunction::



Public Member Functions

- virtual bool **evaluate** (**AttributeValue** *arg0, **AttributeValue** *arg1)

Static Public Member Functions

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

5.71.1 Detailed Description

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

5.71.2 Member Function Documentation

5.71.2.1 virtual bool ArcSec::MatchFunction::evaluate (AttributeValue * *arg0*, AttributeValue * *arg1*) [virtual]

Evaluate two **AttributeValue** (p. 43) objects

Implements **ArcSec::Function** (p. 153).

5.71.2.2 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:

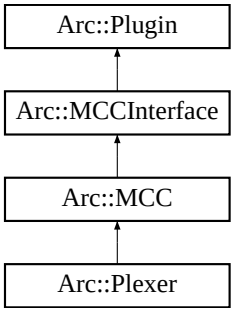
- MatchFunction.h

5.72 Arc::MCC Class Reference

Message (p. 192) Chain Component - base class for every **MCC** (p. 181) plugin.

`#include <MCC.h>`

Inheritance diagram for Arc::MCC::



Public Member Functions

- **MCC** (**Config** ∗)
- virtual void **Next** (**MCCInterface** ∗next, const std::string &label="")
- virtual void **AddSecHandler** (**Config** ∗cfg, **ArcSec::SecHandler** ∗sechandler, const std::string &label="")
- virtual void **Unlink** ()
- virtual **MCC_Status process** (**Message** &, **Message** &)

Protected Member Functions

- bool **ProcessSecHandlers** (**Message** &message, const std::string &label="")

Protected Attributes

- std::map< std::string, **MCCInterface** ∗ > **next_**
- std::map< std::string, std::list< **ArcSec::SecHandler** ∗ > > **sechandlers_**

Static Protected Attributes

- static **Logger logger**

5.72.1 Detailed Description

Message (p. 192) Chain Component - base class for every **MCC** (p. 181) plugin.

This is partialy virtual class which defines interface and common functionality for every **MCC** (p. 181) plugin needed for managing of component in a chain.

5.72.2 Constructor & Destructor Documentation

5.72.2.1 Arc::MCC::MCC (Config *) [inline]

Example contructor - **MCC** (p. 181) takes at least it's configuration subtree

5.72.3 Member Function Documentation

5.72.3.1 virtual void Arc::MCC::AddSecHandler (Config * *cfg*, ArcSec::SecHandler * *sechandler*, const std::string & *label* = "") [virtual]

Add security components/handlers to this **MCC** (p. 181). Security handlers are stacked into a few queues with each queue identified by its label. The queue labelled 'incoming' is executed for every 'request' message after the message is processed by the **MCC** (p. 181) on the service side and before processing on the client side. The queue labelled 'outgoing' is run for response message before it is processed by **MCC** (p. 181) algorithms on the service side and after processing on the client side. Those labels are just a matter of agreement and some MCCs may implement different queues executed at various message processing steps.

5.72.3.2 virtual void Arc::MCC::Next (MCCInterface * *next*, const std::string & *label* = "") [virtual]

Add reference to next **MCC** (p. 181) in chain. This method is called by **Loader** (p. 170) for every potentially labeled link to next component which implements **MCCInterface** (p. 187). If next is NULL corresponding link is removed.

Reimplemented in **Arc::Plexer** (p. 229).

5.72.3.3 virtual MCC_Status Arc::MCC::process (Message &, Message &) [inline, virtual]

Dummy **Message** (p. 192) processing method. Just a placeholder.

Implements **Arc::MCCInterface** (p. 187).

Reimplemented in **Arc::Plexer** (p. 229).

5.72.3.4 bool Arc::MCC::ProcessSecHandlers (Message & *message*, const std::string & *label* = "") [protected]

Executes security handlers of specified queue. Returns true if the message is authorized for further processing or if there are no security handlers which implement authorization functionality. This is a convenience method and has to be called by the implemention of the **MCC** (p. 181).

5.72.3.5 virtual void Arc::MCC::Unlink () [virtual]

Removing all links. Useful for destroying chains.

5.72.4 Field Documentation

5.72.4.1 Logger Arc::MCC::logger [static, protected]

A logger for MCCs.

A logger intended to be the parent of loggers in the different MCCs.

Reimplemented in **Arc::Plexer** (p. 229).

5.72.4.2 std::map<std::string, MCCInterface *> Arc::MCC::next_ [protected]

Set of labeled "next" components. Each implemented **MCC** (p. 181) must call **process()** (p. 182) method of corresponding **MCCInterface** (p. 187) from this set in own **process()** (p. 182) method.

5.72.4.3 std::map<std::string, std::list<ArcSec::SecHandler *> > Arc::MCC::sechandlers_ [protected]

Set of labeled authentication and authorization handlers. **MCC** (p. 181) 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:

- MCC.h

5.73 Arc::MCC_Status Class Reference

A class for communication of **MCC** (p. 181) 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

5.73.1 Detailed Description

A class for communication of **MCC** (p. 181) processing results.

This class is used to communicate result status between MCCs. It contains a status kind, a string specifying the origin (**MCC** (p. 181)) of the status object and an explanation.

5.73.2 Constructor & Destructor Documentation

5.73.2.1 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** (p. 184) object.

Parameters:

kind The StatusKind (default: STATUS_UNDEFINED)

origin The origin **MCC** (p. 181) (default: "???")

explanation An explanation (default: "No explanation.")

5.73.3 Member Function Documentation

5.73.3.1 const std::string& Arc::MCC_Status::getExplanation () const

Returns an explanation.

This method returns an explanation of this object.

Returns:

An explanation of this object.

5.73.3.2 StatusKind Arc::MCC_Status::getKind () const

Returns the status kind.

Returns the status kind of this object.

Returns:

The status kind of this object.

5.73.3.3 const std::string& Arc::MCC_Status::getOrigin () const

Returns the origin.

This method returns a string specifying the origin **MCC** (p. 181) of this object.

Returns:

A string specifying the origin **MCC** (p. 181) of this object.

5.73.3.4 bool Arc::MCC_Status::isOk () const

Is the status kind ok?

This method returns true if the status kind of this object is STATUS_OK

Returns:

true if kind==STATUS_OK

Referenced by operator bool(), and operator!().

5.73.3.5 Arc::MCC_Status::operator bool (void) const [inline]

Is the status kind ok?

This method returns true if the status kind of this object is STATUS_OK

Returns:

true if kind==STATUS_OK

References isOk().

5.73.3.6 Arc::MCC_Status::operator std::string () const

Conversion to string.

This operator converts a **MCC_Status** (p. 184) object to a string.

5.73.3.7 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:

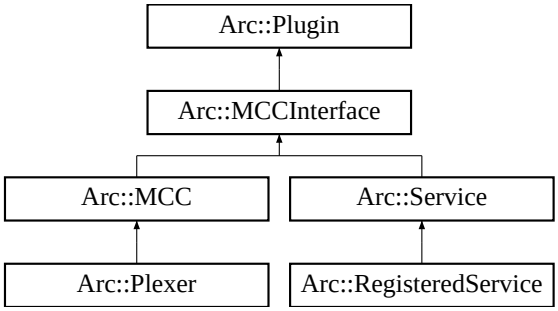
- MCC_Status.h

5.74 Arc::MCCInterface Class Reference

Interface for communication between **MCC** (p. 181), **Service** (p. 270) and **Plexer** (p. 228) objects.

`#include <MCC.h>`

Inheritance diagram for Arc::MCCInterface::



Public Member Functions

- virtual **MCC_Status process** (**Message** &request, **Message** &response)=0

5.74.1 Detailed Description

Interface for communication between **MCC** (p. 181), **Service** (p. 270) and **Plexer** (p. 228) objects.

The Interface consists of the method **process()** (p. 187) which is called by the previous **MCC** (p. 181) in the chain. For memory management policies please read the description of the **Message** (p. 192) class.

5.74.2 Member Function Documentation

5.74.2.1 virtual MCC_Status Arc::MCCInterface::process (Message & *request*, Message & *response*) [pure virtual]

Method for processing of requests and responses. This method is called by preceeding **MCC** (p. 181) in chain when a request needs to be processed. This method must call similar method of next **MCC** (p. 181) in chain unless any failure happens. Result returned by call to next **MCC** (p. 181) should be processed and passed back to previous **MCC** (p. 181). In case of failure this method is expected to generate valid error response and return it back to previous **MCC** (p. 181) without calling the next one.

Parameters:

request The request that needs to be processed.

response A **Message** (p. 192) object that will contain the response of the request when the method returns.

Returns:

An object representing the status of the call.

Implemented in **Arc::MCC** (p. 182), and **Arc::Plexer** (p. 229).

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

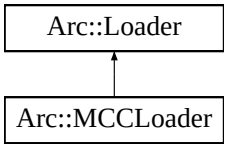
- MCC.h

5.75 Arc::MCCLoader Class Reference

Creator of **Message** (p. 192) Component Chains (**MCC** (p. 181)).

`#include <MCCLoader.h>`

Inheritance diagram for Arc::MCCLoader::



Public Member Functions

- **MCCLoader** (**Config** &cfg)
- ~**MCCLoader** ()
- **MCC** * **operator[]** (const std::string &id)

5.75.1 Detailed Description

Creator of **Message** (p. 192) Component Chains (**MCC** (p. 181)).

This class processes XML configration and creates message chains. Accepted configuration is defined by XML schema mcc.xsd. Supported components are of types **MCC** (p. 181), **Service** (p. 270) and **Plexer** (p. 228). **MCC** (p. 181) and **Service** (p. 270) are loaded from dynamic libraries. For **Plexer** (p. 228) 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** (p. 192) 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.

5.75.2 Constructor & Destructor Documentation

5.75.2.1 Arc::MCCLoader::MCCLoader (Config & *cfg*)

Constructor that takes whole XML configuration and creates component chains

5.75.2.2 Arc::MCCLoader::~MCCLoader ()

Destructor destroys all components created by constructor

5.75.3 Member Function Documentation

5.75.3.1 MCC* Arc::MCCLoader::operator[ ] (const std::string & *id*)

Access entry MCCs in chains. Those are components exposed for external access using 'entry' attribute
The documentation for this class was generated from the following file:

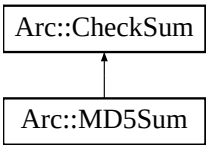
- MCCLoader.h

5.76 Arc::MD5Sum Class Reference

Implementation of MD5 checksum.

`#include <CheckSum.h>`

Inheritance diagram for Arc::MD5Sum::



5.76.1 Detailed Description

Implementation of MD5 checksum.

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

- CheckSum.h

5.77 Arc::Message Class Reference

Object being passed through chain of MCCs.

`#include <Message.h>`

Public Member Functions

- **Message** (void)
- **Message** (**Message** &msg)
- **Message** (long msg_ptr_addr)
- ~**Message** (void)
- **Message** & **operator=** (**Message** &msg)
- **MessagePayload** * **Payload** (void)
- **MessagePayload** * **Payload** (**MessagePayload** *payload)
- **MessageAttributes** * **Attributes** (void)
- **MessageAuth** * **Auth** (void)
- **MessageContext** * **Context** (void)
- **MessageAuthContext** * **AuthContext** (void)
- void **Context** (**MessageContext** *ctx)
- void **AuthContext** (**MessageAuthContext** *auth_ctx)

5.77.1 Detailed Description

Object being passed through chain of MCCs.

An instance of this class refers to objects with main content (**MessagePayload** (p. 203)), authentication/authorization information (**MessageAuth** (p. 198)) and common purpose attributes (**MessageAttributes** (p. 195)). **Message** (p. 192) class does not manage pointers to objects and their content. It only serves for grouping those objects. **Message** (p. 192) objects are supposed to be processed by MCCs and Services implementing **MCCInterface** (p. 187) method process(). All objects constituting content of **Message** (p. 192) 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** (p. 192). b) Objects whose management is completely acquired by objects assigned to 'response' **Message** (p. 192).

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** (p. 192) object).

4. It is allowed to change content of pointers of 'request' **Message** (p. 192). Calling process() method must not rely on that object to stay intact.

5. Called process() method should either fill 'response' **Message** (p. 192) with pointers to valid objects or to keep them intact. This makes it possible for calling process() to preload 'response' with valid error message.

5.77.2 Constructor & Destructor Documentation

5.77.2.1 Arc::Message::Message (void) [inline]

true if auth_ctx_ was created internally Dummy constructor

5.77.2.2 Arc::Message::Message (Message & *msg*) [inline]

Copy constructor. Ensures shallow copy.

5.77.2.3 Arc::Message::Message (long *msg_ptr_addr*)

Copy constructor. Used by language bindigs

5.77.2.4 Arc::Message::~Message (void) [inline]

Destructor does not affect refered objects except those created internally

5.77.3 Member Function Documentation

5.77.3.1 MessageAttributes* Arc::Message::Attributes (void) [inline]

Returns a pointer to the current attributes object or creates it if no attributes object has been assigned.

Referenced by operator=().

5.77.3.2 MessageAuth* Arc::Message::Auth (void) [inline]

Returns a pointer to the current authentication/authorization object or creates it if no object has been assigned.

Referenced by operator=().

5.77.3.3 void Arc::Message::AuthContext (MessageAuthContext * *auth_ctx*) [inline]

Assigns auth* context object

5.77.3.4 MessageAuthContext* Arc::Message::AuthContext (void) [inline]

Returns a pointer to the current auth* context object or creates it if no object has been assigned.

Referenced by operator=().

5.77.3.5 void Arc::Message::Context (MessageContext * *ctx*) [inline]

Assigns message context object

5.77.3.6 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** (p. 181) in a chain is connectionless like one implementing UDP protocol.

Referenced by operator=().

5.77.3.7 Message& Arc::Message::operator= (Message & *msg*) [inline]

Assignment. Ensures shallow copy.

References attr_, Attributes(), Auth(), auth_, auth_ctx_, AuthContext(), Context(), ctx_, and payload_.

5.77.3.8 MessagePayload* Arc::Message::Payload (MessagePayload * *payload*) [inline]

Replaces payload with new one. Returns the old one.

5.77.3.9 MessagePayload* Arc::Message::Payload (void) [inline]

Returns pointer to current payload or NULL if no payload assigned.

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

- Message.h

5.78 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_**

5.78.1 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** (p. 192) Chain Component (**MCC** (p. 181)) 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** (p. 181) 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** (p. 181). 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** (p. 181) and used by the plexer for routing the message to the appropriate service.

5.78.2 Constructor & Destructor Documentation

5.78.2.1 Arc::MessageAttributes::MessageAttributes ()

The default constructor.

This is the default constructor of the **MessageAttributes** (p. 195) class. It constructs an empty object that initially contains no attributes.

5.78.3 Member Function Documentation

5.78.3.1 void Arc::MessageAttributes::add (const std::string & *key*, const std::string & *value*)

Adds a value to an attribute.

This method adds a new value to an attribute. Any previous value will be preserved, i.e. the attribute may become multiple valued.

Parameters:

key The key of the attribute.

value The (new) value of the attribute.

5.78.3.2 int Arc::MessageAttributes::count (const std::string & *key*) const

Returns the number of values of an attribute.

Returns the number of values of an attribute that matches a certain key.

Parameters:

key The key of the attribute for which to count values.

Returns:

The number of values that corresponds to the key.

5.78.3.3 const std::string& Arc::MessageAttributes::get (const std::string & *key*) const

Returns the value of a single-valued attribute.

This method returns the value of a single-valued attribute. If the attribute is not single valued (i.e. there is no such attribute or it is a multiple-valued attribute) an empty string is returned.

Parameters:

key The key of the attribute for which to return the value.

Returns:

The value of the attribute.

5.78.3.4 AttributeIterator Arc::MessageAttributes::getAll (void) const

Access all value and attributes.

5.78.3.5 AttributeIterator Arc::MessageAttributes::getAll (const std::string & *key*) const

Access the value(s) of an attribute.

This method returns an **AttributeIterator** (p. 39) 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** (p. 39) for access of the values of the attribute.

5.78.3.6 void Arc::MessageAttributes::remove (const std::string & *key*, const std::string & *value*)

Removes one value of an attribute.

This method removes a certain value from the attribute that matches a certain key.

Parameters:

key The key of the attribute from which the value shall be removed.

value The value to remove.

5.78.3.7 void Arc::MessageAttributes::removeAll (const std::string & *key*)

Removes all attributes with a certain key.

This method removes all attributes that match a certain key.

Parameters:

key The key of the attributes to remove.

5.78.3.8 void Arc::MessageAttributes::set (const std::string & *key*, const std::string & *value*)

Sets a unique value of an attribute.

This method removes any previous value of an attribute and sets the new value as the only value.

Parameters:

key The key of the attribute.

value The (new) value of the attribute.

5.78.4 Field Documentation

5.78.4.1 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:

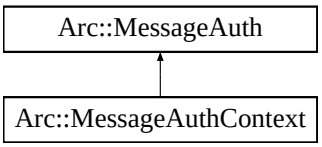
- MessageAttributes.h

5.79 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** (**SecAttrFormat** format, **XMLNode** &val) const
- **MessageAuth** ∗ **Filter** (const std::list< std::string > selected_keys, const std::list< std::string > rejected_keys) const

5.79.1 Detailed Description

Contains authencity information, authorization tokens and decisions.

This class only supports string keys and **SecAttr** (p. 261) values.

5.79.2 Member Function Documentation

5.79.2.1 bool Arc::MessageAuth::Export (SecAttrFormat *format*, XMLNode & *val*) const

Returns properly catenated attributes in specified format.

Content of XML node at is replaced with generated information if XML tree is empty. If tree at is not empty then **Export()** (p. 198) tries to merge generated information to already existing like everything would be generated inside same **Export()** (p. 198) method. If does not represent valid node then new XML tree is created.

5.79.2.2 MessageAuth∗ Arc::MessageAuth::Filter (const std::list< std::string > *selected_keys*, const std::list< std::string > *rejected_keys*) const

Creates new instance of **MessageAuth** (p. 198) 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.

5.79.2.3 SecAttr* Arc::MessageAuth::get (const std::string & *key*)

Retrieves reference to security attribute stored under specified key.

5.79.2.4 SecAttr* Arc::MessageAuth::operator[ ] (const std::string & *key*) [inline]

Same as **MessageAuth::get** (p. 199).

5.79.2.5 void Arc::MessageAuth::remove (const std::string & *key*)

Deletes security attribute stored under specified key.

5.79.2.6 void Arc::MessageAuth::set (const std::string & *key*, SecAttr * *value*)

Adds/overwrites security attribute stored under specified key.

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

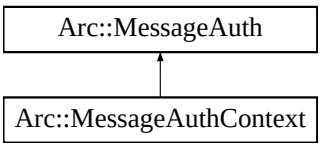
- MessageAuth.h

5.80 Arc::MessageAuthContext Class Reference

Handler for content of message auth∗ context.

`#include <Message.h>`

Inheritance diagram for Arc::MessageAuthContext::



5.80.1 Detailed Description

Handler for content of message auth∗ context.

This class is a container for authorization and authentication information. It gets associated with **Message** (p. 192) object usually by first **MCC** (p. 181) in a chain and is kept as long as connection persists.

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

- Message.h

5.81 Arc::MessageContext Class Reference

Handler for content of message context.

`#include <Message.h>`

Public Member Functions

- void **Add** (const std::string &name, **MessageContextElement** *element)

5.81.1 Detailed Description

Handler for content of message context.

This class is a container for objects derived from **MessageContextElement** (p. 202). It gets associated with **Message** (p. 192) object usually by first **MCC** (p. 181) in a chain and is kept as long as connection persists.

5.81.2 Member Function Documentation

5.81.2.1 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

5.82 Arc::MessageContextElement Class Reference

Top class for elements contained in message context.

`#include <Message.h>`

Inherited by ArcSec::PDPConfigContext.

5.82.1 Detailed Description

Top class for elements contained in message context.

Objects of classes inherited with this one may be stored in **MessageContext** (p. 201) container.

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

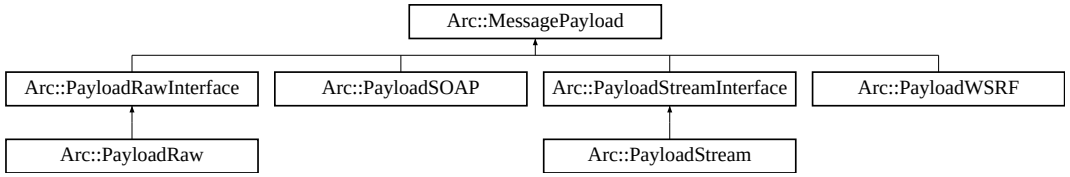
- Message.h

5.83 Arc::MessagePayload Class Reference

Base class for content of message passed through chain.

`#include <Message.h>`

Inheritance diagram for Arc::MessagePayload::



5.83.1 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:

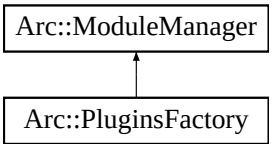
- Message.h

5.84 Arc::ModuleManager Class Reference

Manager of shared libraries.

`#include <ModuleManager.h>`

Inheritance diagram for Arc::ModuleManager::



Public Member Functions

- **ModuleManager** (const **Config** ∗cfg)
- Glib::Module ∗ **load** (const std::string &name, bool probe=false)
- Glib::Module ∗ **reload** (Glib::Module ∗module)
- std::string **findLocation** (const std::string &name)
- void **setCfg** (**Config** ∗cfg)

5.84.1 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.

5.84.2 Constructor & Destructor Documentation

5.84.2.1 Arc::ModuleManager::ModuleManager (const 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.

5.84.3 Member Function Documentation

5.84.3.1 std::string Arc::ModuleManager::findLocation (const std::string & *name*)

Finds shared library corresponding to module 'name' and returns path to it

5.84.3.2 Glib::Module∗ Arc::ModuleManager::load (const std::string & *name*, bool *probe* = `false`)

Finds module 'name' in cache or loads corresponding shared library

5.84.3.3 Glib::Module* Arc::ModuleManager::reload (Glib::Module * *module*)

Reload module previously loaded in probe mode. New module is loaded with all symbols resolved and old module handler is unloaded. In case of error old module is not unloaded.

5.84.3.4 void Arc::ModuleManager::setCfg (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:

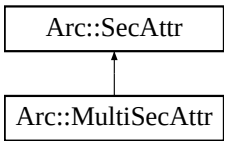
- ModuleManager.h

5.85 Arc::MultiSecAttr Class Reference

Container of multiple **SecAttr** (p. 261) attributes.

`#include <SecAttr.h>`

Inheritance diagram for Arc::MultiSecAttr::



Public Member Functions

- virtual **operator bool** () const
- virtual bool **Export** (**SecAttrFormat** format, **XMLNode** &val) const

5.85.1 Detailed Description

Container of multiple **SecAttr** (p. 261) 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.

5.85.2 Member Function Documentation

5.85.2.1 virtual bool Arc::MultiSecAttr::Export (SecAttrFormat *format*, XMLNode & *val*) const [virtual]

Convert internal structure into specified format. Returns false if format is not supported/suitable for this attribute. XML node referenced by is turned into top level element of specified format.

Reimplemented from **Arc::SecAttr** (p. 262).

5.85.2.2 virtual Arc::MultiSecAttr::operator bool () const [virtual]

This function should return false if the value is to be considered null, e.g. if it hasn't been set or initialized. In other cases it should return true.

Reimplemented from **Arc::SecAttr** (p. 262).

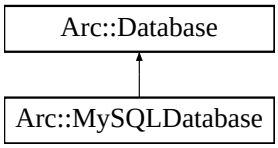
The documentation for this class was generated from the following file:

- SecAttr.h

5.86 Arc::MySQLDatabase Class Reference

`#include <MysqlWrapper.h>`

Inheritance diagram for Arc::MySQLDatabase::



Public Member Functions

- virtual bool **connect** (std::string &dbname, std::string &user, std::string &password)
- virtual bool **isconnected** () const
- virtual void **close** ()
- virtual bool **enable_ssl** (const std::string keyfile="", const std::string certfile="", const std::string cafile="", const std::string capath="")
- virtual bool **shutdown** ()

5.86.1 Detailed Description

Implement the database accessing interface in **DBInterface.h** (p. **??**) by using mysql client library for accessing mysql database

5.86.2 Member Function Documentation

5.86.2.1 virtual void Arc::MySQLDatabase::close () [virtual]

Close the connection with database server

Implements **Arc::Database** (p. 77).

5.86.2.2 virtual bool Arc::MySQLDatabase::connect (std::string & *dbname*, std::string & *user*, std::string & *password*) [virtual]

Do connection with database server

Parameters:

dbname The database name which will be used.

user The username which will be used to access database.

password The password which will be used to access database.

Implements **Arc::Database** (p. 77).

5.86.2.3 virtual bool Arc::MySQLDatabase::enable_ssl (const std::string *keyfile* = "", const std::string *certfile* = "", const std::string *cafile* = "", const std::string *capath* = "") [virtual]

Enable ssl communication for the connection

Parameters:

keyfile The location of key file.

certfile The location of certificate file.

cafile The location of ca file.

capath The location of ca directory

Implements **Arc::Database** (p. 77).

5.86.2.4 virtual bool Arc::MySQLDatabase::isconnected () const [inline, virtual]

Get the connection status

Implements **Arc::Database** (p. 77).

5.86.2.5 virtual bool Arc::MySQLDatabase::shutdown () [virtual]

Ask database server to shutdown

Implements **Arc::Database** (p. 77).

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

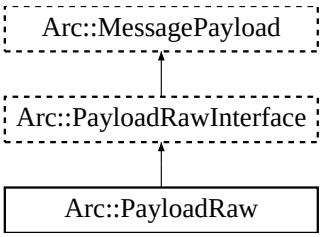
- MysqlWrapper.h

5.87 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=-1)
- 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)

5.87.1 Detailed Description

Raw byte multi-buffer.

This is implementation of **PayloadRawInterface** (p. 212). Buffers are memory blocks logically placed one after another.

5.87.2 Constructor & Destructor Documentation

5.87.2.1 Arc::PayloadRaw::PayloadRaw (void) [inline]

List of handled buffers. Constructor. Created object contains no buffers.

5.87.2.2 virtual Arc::PayloadRaw::~PayloadRaw (void) [virtual]

Destructor. Frees allocated buffers.

5.87.3 Member Function Documentation

5.87.3.1 virtual char* Arc::PayloadRaw::Buffer (unsigned int *num* = 0) [virtual]

Returns pointer to num'th buffer

Implements **Arc::PayloadRawInterface** (p. 212).

5.87.3.2 virtual int Arc::PayloadRaw::BufferPos (unsigned int *num* = 0) const [virtual]

Returns position of num'th buffer

Implements **Arc::PayloadRawInterface** (p. 212).

5.87.3.3 virtual int Arc::PayloadRaw::BufferSize (unsigned int *num* = 0) const [virtual]

Returns length of num'th buffer

Implements **Arc::PayloadRawInterface** (p. 213).

5.87.3.4 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** (p. 213).

5.87.3.5 virtual char* Arc::PayloadRaw::Insert (const char * *s*, int *pos* = 0, int *size* = -1) [virtual]

Create new buffer at global position 'pos' of size 'size'. Created buffer is filled with content of memory at 's'. If 'size' is negative content at 's' is expected to be null-terminated.

Implements **Arc::PayloadRawInterface** (p. 213).

5.87.3.6 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** (p. 213).

5.87.3.7 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** (p. 213).

5.87.3.8 virtual int Arc::PayloadRaw::Size (void) const [virtual]

Returns logical size of whole structure.

Implements **Arc::PayloadRawInterface** (p. 213).

5.87.3.9 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** (p. 213).

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

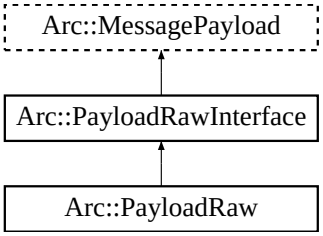
- PayloadRaw.h

5.88 Arc::PayloadRawInterface Class Reference

Random Access Payload for **Message** (p. 192) 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=-1)=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

5.88.1 Detailed Description

Random Access Payload for **Message** (p. 192) objects.

This class is a virtual interface for managing **Message** (p. 192) 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.

5.88.2 Member Function Documentation

5.88.2.1 virtual char* Arc::PayloadRawInterface::Buffer (unsigned int *num*) [pure virtual]

Returns pointer to num'th buffer

Implemented in **Arc::PayloadRaw** (p. 210).

5.88.2.2 virtual int Arc::PayloadRawInterface::BufferPos (unsigned int *num*) const [pure virtual]

Returns position of num'th buffer

Implemented in **Arc::PayloadRaw** (p. 210).

5.88.2.3 virtual int Arc::PayloadRawInterface::BufferSize (unsigned int *num*) const [pure virtual]

Returns length of num'th buffer

Implemented in **Arc::PayloadRaw** (p. 210).

5.88.2.4 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** (p. 210).

5.88.2.5 virtual char* Arc::PayloadRawInterface::Insert (const char * *s*, int *pos* = 0, int *size* = -1) [pure virtual]

Create new buffer at global position 'pos' of size 'size'. Created buffer is filled with content of memory at 's'. If 'size' is negative content at 's' is expected to be null-terminated.

Implemented in **Arc::PayloadRaw** (p. 210).

5.88.2.6 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** (p. 210).

5.88.2.7 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** (p. 210).

5.88.2.8 virtual int Arc::PayloadRawInterface::Size (void) const [pure virtual]

Returns logical size of whole structure.

Implemented in **Arc::PayloadRaw** (p. 210).

5.88.2.9 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** (p. 211).

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

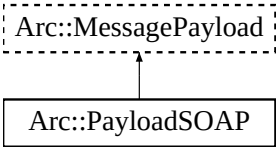
- PayloadRaw.h

5.89 Arc::PayloadSOAP Class Reference

Payload of **Message** (p. 192) with SOAP content.

`#include <PayloadSOAP.h>`

Inheritance diagram for Arc::PayloadSOAP::



Public Member Functions

- **PayloadSOAP** (const NS &ns, bool fault=false)
- **PayloadSOAP** (const SOAPEnvelope &soap)
- **PayloadSOAP** (const **MessagePayload** &source)

5.89.1 Detailed Description

Payload of **Message** (p. 192) with SOAP content.

This class combines **MessagePayload** (p. 203) with SOAPEnvelope to make it possible to pass SOAP messages through **MCC** (p. 181) chain.

5.89.2 Constructor & Destructor Documentation

5.89.2.1 Arc::PayloadSOAP::PayloadSOAP (const NS & *ns*, bool *fault* = `false`)

Constructor - creates new **Message** (p. 192) payload

5.89.2.2 Arc::PayloadSOAP::PayloadSOAP (const SOAPEnvelope & *soap*)

Constructor - creates **Message** (p. 192) payload from SOAP document. Provided SOAP document is copied to new object.

5.89.2.3 Arc::PayloadSOAP::PayloadSOAP (const MessagePayload & *source*)

Constructor - creates SOAP message from payload. **PayloadRawInterface** (p. 212) and derived classes are supported.

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

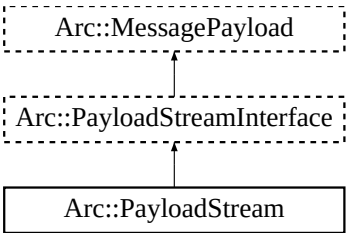
- PayloadSOAP.h

5.90 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)
- virtual int **Pos** (void) const

Protected Attributes

- int **handle_**
- bool **seekable_**

5.90.1 Detailed Description

POSIX handle as Payload.

Thsi is an implemetation of **PayloadStreamInterface** (p. 219) for generic POSIX handle.

5.90.2 Constructor & Destructor Documentation

5.90.2.1 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.

5.90.2.2 virtual Arc::PayloadStream::~PayloadStream (void) [inline, virtual]

Destructor.

5.90.3 Member Function Documentation

5.90.3.1 virtual std::string Arc::PayloadStream::Get (void) [inline, virtual]

Read as many as possible (sane amount) of bytes.

Implements **Arc::PayloadStreamInterface** (p. 219).

References Get().

Referenced by Get().

5.90.3.2 virtual bool Arc::PayloadStream::Get (std::string & *buf*) [virtual]

Read as many as possible (sane amount) of bytes into buf.

Implements **Arc::PayloadStreamInterface** (p. 219).

5.90.3.3 virtual bool Arc::PayloadStream::Get (char * *buf*, int & *size*) [virtual]

Extracts information from stream up to 'size' bytes. 'size' contains number of read bytes on exit. Returns true in case of success.

Implements **Arc::PayloadStreamInterface** (p. 220).

5.90.3.4 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** (p. 181).

References handle_.

5.90.3.5 virtual Arc::PayloadStream::operator bool (void) [inline, virtual]

Returns true if stream is valid.

Implements **Arc::PayloadStreamInterface** (p. 220).

References handle_.

5.90.3.6 virtual bool Arc::PayloadStream::operator! (void) [inline, virtual]

Returns true if stream is invalid.

Implements **Arc::PayloadStreamInterface** (p. 220).

References handle_.

5.90.3.7 virtual int Arc::PayloadStream::Pos (void) const [inline, virtual]

Returns current position in stream if supported.

Implements **Arc::PayloadStreamInterface** (p. 220).

5.90.3.8 virtual bool Arc::PayloadStream::Put (const char * *buf*) [inline, virtual]

Push null terminated information from 'buf' into stream. Returns true on success.

Implements **Arc::PayloadStreamInterface** (p. 220).

References Put().

Referenced by Put().

5.90.3.9 virtual bool Arc::PayloadStream::Put (const std::string & *buf*) [inline, virtual]

Push information from 'buf' into stream. Returns true on success.

Implements **Arc::PayloadStreamInterface** (p. 220).

References Put().

Referenced by Put().

5.90.3.10 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** (p. 220).

5.90.3.11 virtual void Arc::PayloadStream::Timeout (int *to*) [inline, virtual]

Set current timeout for **Get()** (p. 216) and **Put()** (p. 217) operations.

Implements **Arc::PayloadStreamInterface** (p. 220).

5.90.3.12 virtual int Arc::PayloadStream::Timeout (void) const [inline, virtual]

Query current timeout for **Get()** (p. 216) and **Put()** (p. 217) operations.

Implements **Arc::PayloadStreamInterface** (p. 221).

5.90.4 Field Documentation

5.90.4.1 int Arc::PayloadStream::handle_ [protected]

Timeout for read/write operations

Referenced by GetHandle(), operator bool(), and operator!().

5.90.4.2 bool Arc::PayloadStream::seekable_ [protected]

Handle for operations

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

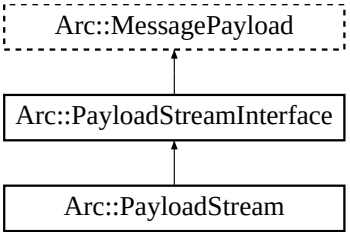
- PayloadStream.h

5.91 Arc::PayloadStreamInterface Class Reference

Stream-like Payload for **Message** (p. 192) 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
- virtual int **Pos** (void) const =0

5.91.1 Detailed Description

Stream-like Payload for **Message** (p. 192) object.

This class is a virtual interface for managing stream-like source and destination. It's supposed to be passed through **MCC** (p. 181) chain as payload of **Message** (p. 192). It must be treated by MCCs and Services as dynamic payload. This class is purely virtual.

5.91.2 Member Function Documentation

5.91.2.1 virtual std::string Arc::PayloadStreamInterface::Get (void) [pure virtual]

Read as many as possible (sane amount) of bytes.

Implemented in **Arc::PayloadStream** (p. 216).

5.91.2.2 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** (p. 216).

5.91.2.3 virtual bool Arc::PayloadStreamInterface::Get (char * *buf*, int & *size*) [pure virtual]

Extracts information from stream up to 'size' bytes. 'size' contains number of read bytes on exit. Returns true in case of success.

Implemented in **Arc::PayloadStream** (p. 216).

5.91.2.4 virtual Arc::PayloadStreamInterface::operator bool (void) [pure virtual]

Returns true if stream is valid.

Implemented in **Arc::PayloadStream** (p. 216).

5.91.2.5 virtual bool Arc::PayloadStreamInterface::operator! (void) [pure virtual]

Returns true if stream is invalid.

Implemented in **Arc::PayloadStream** (p. 216).

5.91.2.6 virtual int Arc::PayloadStreamInterface::Pos (void) const [pure virtual]

Returns current position in stream if supported.

Implemented in **Arc::PayloadStream** (p. 217).

5.91.2.7 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** (p. 217).

5.91.2.8 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** (p. 217).

5.91.2.9 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** (p. 217).

5.91.2.10 virtual void Arc::PayloadStreamInterface::Timeout (int *to*) [pure virtual]

Set current timeout for **Get()** (p. 219) and **Put()** (p. 220) operations.

Implemented in **Arc::PayloadStream** (p. 217).

5.91.2.11 virtual int Arc::PayloadStreamInterface::Timeout (void) const [pure virtual]

Query current timeout for **Get()** (p. 219) and **Put()** (p. 220) operations.

Implemented in **Arc::PayloadStream** (p. 217).

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

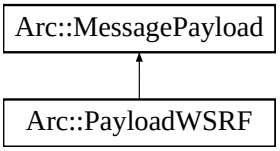
- PayloadStream.h

5.92 Arc::PayloadWSRF Class Reference

This class combines **MessagePayload** (p. 203) with **WSRF** (p. 306).

`#include <PayloadWSRF.h>`

Inheritance diagram for Arc::PayloadWSRF::



Public Member Functions

- **PayloadWSRF** (const SOAPEnvelope &soap)
- **PayloadWSRF** (**WSRF** &wsrp)
- **PayloadWSRF** (const **MessagePayload** &source)

5.92.1 Detailed Description

This class combines **MessagePayload** (p. 203) with **WSRF** (p. 306).

It's intention is to make it possible to pass **WSRF** (p. 306) messages through **MCC** (p. 181) chain as one more Payload type.

5.92.2 Constructor & Destructor Documentation

5.92.2.1 Arc::PayloadWSRF::PayloadWSRF (const SOAPEnvelope & *soap*)

Constructor - creates **Message** (p. 192) payload from SOAP message. Returns invalid **WSRF** (p. 306) if SOAP does not represent WS-ResourceProperties

5.92.2.2 Arc::PayloadWSRF::PayloadWSRF (WSRF & *wsrp*)

Constructor - creates **Message** (p. 192) payload with acquired **WSRF** (p. 306) message. **WSRF** (p. 306) message will be destroyed by destructor of this object.

5.92.2.3 Arc::PayloadWSRF::PayloadWSRF (const MessagePayload & *source*)

Constructor - creates **WSRF** (p. 306) message from payload. All classes derived from SOAPEnvelope are supported.

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

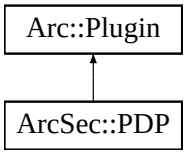
- PayloadWSRF.h

5.93 ArcSec::PDP Class Reference

Base class for **Policy** (p. 236) Decision Point plugins.

`#include <PDP.h>`

Inheritance diagram for ArcSec::PDP::



5.93.1 Detailed Description

Base class for **Policy** (p. 236) Decision Point plugins.

This virtual class defines method isPermitted() which processes security related information/attributes in Message and makes security decision - permit (true) or deny (false). Configuration of **PDP** (p. 223) is consumed during creation of instance through XML subtree fed to constructor.

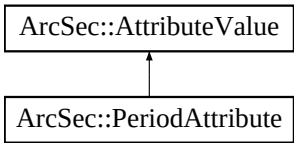
The documentation for this class was generated from the following file:

- PDP.h

5.94 ArcSec::PeriodAttribute Class Reference

`#include <DateTimeAttribute.h>`

Inheritance diagram for ArcSec::PeriodAttribute::



Public Member Functions

- virtual bool **equal** (**AttributeValue** *other)
- virtual std::string **encode** ()
- virtual std::string **getType** ()
- virtual std::string **getId** ()

5.94.1 Detailed Description

Formate: datetime"/"duration datetime"/"datetime duration"/"datetime

5.94.2 Member Function Documentation

5.94.2.1 virtual std::string ArcSec::PeriodAttribute::encode () [virtual]

encode the value in a string format

Implements **ArcSec::AttributeValue** (p. 43).

5.94.2.2 virtual bool ArcSec::PeriodAttribute::equal (AttributeValue * *value*) [virtual]

Evluate whether "this" equale to the parameter value

Implements **ArcSec::AttributeValue** (p. 43).

5.94.2.3 virtual std::string ArcSec::PeriodAttribute::getId () [inline, virtual]

Get the identifier of the <Attribute>

Implements **ArcSec::AttributeValue** (p. 43).

5.94.2.4 virtual std::string ArcSec::PeriodAttribute::getType () [inline, virtual]

Get the type of the <Attribute>

Implements **ArcSec::AttributeValue** (p. 44).

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

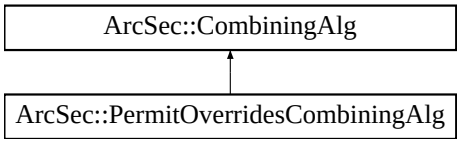
- DateTimeAttribute.h

5.95 ArcSec::PermitOverridesCombiningAlg Class Reference

Implement the "Permit-Overrides" algorithm.

`#include <PermitOverridesAlg.h>`

Inheritance diagram for ArcSec::PermitOverridesCombiningAlg::



Public Member Functions

- virtual Result **combine** (**EvaluationCtx** *ctx, std::list< **Policy** * > policies)
- virtual const std::string & **getalgId** (void) const

5.95.1 Detailed Description

Implement the "Permit-Overrides" algorithm.

Permit-Overrides, scans the policy set which is given as the parameters of "combine" method, if gets "permit" result from any policy, then stops scanning and gives "permit" as result, otherwise gives "deny".

5.95.2 Member Function Documentation

5.95.2.1 virtual Result ArcSec::PermitOverridesCombiningAlg::combine (EvaluationCtx * *ctx*, std::list< Policy * > *policies*) [virtual]

If there is one policy which return positive evaluation result, then omit the other policies and return DECISION_PERMIT

Parameters:

ctx This object contains request information which will be used to evaluated against policy.

policlies This is a container which contains policy objects.

Returns:

The combined result according to the algorithm.

Implements **ArcSec::CombiningAlg** (p. 59).

5.95.2.2 virtual const std::string& ArcSec::PermitOverridesCombiningAlg::getalgId (void) const [inline, virtual]

Get the identifier

Implements **ArcSec::CombiningAlg** (p. 59).

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

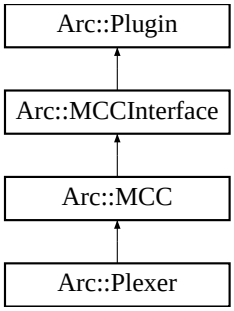
- PermitOverridesAlg.h

5.96 Arc::Plexer Class Reference

The **Plexer** (p. 228) 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 **Logger logger**

5.96.1 Detailed Description

The **Plexer** (p. 228) class, used for routing messages to services.

This is the **Plexer** (p. 228) class. Its purpose is to route incoming messages to appropriate Services and **MCC** (p. 181) chains.

5.96.2 Constructor & Destructor Documentation

5.96.2.1 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.

5.96.2.2 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.

5.96.3 Member Function Documentation

5.96.3.1 virtual void Arc::Plexer::Next (MCCInterface * *next*, const std::string & *label*) [virtual]

Add reference to next **MCC** (p. 181) in chain.

This method is called by **Loader** (p. 170) for every potentially labeled link to next component which implements **MCCInterface** (p. 187). If next is set NULL corresponding link is removed.

Reimplemented from **Arc::MCC** (p. 182).

5.96.3.2 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** (p. 182).

5.96.4 Field Documentation

5.96.4.1 Logger Arc::Plexer::logger [static]

A logger for MCCs.

A logger intended to be the parent of loggers in the different MCCs.

Reimplemented from **Arc::MCC** (p. 183).

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

- Plexer.h

5.97 Arc::PlexerEntry Class Reference

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

`#include <Plexer.h>`

5.97.1 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:

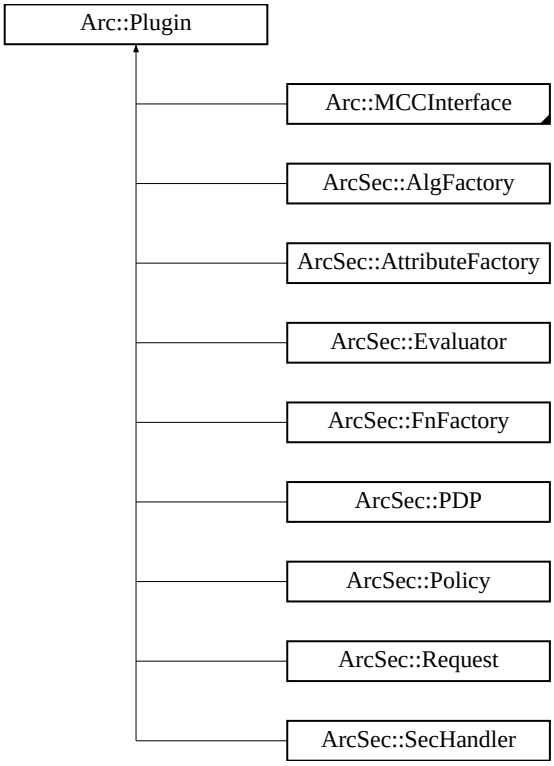
- Plexer.h

5.98 Arc::Plugin Class Reference

Base class for loadable ARC components.

`#include <Plugin.h>`

Inheritance diagram for Arc::Plugin::



5.98.1 Detailed Description

Base class for loadable ARC components.

All classes representing loadable ARC components must be either descendants of this class or be wrapped by its offspring.

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

- Plugin.h

5.99 Arc::PluginArgument Class Reference

Base class for passing arguments to loadable ARC components.

`#include <Plugin.h>`

Inherited by Arc::ACCPluginArgument, Arc::ClassLoaderPluginArgument, Arc::DMCPluginArgument, Arc::MCCPluginArgument, Arc::ServicePluginArgument, ArcSec::PDPPluginArgument, and ArcSec::SecHandlerPluginArgument.

5.99.1 Detailed Description

Base class for passing arguments to loadable ARC components.

During its creation constructor function of ARC loadable component expects instance of class inherited from this one or wrapped in it. Then dynamic type casting is used for obtaining class of expected kind.

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

- Plugin.h

5.100 Arc::PluginDescriptor Struct Reference

Description of ARC lodable component.

`#include <Plugin.h>`

5.100.1 Detailed Description

Description of ARC lodable component.

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

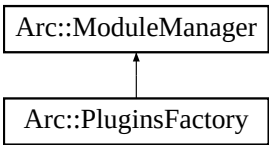
- Plugin.h

5.101 Arc::PluginsFactory Class Reference

Generic ARC plugins loader.

`#include <Plugin.h>`

Inheritance diagram for Arc::PluginsFactory::



Public Member Functions

- **PluginsFactory** (const **Config** &cfg)
- **Plugin** ∗ **get_instance** (const std::string &kind, **PluginArgument** ∗arg)
- bool **load** (const std::string &name)

5.101.1 Detailed Description

Generic ARC plugins loader.

The instance of this class provides functionality of loading pluggable ARC components stored in shared libraries. For more information please check HED documentation.

5.101.2 Constructor & Destructor Documentation

5.101.2.1 Arc::PluginsFactory::PluginsFactory (const Config & *cfg*)

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

5.101.3 Member Function Documentation

5.101.3.1 Plugin∗ Arc::PluginsFactory::get_instance (const std::string & *kind*, PluginArgument ∗ *arg*)

These methods load shared library named lib'name', locate plugin constructor functions of specified 'kind' and 'name' (if specified) and call it. Supplied argument affects way plugin instance is created in plugin-specific way. If name of plugin is not specified then all plugins of specified kind are tried with supplied argument till valid instance is created. All loaded plugins are also registered in internal list of this instance of **PluginsFactory** (p. 234) class. Returns created instance.

5.101.3.2 bool Arc::PluginsFactory::load (const std::string & *name*)

These methods load shared library named lib'name' and check if it contains ARC plugins of specified 'kind'. If there are no specified plugins or if library does not contain any plugins it is unloaded. All loaded plugins are also registered in internal list of this instance of **PluginsFactory** (p. 234) class. Returns true if library was loaded.

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

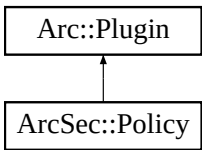
- Plugin.h

5.102 ArcSec::Policy Class Reference

Interface for containing and processing different types of policy.

`#include <Policy.h>`

Inheritance diagram for ArcSec::Policy::



Public Member Functions

- **Policy** ()
- **Policy** (const **Arc::XMLNode**)
- **Policy** (const **Arc::XMLNode**, **EvaluatorContext** ∗)
- virtual **operator bool** (void) const =0
- virtual MatchResult **match** (**EvaluationCtx** ∗)=0
- virtual Result **eval** (**EvaluationCtx** ∗)=0
- virtual void **addPolicy** (**Policy** ∗pl)
- virtual void **setEvaluatorContext** (**EvaluatorContext** ∗)
- virtual void **make_policy** ()
- virtual std::string **getEffect** () const =0
- virtual **EvalResult** & **getEvalResult** ()=0
- virtual void **setEvalResult** (**EvalResult** &res)=0
- virtual const char ∗ **getEvalName** () const =0
- virtual const char ∗ **getName** () const =0

5.102.1 Detailed Description

Interface for containing and processing different types of policy.

Basically, each policy object is a container which includes a few elements e.g., ArcPolicySet objects includes a few ArcPolicy objects; ArcPolicy object includes a few ArcRule objects. There is logical relationship between ArcRules or ArcPolicies, which is called combining algorithm. According to algorithm, evaluation results from the elements are combined, and then the combined evaluation result is returned to the up-level.

5.102.2 Constructor & Destructor Documentation

5.102.2.1 ArcSec::Policy::Policy () [inline]

Template constructor - creates empty policy.

5.102.2.2 ArcSec::Policy::Policy (const Arc::XMLNode) [inline]

Template constructor - creates policy based on XML document.

If XML document is empty then empty policy is created. If it is not empty then it must be valid policy document - otherwise created object should be invalid.

5.102.2.3 ArcSec::Policy::Policy (const Arc::XMLNode, EvaluatorContext *) [inline]

Template constructor - creates policy based on XML document.

If XML document is empty then empty policy is created. If it is not empty then it must be valid policy document - otherwise created object should be invalid. This constructor is based on the policy node and i the **EvaluatorContext** (p. 140) which includes the factory objects for combining algorithm and function

5.102.3 Member Function Documentation

5.102.3.1 virtual void ArcSec::Policy::addPolicy (Policy * *pl*) [inline, virtual]

Add a policy element to into "this" object

5.102.3.2 virtual Result ArcSec::Policy::eval (EvaluationCtx *) [pure virtual]

Evaluate policy For the <Rule> of **Arc** (p. 11), only get the "Effect" from rules; For the <Policy> of **Arc** (p. 11), combine the evaluation result from <Rule>; For the <Rule> of XACML, evaluate the <Condition> node by using information from request, and use the "Effect" attribute of <Rule>; For the <Policy> of XACML, combine the evaluation result from <Rule>

5.102.3.3 virtual std::string ArcSec::Policy::getEffect () const [pure virtual]

Get the "Effect" attribute

5.102.3.4 virtual const char* ArcSec::Policy::getEvalName () const [pure virtual]

Get the name of **Evaluator** (p. 137) which can evaluate this policy

5.102.3.5 virtual EvalResult& ArcSec::Policy::getEvalResult () [pure virtual]

Get eveluation result

5.102.3.6 virtual const char* ArcSec::Policy::getName () const [pure virtual]

Get the name of this policy

5.102.3.7 virtual void ArcSec::Policy::make_policy () [inline, virtual]

Parse XMLNode, and construct the low-level Rule object

5.102.3.8 virtual MatchResult ArcSec::Policy::match (EvaluationCtx *) [pure virtual]

Evaluate whether the two targets to be evaluated match to each other.

5.102.3.9 virtual ArcSec::Policy::operator bool (void) const [pure virtual]

Returns true is object is valid.

5.102.3.10 virtual void ArcSec::Policy::setEvalResult (EvalResult & *res*) [pure virtual]

Set eveluation result

5.102.3.11 virtual void ArcSec::Policy::setEvaluatorContext (EvaluatorContext *) [inline, virtual]

Set **Evaluator** (p. 137) Context for the usage in creating low-level policy object

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

- Policy.h

5.103 ArcSec::PolicyParser Class Reference

A interface which will isolate the policy object from actual policy storage (files, urls, database).

`#include <PolicyParser.h>`

Public Member Functions

- virtual **Policy** ∗ **parsePolicy** (const **Source** &source, std::string policyclassname, **EvaluatorContext** ∗ctx)

5.103.1 Detailed Description

A interface which will isolate the policy object from actual policy storage (files, urls, database).

Parse the policy from policy source (e.g. files, urls, database, etc.).

5.103.2 Member Function Documentation

5.103.2.1 virtual Policy∗ ArcSec::PolicyParser::parsePolicy (const Source & *source*, std::string *policyclassname*, EvaluatorContext ∗ *ctx*) [virtual]

Parse policy

Parameters:

source location of the policy

policyclassname name of the policy for ClassLoader

ctx **EvaluatorContext** (p. 140) which includes the ∗∗Factory

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

- PolicyParser.h

5.104 ArcSec::PolicyStore Class Reference

Storage place for policy objects.

`#include <PolicyStore.h>`

Data Structures

- class **PolicyElement**

Public Member Functions

- **PolicyStore** (const std::string &alg, const std::string &policyclassname, **EvaluatorContext** *ctx)

5.104.1 Detailed Description

Storage place for policy objects.

5.104.2 Constructor & Destructor Documentation

5.104.2.1 ArcSec::PolicyStore::PolicyStore (const std::string & *alg*, const std::string & *policyclassname*, EvaluatorContext * *ctx*)

Creates policy store with specified combing algorithm (alg - not used yet), policy name (policyclassname) and context (ctx)

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

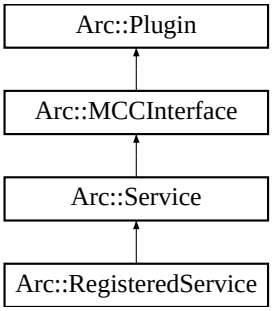
- PolicyStore.h

5.105 Arc::RegisteredService Class Reference

Service (p. 270) - last component in a **Message** (p. 192) Chain.

`#include <RegisteredService.h>`

Inheritance diagram for Arc::RegisteredService::



Public Member Functions

- **RegisteredService** (**Config** ∗)

5.105.1 Detailed Description

Service (p. 270) - last component in a **Message** (p. 192) Chain.

This class which defines interface and common functionality for every **Service** (p. 270) plugin. Interface is made of method **process()** (p. 187) which is called by **Plexer** (p. 228) or **MCC** (p. 181) class. There is one **Service** (p. 270) object created for every service description processed by **Loader** (p. 170) class objects. Classes derived from **Service** (p. 270) class must implement **process()** (p. 187) method of **MCCInterface** (p. 187). It is up to developer how internal state of service is stored and communicated to other services and external utilites. **Service** (p. 270) 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** (p. 181) it must accept and generate messages with **PayloadSOAP** (p. 214) payload. Method **process()** (p. 187) of class derived from **Service** (p. 270) 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 .

5.105.2 Constructor & Destructor Documentation

5.105.2.1 Arc::RegisteredService::RegisteredService (Config ∗)

Example contructor - Server takes at least it's configuration subtree

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

- RegisteredService.h

5.106 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** () const

5.106.1 Detailed Description

A regular expression class.

This class is a wrapper around the functions provided in regex.h.

5.106.2 Constructor & Destructor Documentation

5.106.2.1 Arc::RegularExpression::RegularExpression () `[inline]`

default constructor

5.106.2.2 Arc::RegularExpression::RegularExpression (std::string *pattern*)

Creates a reges from a pattern string.

5.106.2.3 Arc::RegularExpression::RegularExpression (const RegularExpression & *regex*)

Copy constructor.

5.106.2.4 Arc::RegularExpression::~RegularExpression ()

Destructor.

5.106.3 Member Function Documentation

5.106.3.1 std::string Arc::RegularExpression::getPattern () const

Returns patter.

5.106.3.2 bool Arc::RegularExpression::hasPattern (std::string *str*)

Returns true if this regex has the pattern provided.

5.106.3.3 bool Arc::RegularExpression::isOk ()

Returns true if the pattern of this regex is ok.

5.106.3.4 bool Arc::RegularExpression::match (const std::string & *str*, std::list< std::string > & *unmatched*, std::list< std::string > & *matched*) const

Returns true if this regex matches the string provided. Unmatched parts of the string are stored in 'unmatched'. Matched parts of the string are stored in 'matched'.

5.106.3.5 bool Arc::RegularExpression::match (const std::string & *str*) const

Returns true if this regex matches whole string provided.

5.106.3.6 const RegularExpression& Arc::RegularExpression::operator= (const RegularExpression & *regex*)

Assignment operator.

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

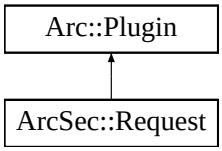
- ArcRegex.h

5.107 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
- virtual const char * **getEvalName** () const =0
- virtual const char * **getName** () const =0
- **Request** ()
- **Request** (const **Source** &)

5.107.1 Detailed Description

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

A **Request** (p. 244) object can has a few <subjects, actions, objects> tuples, i.e. **RequestItem** (p. 247) The **Request** (p. 244) 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** (p. 244) name="arc.request" /> <.....> </pdp:PDPConfig> </Service>

There can be different types of subclass which inherit **Request** (p. 244), such like XACMLRequest, ArcRequest, GACLRequest

5.107.2 Constructor & Destructor Documentation

5.107.2.1 ArcSec::Request::Request () `[inline]`

Default constructor

5.107.2.2 ArcSec::Request::Request (const Source &) `[inline]`

Constructor: Parse request information from a xml stucture in memory

5.107.3 Member Function Documentation

5.107.3.1 virtual void ArcSec::Request::addRequestItem (Attrs & *sub*, Attrs & *res*, Attrs & *act*, Attrs & *ctx*) [pure virtual]

Add request tuple from non-XMLNode

5.107.3.2 virtual const char* ArcSec::Request::getEvalName () const [pure virtual]

Get the name of corresponding evaulator

5.107.3.3 virtual const char* ArcSec::Request::getName () const [pure virtual]

Get the name of this request

5.107.3.4 virtual ReqItemList ArcSec::Request::getRequestItems () const [pure virtual]

Get all the **RequestItem** (p. 247) inside **RequestItem** (p. 247) container

5.107.3.5 virtual void ArcSec::Request::make_request () [pure virtual]

Create the objects included in **Request** (p. 244) according to the node attached to the **Request** (p. 244) object

5.107.3.6 virtual void ArcSec::Request::setAttributeFactory (AttributeFactory * *attributefactory*) [pure virtual]

Set the attribute factory for the usage of **Request** (p. 244)

5.107.3.7 virtual void ArcSec::Request::setRequestItems (ReqItemList *sl*) [pure virtual]

Set the content of the container

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

- Request.h

5.108 ArcSec::RequestAttribute Class Reference

Wrapper which includes **AttributeValue** (p. 43) 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)
- **RequestAttribute** & **duplicate** (**RequestAttribute** &)

5.108.1 Detailed Description

Wrapper which includes **AttributeValue** (p. 43) object which is generated according to date type of one spefic node in Request.xml.

5.108.2 Constructor & Destructor Documentation

5.108.2.1 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>

5.108.3 Member Function Documentation

5.108.3.1 RequestAttribute& ArcSec::RequestAttribute::duplicate (RequestAttribute &)

Duplicate the parameter into "this"

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

- RequestAttribute.h

5.109 ArcSec::RequestItem Class Reference

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

`#include <RequestItem.h>`

Public Member Functions

- **RequestItem** (**Arc::XMLNode** &, **AttributeFactory** ∗)

5.109.1 Detailed Description

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

5.109.2 Constructor & Destructor Documentation

5.109.2.1 ArcSec::RequestItem::RequestItem (Arc::XMLNode &, AttributeFactory ∗) [inline]

Constructor

Parameters:

node The XMLNode structure of the request item

attributefactory The **AttributeFactory** (p. 38) which will be used to generate **RequestAttribute** (p. 246)

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

- RequestItem.h

5.110 ArcSec::RequestTuple Class Reference

RequestTuple (p. 248), container which includes the.

`#include <EvaluationCtx.h>`

5.110.1 Detailed Description

RequestTuple (p. 248), container which includes the.

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

- EvaluationCtx.h

5.111 ArcSec::Response Class Reference

Container for the evaluation results.

`#include <Response.h>`

5.111.1 Detailed Description

Container for the evaluation results.

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

- Response.h

5.112 ArcSec::ResponseItem Struct Reference

Evaluation result concerning one **RequestTuple** (p. 248).

`#include <Response.h>`

5.112.1 Detailed Description

Evaluation result concerning one **RequestTuple** (p. 248).

Include the **RequestTuple** (p. 248), 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

5.113 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 **AssignWorkingDirectory** (std::string &wd)
- void **Kill** (int timeout)

5.113.1 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.

5.113.2 Constructor & Destructor Documentation

5.113.2.1 Arc::Run::Run (const std::string & *cmdline*)

Constructor preapres object to run cmdline

5.113.2.2 Arc::Run::Run (const std::list< std::string > & *argv*)

Constructor preapres object to run executable and arguments specified in argv

5.113.2.3 Arc::Run::~Run (void)

Destructor kill running executable and releases associated resources

5.113.3 Member Function Documentation

5.113.3.1 void Arc::Run::AssignStderr (std::string & *str*)

Associate stderr handle of executable with string. This method must be called before **Start()** (p. 253). str object must be valid as long as this object exists.

5.113.3.2 void Arc::Run::AssignStdin (std::string & *str*)

Associate stdin handle of executable with string. This method must be called before **Start()** (p. 253). str object must be valid as long as this object exists.

5.113.3.3 void Arc::Run::AssignStdout (std::string & *str*)

Associate stdout handle of executable with string. This method must be called before **Start()** (p. 253). str object must be valid as long as this object exists.

5.113.3.4 void Arc::Run::AssignWorkingDirectory (std::string & *wd*) [inline]

Assign working direcotry of the running process

5.113.3.5 void Arc::Run::CloseStderr (void)

Closes pipe associated with stderr handle

5.113.3.6 void Arc::Run::CloseStdin (void)

Closes pipe associated with stdin handle

5.113.3.7 void Arc::Run::CloseStdout (void)

Closes pipe associated with stdout handle

5.113.3.8 void Arc::Run::KeepStderr (bool *keep* = true)

Keep stderr same as parent's if keep = true

5.113.3.9 void Arc::Run::KeepStdin (bool *keep* = true)

Keep stdin same as parent's if keep = true

5.113.3.10 void Arc::Run::KeepStdout (bool *keep* = true)

Keep stdout same as parent's if keep = true

5.113.3.11 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

5.113.3.12 Arc::Run::operator bool (void) [inline]

Returns true if object is valid

5.113.3.13 bool Arc::Run::operator! (void) [inline]

Returns true if object is invalid

5.113.3.14 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.

5.113.3.15 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.

5.113.3.16 int Arc::Run::Result (void) [inline]

Returns exit code of execution.

5.113.3.17 bool Arc::Run::Running (void)

Return true if execution is going on.

5.113.3.18 bool Arc::Run::Start (void)

Starts running executable. This method may be called only once.

5.113.3.19 bool Arc::Run::Wait (void)

Wait till execution finished

5.113.3.20 bool Arc::Run::Wait (int *timeout*)

Wait till execution finished or till timeout seconds expires. Returns true if execution is complete.

5.113.3.21 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.

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

- Run.h

5.114 RuntimeEnvironment Class Reference

`#include <runtimeenvironment.h>`

Public Member Functions

- **RuntimeEnvironment** (const std::string &re)
- ~**RuntimeEnvironment** ()
- std::string **str** () const
- std::string **Name** () const
- std::string **Version** () const
- bool **operator==** (const **RuntimeEnvironment** &other) const
- bool **operator!=** (const **RuntimeEnvironment** &other) const
- bool **operator**> (const **RuntimeEnvironment** &other) const
- bool **operator**< (const **RuntimeEnvironment** &other) const
- bool **operator**>= (const **RuntimeEnvironment** &other) const
- bool **operator**<= (const **RuntimeEnvironment** &other) const

5.114.1 Detailed Description

RuntimeEnvironment (p. 255) class. It represents a runtime environment, and provides functionality for getting information about them.

5.114.2 Constructor & Destructor Documentation

5.114.2.1 RuntimeEnvironment::RuntimeEnvironment (const std::string & *re*)

Constructs a new runtime environemt. String should in general be of the type: STRING-VERSION. Where version consists of numbers with . between them.

5.114.2.2 RuntimeEnvironment::~RuntimeEnvironment ()

Destructor. Not that much to say.

5.114.3 Member Function Documentation

5.114.3.1 std::string RuntimeEnvironment::Name () const

Returns the name of the runtime environment.

5.114.3.2 bool RuntimeEnvironment::operator!= (const RuntimeEnvironment & *other*) const

Inequility operator. Return the opsite of ==

5.114.3.3 bool RuntimeEnvironment::operator< (const RuntimeEnvironment & *other*) const

Less than operator. Returns false if the other is equal, otherwise it returns the opposite of >

5.114.3.4 bool RuntimeEnvironment::operator<= (const RuntimeEnvironment & *other*) const

Less than or equal operator. Returns the oppsite of >

5.114.3.5 bool RuntimeEnvironment::operator== (const RuntimeEnvironment & *other*) const

Equliaty operator. Returns true if the runtime environments have the string representation.

5.114.3.6 bool RuntimeEnvironment::operator> (const RuntimeEnvironment & *other*) const

Greater than operator. Returns true if the compared runtime environment is greater than the current.

5.114.3.7 bool RuntimeEnvironment::operator>= (const RuntimeEnvironment & *other*) const

Greater or equal operator. Returns the opposite of <

5.114.3.8 std::string RuntimeEnvironment::str () const

Returns a string representation of the runtime environment. This is usually the same as given in the constructor.

5.114.3.9 std::string RuntimeEnvironment::Version () const

Returns the version of the runtime environment.

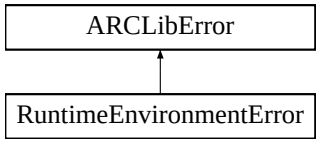
The documentation for this class was generated from the following file:

- runtimeenvironment.h

5.115 RuntimeEnvironmentError Class Reference

`#include <runtimeenvironment.h>`

Inheritance diagram for RuntimeEnvironmentError::



Public Member Functions

- **RuntimeEnvironmentError** (std::string message)

5.115.1 Detailed Description

Interface for handling runtime environments. **RuntimeEnvironment** (p. 255) exceptions. Gets thrown when an error occurs regarding a runtime environment.

5.115.2 Constructor & Destructor Documentation

5.115.2.1 RuntimeEnvironmentError::RuntimeEnvironmentError (std::string *message*) [inline]

Standard exception class constructor.

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

- runtimeenvironment.h

5.116 Arc::SAMLToken Class Reference

Class for manipulating SAML Token Profile.

`#include <SAMLToken.h>`

Public Types

- enum **SAMLVersion**

Public Member Functions

- **SAMLToken** (SOAPEnvelope &soap)
- **SAMLToken** (SOAPEnvelope &soap, const std::string &certfile, const std::string &keyfile, **SAMLVersion** saml_version=SAML2)
- ~**SAMLToken** (void)
- **operator bool** (void)
- bool **Authenticate** (const std::string &cafile, const std::string &capath)
- bool **Authenticate** (void)

5.116.1 Detailed Description

Class for manipulating SAML Token Profile.

This class is for generating/consuming SAML Token profile. See WS-Security SAML Token Profile v1.1 (www.oasis-open.org/committees/wss) Currently this class is used by samltoken handler (will appears in src/hed/pdc/samltokensh/) It is not a must to directly called this class. If we need to use SAML Token functionality, we only need to configure the samltoken handler into service and client. Currently, only a minor part of the specification has been implemented.

About how to identify and reference security token for signing message, currently, only the "SAML Assertion Referenced from KeyInfo" (part 3.4.2 of WS-Security SAML Token Profile v1.1 specification) is supported, which means the implementation can only process SAML assertion "referenced from KeyInfo", and also can only generate SAML Token with SAML assertion "referenced from KeyInfo". More complete support need to implement.

About subject confirmation method, the implementation can process "hold-of-key" (part 3.5.1 of WS-Security SAML Token Profile v1.1 specification) subject subject confirmation method.

About SAML vertion, the implementation can process SAML assertion with SAML version 1.1 and 2.0; can only generate SAML assertion with SAML vertion 2.0.

In the SAML Token profile, for the hold-of-key subject confirmation method, there are three interaction parts: the attesting entity, the relying party and the issuing authority. In the hold-of-key subject confirmation method, it is the attesting entity's subject identity which will be inserted into the SAML assertion.

Firstly the attesting entity authenticates to issuing authority by using some authentication scheme such as WSS x509 Token profile (Alterbatively the usename/password authentication scheme or other different authentication scheme can also be used, unless the issuing authority can retrive the key from a trusted certificate server after firmly establishing the subject's identity under the username/password scheme). So then issuing authority is able to make a definitive statement (sign a SAML assertion) about an act of authentication that has already taken place.

The attesting entity gets the SAML assertion and then signs the soap message together with the assertion by using its private key (the relevant certificate has been authenticated by issuing authority, and its relevant

public key has been put into SubjectConfirmation element under saml assertion by issuing authority. Only the actual owner of the saml assertion can do this, as only the subject possesses the private key paired with the public key in the assertion. This establishes an irrefutable connection between the author of the SOAP message and the assertion describing an authentication event.)

The relying party is supposed to trust the issuing authority. When it receives a message from the asserting entity, it will check the saml assertion based on its predetermined trust relationship with the SAML issuing authority, and check the signature of the soap message based on the public key in the saml assertion without directly trust relationship with attesting entity (subject owner).

5.116.2 Member Enumeration Documentation

5.116.2.1 enum Arc::SAMLToken::SAMLVersion

Since the specfication SAMLVersion is for distinguishing two types of saml version. It is used as the parameter of constructor.

5.116.3 Constructor & Destructor Documentation

5.116.3.1 Arc::SAMLToken::SAMLToken (SOAPEnvelope & *soap*)

Constructor. Parse SAML Token information from SOAP header. SAML Token related information is extracted from SOAP header and stored in class variables. And then it the **SAMLToken** (p. 258) object will be used for authentication.

Parameters:

soap The SOAP message which contains the **SAMLToken** (p. 258) in the soap header

5.116.3.2 Arc::SAMLToken::SAMLToken (SOAPEnvelope & *soap*, const std::string & *certfile*, const std::string & *keyfile*, SAMLVersion *saml_version* = SAML2)

Constructor. Add SAML Token information into the SOAP header. Generated token contains elements SAML token and signature, and is meant to be used for authentication on the consuming side. This constructor is for a specific SAML Token profile usage, in which the attesting entity signs the SAML assertion for itself (self-sign). This usage implicitly requires that the relying party trust the attesting entity. More general (requires issuing authority) usage will be provided by other constructor. And the under-developing SAML service will be used as the issuing authority.

Parameters:

soap The SOAP message to which the SAML Token will be inserted.

certfile The certificate file.

keyfile The key file which will be used to create signature.

samlversion The SAML version, only SAML2 is supported currently.

5.116.3.3 Arc::SAMLToken::~SAMLToken (void)

Deconstructor. Nothing to be done except finalizing the xmlsec library.

5.116.4 Member Function Documentation

5.116.4.1 bool Arc::SAMLToken::Authenticate (void)

Check signature by using the cert information in soap message

5.116.4.2 bool Arc::SAMLToken::Authenticate (const std::string & *cafile*, const std::string & *capath*)

Check signature by using the trusted certificates It is used by relying parting after calling **SAMLToken(SOAPEnvelope& soap)** (p. 259) This method will check the SAML assertion based on the trusted certificated specified as parameter cafile or capath; and also check the signature to soap message (the signature is generated by attesting entity by signing soap body together witl SAML assertion) by using the public key inside SAML assetion.

Parameters:

cafile ca file

capath ca directory

5.116.4.3 Arc::SAMLToken::operator bool (void)

Returns true of constructor succeeded

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

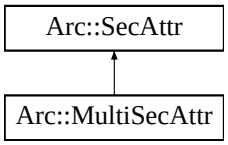
- SAMLToken.h

5.117 Arc::SecAttr Class Reference

This is an abstract interface to a security attribute.

`#include <SecAttr.h>`

Inheritance diagram for Arc::SecAttr::



Public Member Functions

- **SecAttr** ()
- bool **operator==** (const **SecAttr** &b) const
- bool **operator!=** (const **SecAttr** &b) const
- virtual **operator bool** () const
- virtual bool **Export** (**SecAttrFormat** format, std::string &val) const
- virtual bool **Export** (**SecAttrFormat** format, **XMLNode** &val) const
- virtual bool **Import** (**SecAttrFormat** format, const std::string &val)

Static Public Attributes

- static **SecAttrFormat ARCAuth**
- static **SecAttrFormat XACML**
- static **SecAttrFormat SAML**
- static **SecAttrFormat GACL**

5.117.1 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.

5.117.2 Constructor & Destructor Documentation

5.117.2.1 Arc::SecAttr::SecAttr () [inline]

representation for GACL policy

5.117.3 Member Function Documentation

5.117.3.1 virtual bool Arc::SecAttr::Export (SecAttrFormat *format*, XMLNode & *val*) const [virtual]

Convert internal structure into specified format. Returns false if format is not supported/suitable for this attribute. XML node referenced by is turned into top level element of specified format.

Reimplemented in **Arc::MultiSecAttr** (p. 206).

5.117.3.2 virtual bool Arc::SecAttr::Export (SecAttrFormat *format*, std::string & *val*) const [virtual]

Convert internal structure into specified format. Returns false if format is not supported/suitable for this attribute.

5.117.3.3 virtual bool Arc::SecAttr::Import (SecAttrFormat *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.

5.117.3.4 virtual Arc::SecAttr::operator bool () const [virtual]

This function should return false if the value is to be considered null, e.g. if it hasn't been set or initialized. In other cases it should return true.

Reimplemented in **Arc::MultiSecAttr** (p. 206).

5.117.3.5 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.

5.117.3.6 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.

5.117.4 Field Documentation

5.117.4.1 SecAttrFormat Arc::SecAttr::ARCAuth [static]

own serialization/deserialization format

5.117.4.2 SecAttrFormat Arc::SecAttr::GACL [static]

suitable for inclusion into SAML structures

5.117.4.3 SecAttrFormat Arc::SecAttr::SAML [static]

represenation for XACML policy

5.117.4.4 SecAttrFormat Arc::SecAttr::XACML [static]

representation for ARC authorization policy

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

- SecAttr.h

5.118 Arc::SecAttrFormat Class Reference

Export/import format.

`#include <SecAttr.h>`

5.118.1 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:

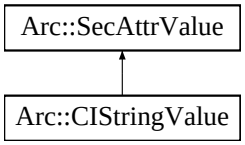
- SecAttr.h

5.119 Arc::SecAttrValue Class Reference

This is an abstract interface to a security attribute.

`#include <SecAttrValue.h>`

Inheritance diagram for Arc::SecAttrValue::



Public Member Functions

- bool **operator==** (**SecAttrValue** &b)
- bool **operator!=** (**SecAttrValue** &b)
- virtual **operator bool** ()

5.119.1 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.

5.119.2 Member Function Documentation

5.119.2.1 virtual Arc::SecAttrValue::operator bool () `[virtual]`

This function should return false if the value is to be considered null, e g if it hasn't been set or initialized. In other cases it should return true.

Reimplemented in **Arc::CIStringValue** (p. 56).

5.119.2.2 bool Arc::SecAttrValue::operator!= (SecAttrValue & *b*)

This is a convenience function to allow the usage of "not equal" conditions and need not be overridden.

5.119.2.3 bool Arc::SecAttrValue::operator== (SecAttrValue & *b*)

This function should (in inheriting classes) return true if this and b are considered to be the same. Identifying and restricting the type of b should be done using dynamic_cast operations.

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

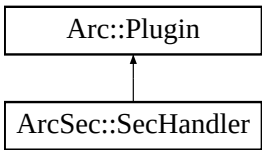
- SecAttrValue.h

5.120 ArcSec::SecHandler Class Reference

Base class for simple security handling plugins.

`#include <SecHandler.h>`

Inheritance diagram for ArcSec::SecHandler::



5.120.1 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** (p. 267) is consumed during creation of instance through XML subtree fed to constructor.

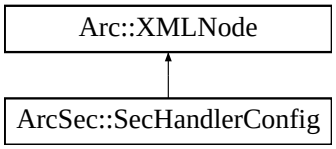
The documentation for this class was generated from the following file:

- SecHandler.h

5.121 ArcSec::SecHandlerConfig Class Reference

`#include <SecHandler.h>`

Inheritance diagram for ArcSec::SecHandlerConfig::



5.121.1 Detailed Description

Helper class to create **Security** (p. 269) Handler configuration

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

- SecHandler.h

5.122 ArcSec::Security Class Reference

Common stuff used by security related slasses.

`#include <Security.h>`

5.122.1 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:

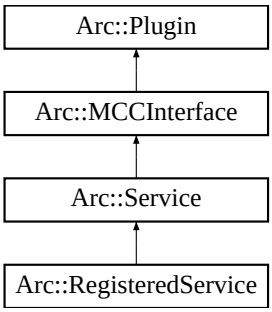
- Security.h

5.123 Arc::Service Class Reference

Service (p. 270) - last component in a **Message** (p. 192) Chain.

`#include <Service.h>`

Inheritance diagram for Arc::Service::



Public Member Functions

- **Service** (**Config** ∗)
- virtual void **AddSecHandler** (**Config** ∗cfg, **ArcSec::SecHandler** ∗sechandler, const std::string &label="")
- virtual bool **RegistrationCollector** (**XMLNode** &doc)
- virtual std::string **getID** ()

Protected Member Functions

- bool **ProcessSecHandlers** (**Message** &message, const std::string &label="")

Protected Attributes

- std::map< std::string, std::list< **ArcSec::SecHandler** ∗ > > **sechandlers_**

Static Protected Attributes

- static **Logger logger**

5.123.1 Detailed Description

Service (p. 270) - last component in a **Message** (p. 192) Chain.

This class which defines interface and common functionality for every **Service** (p. 270) plugin. Interface is made of method **process()** (p. 187) which is called by **Plexer** (p. 228) or **MCC** (p. 181) class. There is one **Service** (p. 270) object created for every service description processed by **Loader** (p. 170) class objects. Classes derived from **Service** (p. 270) class must implement **process()** (p. 187) method of **MCCInterface** (p. 187). It is up to developer how internal state of service is stored and communicated to other services and external utilites. **Service** (p. 270) 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** (p. 181) it must accept and generate messages with **PayloadSOAP** (p. 214) payload. Method **process()** (p. 187) of class derived from **Service** (p. 270) 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 .

5.123.2 Constructor & Destructor Documentation

5.123.2.1 Arc::Service::Service (Config *)

Example contructor - Server takes at least it's configuration subtree

5.123.3 Member Function Documentation

5.123.3.1 virtual void Arc::Service::AddSecHandler (Config * *cfg*, ArcSec::SecHandler * *sechandler*, const std::string & *label* = "") [virtual]

Add security components/handlers to this **MCC** (p. 181). For more information please see description of **MCC::AddSecHandler** (p. 182)

5.123.3.2 virtual std::string Arc::Service::getID () [inline, virtual]

Service (p. 270) may implement own service identitifer gathering method. This method return identifier of service which is used for registering it Information Services.

5.123.3.3 bool Arc::Service::ProcessSecHandlers (Message & *message*, const std::string & *label* = "") [protected]

Executes security handlers of specified queue. For more information please see description of **MCC::ProcessSecHandlers** (p. 182)

5.123.3.4 virtual bool Arc::Service::RegistrationCollector (XMLNode & *doc*) [virtual]

Service (p. 270) specific registartion collector, used for generate service registartions. In implemented service this method should generate GLUE2 document with part of service description which service wishes to advertise to Information Services.

5.123.4 Field Documentation

5.123.4.1 Logger Arc::Service::logger [static, protected]

Logger (p. 173) object used to print messages generated by this class.

5.123.4.2 std::map<std::string,std::list<ArcSec::SecHandler*> > Arc::Service::sechandlers_ [protected]

Set of labeled authentication and authorization handlers. **MCC** (p. 181) 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

5.124 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)

5.124.1 Detailed Description

Simple triggered condition.

Provides condition and semaphor objects in one element.

5.124.2 Member Function Documentation

5.124.2.1 void Arc::SimpleCondition::broadcast (void) [inline]

Signal about condition to all waiting threads

5.124.2.2 void Arc::SimpleCondition::lock (void) [inline]

Acquire semaphor

5.124.2.3 void Arc::SimpleCondition::reset (void) [inline]

Reset object to initial state

5.124.2.4 void Arc::SimpleCondition::signal (void) [inline]

Signal about condition

5.124.2.5 void Arc::SimpleCondition::signal_nonblock (void) [inline]

Signal about condition without using semaphor

5.124.2.6 void Arc::SimpleCondition::unlock (void) [inline]

Release semaphor

5.124.2.7 bool Arc::SimpleCondition::wait (int *t*) [inline]

Wait for condition no longer than t milliseconds

5.124.2.8 void Arc::SimpleCondition::wait (void) [inline]

Wait for condition

5.124.2.9 void Arc::SimpleCondition::wait_nonblock (void) [inline]

Wait for condition without using semaphor

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

- Thread.h

5.125 Arc::SOAPMessage Class Reference

Message (p. 192) restricted to SOAP payload.

`#include <SOAPMessage.h>`

Public Member Functions

- **SOAPMessage** (void)
- **SOAPMessage** (long msg_ptr_addr)
- **SOAPMessage** (**Message** &msg)
- ~**SOAPMessage** (void)
- SOAPEnvelope * **Payload** (void)
- void **Payload** (SOAPEnvelope *new_payload)
- **MessageAttributes** * **Attributes** (void)

5.125.1 Detailed Description

Message (p. 192) restricted to SOAP payload.

This is a special **Message** (p. 192) 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** (p. 192) but can carry only SOAP content.

5.125.2 Constructor & Destructor Documentation

5.125.2.1 Arc::SOAPMessage::SOAPMessage (void) [inline]

Dummy constructor

5.125.2.2 Arc::SOAPMessage::SOAPMessage (long *msg_ptr_addr*)

Copy constructor. Used by language bindigs

5.125.2.3 Arc::SOAPMessage::SOAPMessage (Message & *msg*)

Copy constructor. Ensures shallow copy.

5.125.2.4 Arc::SOAPMessage::~SOAPMessage (void)

Destructor does not affect refered objects

5.125.3 Member Function Documentation

5.125.3.1 MessageAttributes* Arc::SOAPMessage::Attributes (void) [inline]

Returns a pointer to the current attributes object or NULL if no attributes object has been assigned.

5.125.3.2 void Arc::SOAPMessage::Payload (SOAPEnvelope * *new_payload*)

Replace payload with a COPY of new one

5.125.3.3 SOAPEnvelope* Arc::SOAPMessage::Payload (void)

Returns pointer to current payload or NULL if no payload assigned.

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

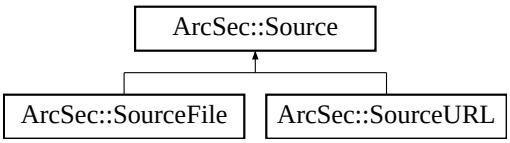
- SOAPMessage.h

5.126 ArcSec::Source Class Reference

Acquires and parses XML document from specified source.

`#include <Source.h>`

Inheritance diagram for ArcSec::Source::



Public Member Functions

- **Source** (const **Source** &s)
- **Source** (**Arc::XMLNode** &xml)
- **Source** (std::istream &stream)
- **Source** (**Arc::URL** &url)
- **Source** (const std::string &str)
- **Arc::XMLNode Get** (void) const
- **operator bool** (void)

5.126.1 Detailed Description

Acquires and parses XML document from specified source.

This class is to be used to provide easy way to specify different sources for XML Authorization Policies and Requests.

5.126.2 Constructor & Destructor Documentation

5.126.2.1 ArcSec::Source::Source (const Source & *s*) `[inline]`

Copy constructor.

Use this constructor only for temporary objects. Parsed XML document is still owned by copied source and hence lifetime of create object should not exceed that of copied one.

5.126.2.2 ArcSec::Source::Source (Arc::XMLNode & *xml*)

Copy XML tree from XML subtree refered by xml.

5.126.2.3 ArcSec::Source::Source (std::istream & *stream*)

Read XML document from stream and parse it.

5.126.2.4 ArcSec::Source::Source (Arc::URL & *url*)

Fetch XML document from specified url and parse it.

This constructor is not implemented yet.

5.126.2.5 ArcSec::Source::Source (const std::string & *str*)

Read XML document from string.

5.126.3 Member Function Documentation

5.126.3.1 Arc::XMLNode ArcSec::Source::Get (void) const [inline]

Get reference to parsed document.

5.126.3.2 ArcSec::Source::operator bool (void) [inline]

Returns true if valid document is available.

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

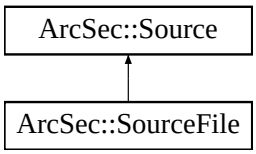
- Source.h

5.127 ArcSec::SourceFile Class Reference

Convenience class for obtaining XML document from file.

`#include <Source.h>`

Inheritance diagram for ArcSec::SourceFile::



Public Member Functions

- **SourceFile** (const **SourceFile** &s)
- **SourceFile** (const char *name)
- **SourceFile** (const std::string &name)

5.127.1 Detailed Description

Convenience class for obtaining XML document from file.

5.127.2 Constructor & Destructor Documentation

5.127.2.1 ArcSec::SourceFile::SourceFile (const SourceFile & *s*) [inline]

See corresponding constructor of **Source** (p. 277) class.

5.127.2.2 ArcSec::SourceFile::SourceFile (const char * *name*)

Read XML document from file named name and store it.

5.127.2.3 ArcSec::SourceFile::SourceFile (const std::string & *name*)

Read XML document from file named name and store it.

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

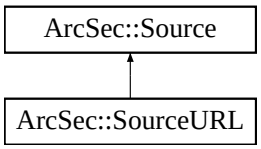
- Source.h

5.128 ArcSec::SourceURL Class Reference

Convenience class for obtaining XML document from remote URL.

`#include <Source.h>`

Inheritance diagram for ArcSec::SourceURL::



Public Member Functions

- **SourceURL** (const **SourceURL** &s)
- **SourceURL** (const char *url)
- **SourceURL** (const std::string &url)

5.128.1 Detailed Description

Convenience class for obtaining XML document from remote URL.

5.128.2 Constructor & Destructor Documentation

5.128.2.1 ArcSec::SourceURL::SourceURL (const SourceURL & *s*) [inline]

See corresponding constructor of **Source** (p. 277) class.

5.128.2.2 ArcSec::SourceURL::SourceURL (const char * *url*)

Read XML document from URL url and store it.

5.128.2.3 ArcSec::SourceURL::SourceURL (const std::string & *url*)

Read XML document from URL url and store it.

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

- Source.h

5.129 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** ()

5.129.1 Detailed Description

A class for storing and manipulating times.

5.129.2 Constructor & Destructor Documentation

5.129.2.1 Arc::Time::Time ()

Default constructor. The time is put equal the current time.

5.129.2.2 Arc::Time::Time (const time_t &)

Constructor that takes a time_t variable and stores it.

5.129.2.3 Arc::Time::Time (const std::string &)

Constructor that tries to convert a string into a time_t.

5.129.3 Member Function Documentation

5.129.3.1 static TimeFormat Arc::Time::GetFormat () [static]

Gets the default format for time strings.

5.129.3.2 time_t Arc::Time::GetTime () const

gets the time

5.129.3.3 Arc::Time::operator std::string () const

Returns a string representation of the time, using the default format.

5.129.3.4 bool Arc::Time::operator!= (const Time &) const

Comparing two **Time** (p. 281) objects.

5.129.3.5 Time Arc::Time::operator+ (const Period &) const

Adding **Time** (p. 281) object with Period object.

5.129.3.6 Period Arc::Time::operator- (const Time &) const

Subtracting **Time** (p. 281) object from the other **Time** (p. 281) object.

5.129.3.7 Time Arc::Time::operator- (const Period &) const

Subtracting Period object from **Time** (p. 281) object.

5.129.3.8 bool Arc::Time::operator< (const Time &) const

Comparing two **Time** (p. 281) objects.

5.129.3.9 bool Arc::Time::operator<= (const Time &) const

Comparing two **Time** (p. 281) objects.

5.129.3.10 Time& Arc::Time::operator= (const Time &)

Assignment operator from a **Time** (p. 281).

5.129.3.11 Time& Arc::Time::operator= (const time_t &)

Assignment operator from a time_t.

5.129.3.12 bool Arc::Time::operator== (const Time &) const

Comparing two **Time** (p. 281) objects.

5.129.3.13 bool Arc::Time::operator> (const Time &) const

Comparing two **Time** (p. 281) objects.

5.129.3.14 bool Arc::Time::operator>= (const Time &) const

Comparing two **Time** (p. 281) objects.

5.129.3.15 static void Arc::Time::SetFormat (const TimeFormat &) [static]

Sets the default format for time strings.

5.129.3.16 void Arc::Time::SetTime (const time_t &)

sets the time

5.129.3.17 std::string Arc::Time::str (const TimeFormat & = time_format) const

Returns a string representation of the time, using the specified format.

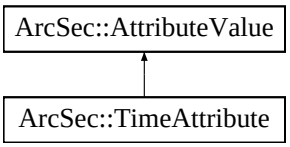
The documentation for this class was generated from the following file:

- DateTime.h

5.130 ArcSec::TimeAttribute Class Reference

`#include <DateTimeAttribute.h>`

Inheritance diagram for ArcSec::TimeAttribute::



Public Member Functions

- virtual bool **equal** (**AttributeValue** *other)
- virtual std::string **encode** ()
- virtual std::string **getType** ()
- virtual std::string **getId** ()

5.130.1 Detailed Description

Format: HHMMSSZ HH:MM:SS HH:MM:SS+HH:MM HH:MM:SSZ

5.130.2 Member Function Documentation

5.130.2.1 virtual std::string ArcSec::TimeAttribute::encode () [virtual]

encode the value in a string format

Implements **ArcSec::AttributeValue** (p. 43).

5.130.2.2 virtual bool ArcSec::TimeAttribute::equal (AttributeValue * *value*) [virtual]

Evluate whether "this" equale to the parameter value

Implements **ArcSec::AttributeValue** (p. 43).

5.130.2.3 virtual std::string ArcSec::TimeAttribute::getId () [inline, virtual]

Get the identifier of the <Attribute>

Implements **ArcSec::AttributeValue** (p. 43).

5.130.2.4 virtual std::string ArcSec::TimeAttribute::getType () [inline, virtual]

Get the type of the <Attribute>

Implements **ArcSec::AttributeValue** (p. 44).

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

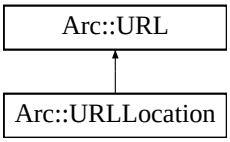
- DateTimeAttribute.h

5.131 Arc::URL Class Reference

Class to hold general URL's.

`#include <URL.h>`

Inheritance diagram for Arc::URL::



Public Types

- enum **Scope**

Public Member Functions

- **URL** ()
- **URL** (const std::string &url)
- virtual ~**URL** ()
- const std::string & **Protocol** () const
- void **ChangeProtocol** (const std::string &newprot)
- const std::string & **Username** () const
- const std::string & **Passwd** () const
- const std::string & **Host** () const
- void **ChangeHost** (const std::string &newhost)
- int **Port** () const
- void **ChangePort** (int newport)
- const std::string & **Path** () const
- std::string **FullPath** () const
- 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
- void **AddLDAPAttribute** (const std::string &attribute)
- **Scope LDAPScope** () const
- void **ChangeLDAPScope** (const **Scope** newscope)
- const std::string & **LDAPFilter** () const
- void **ChangeLDAPFilter** (const std::string &newfilter)
- const std::map< std::string, std::string > & **Options** () const
- const std::string & **Option** (const std::string &option, const std::string &undefined="") const
- const std::map< std::string, std::string > & **MetaDataOptions** () const
- const std::string & **MetaDataOption** (const std::string &option, const std::string &undefined="") const
- 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

Static Public Member Functions

- static std::string **OptionString** (const std::map< std::string, std::string > &options, char separator)

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::map< std::string, std::string > **metadataoptions**
- std::list< std::string > **ldapattributes**
- **Scope ldapscope**
- std::string **ldapfilter**
- std::map< std::string, std::string > **urloptions**
- std::list< **URLLocation** > **locations**
- std::map< std::string, std::string > **commonlocoptions**

Friends

- std::ostream & **operator<<** (std::ostream &out, const **URL** &u)

5.131.1 Detailed Description

Class to hold general URL's.

The **URL** (p. 286) is split into protocol, hostname, port and path. It also accepts file paths which are converted to `file://path`. Usual system dependant file paths are supported. File path can't start from # symbol (why?). If string representation of **URL** (p. 286) starts from '@' then it is treated as path to file containing list of URLs.

5.131.2 Member Enumeration Documentation

5.131.2.1 enum Arc::URL::Scope

Scope for LDAP URLs

5.131.3 Constructor & Destructor Documentation

5.131.3.1 Arc::URL::URL ()

Empty constructor. Necessary when the class is part of another class and the like.

5.131.3.2 Arc::URL::URL (const std::string & *url*)

Constructs a new **URL** (p. 286) from a string representation.

5.131.3.3 virtual Arc::URL::~URL () [virtual]

URL (p. 286) Destructor

5.131.4 Member Function Documentation

5.131.4.1 void Arc::URL::AddLDAPAttribute (const std::string & *attribute*)

Adds an LDAP attribute.

5.131.4.2 void Arc::URL::AddOption (const std::string & *option*, const std::string & *value*, bool *overwrite* = true)

Adds a **URL** (p. 286) option.

5.131.4.3 static std::string Arc::URL::BaseDN2Path (const std::string &) [static, protected]

a private method that converts an ldap basedn to a path.

5.131.4.4 void Arc::URL::ChangeHost (const std::string & *newhost*)

Changes the hostname of the **URL** (p. 286).

5.131.4.5 void Arc::URL::ChangeLDAPFilter (const std::string & *newfilter*)

Changes the LDAP filter.

5.131.4.6 void Arc::URL::ChangeLDAPScope (const Scope *newscope*)

Changes the LDAP scope.

5.131.4.7 void Arc::URL::ChangePath (const std::string & *newpath*)

Changes the path of the **URL** (p. 286).

5.131.4.8 void Arc::URL::ChangePort (int *newport*)

Changes the port of the **URL** (p. 286).

5.131.4.9 void Arc::URL::ChangeProtocol (const std::string & *newprot*)

Changes the protocol of the **URL** (p. 286).

5.131.4.10 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.

5.131.4.11 const std::map<std::string, std::string>& Arc::URL::CommonLocOptions () const

Returns the common location options if any.

5.131.4.12 virtual std::string Arc::URL::ConnectionURL () const [virtual]

Returns a string representation with protocol, host and port only

5.131.4.13 std::string Arc::URL::FullPath () const

Returns the path of the **URL** (p. 286) with all options attached.

5.131.4.14 virtual std::string Arc::URL::fullstr () const [virtual]

Returns a string representation including options and locations

Reimplemented in **Arc::URLLocation** (p. 295).

5.131.4.15 const std::string& Arc::URL::Host () const

Returns the hostname of the **URL** (p. 286).

5.131.4.16 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.

5.131.4.17 const std::map<std::string, std::string>& Arc::URL::HTTPOptions () const

Returns HTTP options if any.

5.131.4.18 const std::list<std::string>& Arc::URL::LDAPAttributes () const

Returns the LDAP attributes if any.

5.131.4.19 const std::string& Arc::URL::LDAPFilter () const

Returns the LDAP filter.

5.131.4.20 Scope Arc::URL::LDAPScope () const

Returns the LDAP scope.

5.131.4.21 const std::list<URLLocation>& Arc::URL::Locations () const

Returns the locations if any.

5.131.4.22 const std::string& Arc::URL::MetaDataOption (const std::string & *option*, const std::string & *undefined* = "") const

Returns the value of a metadata option.

Parameters:

option The option whose value is returned.

undefined This value is returned if the metadata option is not defined.

5.131.4.23 const std::map<std::string, std::string>& Arc::URL::MetaDataOptions () const

Returns metadata options if any.

5.131.4.24 Arc::URL::operator bool () const

Check if instance holds valid **URL** (p. 286)

5.131.4.25 bool Arc::URL::operator< (const URL & *url*) const

Compares one **URL** (p. 286) to another

5.131.4.26 bool Arc::URL::operator== (const URL & *url*) const

Is one **URL** (p. 286) equal to another?

5.131.4.27 const std::string& Arc::URL::Option (const std::string & *option*, const std::string & *undefined* = "") const

Returns the value of a **URL** (p. 286) option.

Parameters:

option The option whose value is returned.

undefined This value is returned if the **URL** (p. 286) option is not defined.

5.131.4.28 const std::map<std::string, std::string>& Arc::URL::Options () const

Returns **URL** (p. 286) options if any.

5.131.4.29 static std::string Arc::URL::OptionString (const std::map< std::string, std::string > & *options*, char *separator*) [static]

Returns a string representation of the options given in the options map

5.131.4.30 const std::string& Arc::URL::Passwd () const

Returns the password of the **URL** (p. 286).

5.131.4.31 const std::string& Arc::URL::Path () const

Returns the path of the **URL** (p. 286).

5.131.4.32 static std::string Arc::URL::Path2BaseDN (const std::string &) [static, protected]

a private method that converts an ldap path to a basedn.

5.131.4.33 int Arc::URL::Port () const

Returns the port of the **URL** (p. 286).

5.131.4.34 const std::string& Arc::URL::Protocol () const

Returns the protocol of the **URL** (p. 286).

5.131.4.35 virtual std::string Arc::URL::str () const [virtual]

Returns a string representation of the **URL** (p. 286).
Reimplemented in **Arc::URLLocation** (p. 295).

5.131.4.36 const std::string& Arc::URL::Username () const

Returns the username of the **URL** (p. 286).

5.131.5 Friends And Related Function Documentation

5.131.5.1 std::ostream& operator<< (std::ostream & *out*, const URL & *u*) [friend]

Overloaded operator << to print a **URL** (p. 286).

5.131.6 Field Documentation

5.131.6.1 std::map<std::string, std::string> Arc::URL::commonlocoptions [protected]

common location options for index server URLs.

5.131.6.2 std::string Arc::URL::host [protected]

hostname of the url.

5.131.6.3 std::map<std::string, std::string> Arc::URL::httpoptions [protected]

HTTP options of the url.

5.131.6.4 std::list<std::string> Arc::URL::ldapattributes [protected]

LDAP attributes of the url.

5.131.6.5 std::string Arc::URL::ldapfilter [protected]

LDAP filter of the url.

5.131.6.6 Scope Arc::URL::ldapscope [protected]

LDAP scope of the url.

5.131.6.7 std::list<URLLocation> Arc::URL::locations [protected]

locations for index server URLs.

5.131.6.8 std::map<std::string, std::string> Arc::URL::metadataoptions [protected]

Meta data options

5.131.6.9 std::string Arc::URL::passwd [protected]

password of the url.

5.131.6.10 std::string Arc::URL::path [protected]

the url path.

5.131.6.11 int Arc::URL::port [protected]

portnumber of the url.

5.131.6.12 std::string Arc::URL::protocol [protected]

the url protocol.

5.131.6.13 std::map<std::string, std::string> Arc::URL::urloptions [protected]

options of the url.

5.131.6.14 std::string Arc::URL::username [protected]

username of the url.

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

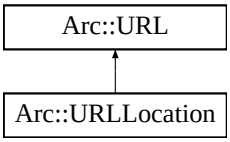
- URL.h

5.132 Arc::URLLocation Class Reference

Class to hold a resolved **URL** (p. 286) 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**

5.132.1 Detailed Description

Class to hold a resolved **URL** (p. 286) location.

It is specific to file indexing service registrations.

5.132.2 Constructor & Destructor Documentation

5.132.2.1 Arc::URLLocation::URLLocation (const std::string & *url*)

Creates a **URLLocation** (p. 294) from a string representaion.

5.132.2.2 Arc::URLLocation::URLLocation (const std::string & *url*, const std::string & *name*)

Creates a **URLLocation** (p. 294) from a string representaion and a name.

5.132.2.3 Arc::URLLocation::URLLocation (const URL & *url*)

Creates a **URLLocation** (p. 294) from a **URL** (p. 286).

5.132.2.4 Arc::URLLocation::URLLocation (const URL & *url*, const std::string & *name*)

Creates a **URLLocation** (p. 294) from a **URL** (p. 286) and a name.

5.132.2.5 Arc::URLLocation::URLLocation (const std::map< std::string, std::string > & *options*, const std::string & *name*)

Creates a **URLLocation** (p. 294) from options and a name.

5.132.2.6 virtual Arc::URLLocation::~URLLocation () [virtual]

URLLocation (p. 294) destructor.

5.132.3 Member Function Documentation

5.132.3.1 virtual std::string Arc::URLLocation::fullstr () const [virtual]

Returns a string representation including options and locations

Reimplemented from **Arc::URL** (p. 289).

5.132.3.2 const std::string& Arc::URLLocation::Name () const

Returns the **URLLocation** (p. 294) name.

5.132.3.3 virtual std::string Arc::URLLocation::str () const [virtual]

Returns a string representation of the **URLLocation** (p. 294).

Reimplemented from **Arc::URL** (p. 292).

5.132.4 Field Documentation

5.132.4.1 std::string Arc::URLLocation::name [protected]

the **URLLocation** (p. 294) name as registered in the indexing service.

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

- URL.h

5.133 Arc::UsernameToken Class Reference

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

`#include <UsernameToken.h>`

Public Types

- enum **PasswordType**

Public Member Functions

- **UsernameToken** (SOAPEnvelope &soap)
- **UsernameToken** (SOAPEnvelope &soap, const std::string &username, const std::string &password, const std::string &uid, **PasswordType** pwdtype)
- **UsernameToken** (SOAPEnvelope &soap, const std::string &username, const std::string &id, bool mac, int iteration)
- **operator bool** (void)
- std::string **Username** (void)
- bool **Authenticate** (const std::string &password, std::string &derived_key)
- bool **Authenticate** (std::istream &password, std::string &derived_key)

5.133.1 Detailed Description

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

5.133.2 Member Enumeration Documentation

5.133.2.1 enum Arc::UsernameToken::PasswordType

SOAP header element

5.133.3 Constructor & Destructor Documentation

5.133.3.1 Arc::UsernameToken::UsernameToken (SOAPEnvelope & *soap*)

Link to existing SOAP header and parse Username Token information. Username Token related information is extracted from SOAP header and stored in class variables.

5.133.3.2 Arc::UsernameToken::UsernameToken (SOAPEnvelope & *soap*, const std::string & *username*, const std::string & *password*, const std::string & *uid*, PasswordType *pwdtype*)

Add Username Token information into the SOAP header. Generated token contains elements Username and Password and is meant to be used for authentication.

Parameters:

soap the SOAP message

username <wsse:Username>...</wsse:Username> - if empty it is entered interactively from stdin

password <wsse:Password Type="...">...</wsse:Password> - if empty it is entered interactively from stdin

uid <wsse:**UsernameToken** (p. 296) wsu:ID="...">

pwdtype <wsse:Password Type="...">...</wsse:Password>

5.133.3.3 Arc::UsernameToken::UsernameToken (SOAPEnvelope & *soap*, const std::string & *username*, const std::string & *id*, bool *mac*, int *iteration*)

Add Username Token information into the SOAP header. Generated token contains elements Username and Salt and is meant to be used for deriving Key Derivation.

Parameters:

soap the SOAP message

username <wsse:Username>...</wsse:Username>

mac if derived key is meant to be used for **Message** (p. 192) Authentication Code

iteration <wsse11:Iteration>...</wsse11:Iteration>

5.133.4 Member Function Documentation

5.133.4.1 bool Arc::UsernameToken::Authenticate (std::istream & *password*, std::string & *derived_key*)

Checks parsed token against password stored in specified stream. If token is meant to be used for deriving a key then key is returned in derived_key

5.133.4.2 bool Arc::UsernameToken::Authenticate (const std::string & *password*, std::string & *derived_key*)

Checks parsed/generated token against specified password. If token is meant to be used for deriving a key then key is returned in derived_key. In that case authentication is performed outside of **UsernameToken** (p. 296) class using obtained derived_key.

5.133.4.3 Arc::UsernameToken::operator bool (void)

Returns true of constructor succeeded

5.133.4.4 std::string Arc::UsernameToken::Username (void)

Returns username associated with this instance

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

- UsernameToken.h

5.134 Arc::UserSwitch Class Reference

`#include <User.h>`

5.134.1 Detailed Description

If this class is created user identity is switched to provided uid and gid. Due to internal lock there will be only one valid instance of this class. Any attempt to create another instance will block till first one is destroyed. If uid and gid are set to 0 then user identity is not switched. But lock is applied anyway. The lock has dual purpose. First and most important is to protect communication with underlying operating system which may depend on user identity. For that it is advisable for code which talks to operating system to acquire valid instance of this class. Care must be taken for not to hold that instance too long cause that may block other code in multithreaded envoronment. Other purpose of this lock is to provide workaround for glibc bug in __nptl_setxid. That bug causes lockup of seteuid() function if racing with fork. To avoid this problem the lock mentioned above is used by **Run** (p. 251) class while spawning new process.

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

- User.h

5.135 Arc::VOMSTrustList Class Reference

`#include <VOMSUtil.h>`

Public Member Functions

- **VOMSTrustList** (const std::vector< std::string > &encoded_list)
- **VOMSTrustList** (const std::vector< VOMSTrustChain > &chains, const std::vector< VOMSTrustRegex > &regexs)
- VOMSTrustChain & **AddChain** (const VOMSTrustChain &chain)
- VOMSTrustChain & **AddChain** (void)
- **RegularExpression** & **AddRegex** (const VOMSTrustRegex &reg)

5.135.1 Detailed Description

Stores definitions for making decision if VOMS server is trusted

5.135.2 Constructor & Destructor Documentation

5.135.2.1 Arc::VOMSTrustList::VOMSTrustList (const std::vector< std::string > & *encoded_list*)

Creates chain lists and regexps from plain list. List is made of chunks delimited by elements containing pattern "NEXT CHAIN". Each chunk with more than one element is converted into one instance of VOMSTrustChain. Chunks with single element are converted to VOMSTrustChain if element does not have special symbols. Otherwise it is treated as regular expression. Those symbols are '^','$' and '∗'. Trusted chains can be congicured in two ways: one way is: <tls:VOMSCertTrustDNChain> <tls:VOMSCertTrustDN>/O=Grid/O=NorduGrid/CN=host/arthur.hep.lu.se</tls:VOMSCertTrustDN> <tls:VOMSCertTrustDN>/O=Grid/O=NorduGrid/CN=NorduGrid Certification Authority</tls:VOMSCertTrustDN> <tls:VOMSCertTrustDN>—-NEXT CHAIN— </tls:VOMSCertTrustDN> <tls:VOMSCertTrustDN>/DC=ch/DC=cern/OU=computers/CN=voms.cern.ch</tls:VOMSCertT <tls:VOMSCertTrustDN>/DC=ch/DC=cern/CN=CERN Trusted Certification Authority</tls:VOMSCertTrustDN> </tls:VOMSCertTrustDNChain> the other way is: <tls:VOMSCertTrustDNChain> <tls:VOMSCertTrustDN>/O=Grid/O=NorduGrid/CN=host/arthur.hep.lu.se</tls:VOMSCert <tls:VOMSCertTrustDN>/O=Grid/O=NorduGrid/CN=NorduGrid Certification Authority</tls:VOMSCertTrustDN> </tls:VOMSCertTrustDNChain> <tls:VOMSCertTrustDNChain> <tls:VOMSCertTrustDN>/DC=ch/DC=cern/OU=computers/CN=voms.cern.ch</tls:VOMSCertTrustDN> <tls:VOMSCertTrustDN>/DC=ch/DC=cern/CN=CERN Trusted Certification Authority</tls:VOMSCertTrustDN> </tls:VOMSCertTrustDNChain> each chunk is supposed to contain a suit of DN of trusted certificate chain, in which the first DN is the DN of the certificate (cert0) which is used to sign the Attribute Certificate (AC), the second DN is the DN of the issuer certificate(cert1) which is used to sign cert0. So if there are one or more intermediate issuers, then there should be 3 or more than 3 DNs in this chunk (considering cert0 and the root certificate, plus the intermediate certificate) .

5.135.2.2 Arc::VOMSTrustList::VOMSTrustList (const std::vector< VOMSTrustChain > & *chains*, const std::vector< VOMSTrustRegex > & *regexs*)

Creates chain lists and regexps from those specified in arguments. See **AddChain()** (p. 300) and **AddRegex()** (p. 300) for more information.

5.135.3 Member Function Documentation

5.135.3.1 VOMSTrustChain& Arc::VOMSTrustList::AddChain (void)

Adds empty chain of trusted DNs to list.

5.135.3.2 VOMSTrustChain& Arc::VOMSTrustList::AddChain (const VOMSTrustChain & *chain*)

Adds chain of trusted DNs to list. During verification each signature of AC is checked against all stored chains. DNs of chain of certificate used for signing AC are compared against DNs stored in these chains one by one. If needed DN of issuer of last certificate is checked too. Comparison succeeds if DNs in at least one stored chain are same as those in certificate chain. Comparison stops when all DNs in stored chain are compared. If there are more DNs in stored chain than in certificate chain then comparison fails. Empty stored list matches any certificate chain. Taking into account that certificate chains are verified down to trusted CA anyway, having more than one DN in stored chain seems to be useless. But such feature may be found useful by some very strict sysadmins. ??? IMO,DN list here is not only for authentication, it is also kind of ACL, which means the AC consumer only trusts those DNs which issues AC.

5.135.3.3 RegularExpression& Arc::VOMSTrustList::AddRegex (const VOMSTrustRegex & *reg*)

Adds regular expression to list. During verification each signature of AC is checked against all stored regular expressions. DN of signing certificate must match at least one of stored regular expressions.

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

- VOMSUtil.h

5.136 Arc::WSAEndpointReference Class Reference

Interface for manipulation of WS-Adressing Endpoint Reference.

`#include <WSA.h>`

Public Member Functions

- **WSAEndpointReference** (const **XMLNode** &epr)
- **WSAEndpointReference** (const **WSAEndpointReference** &wsa)
- **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)

5.136.1 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.

5.136.2 Constructor & Destructor Documentation

5.136.2.1 Arc::WSAEndpointReference::WSAEndpointReference (const XMLNode & *epr*)

Link to top level EPR XML node Linking to existing EPR in XML tree

5.136.2.2 Arc::WSAEndpointReference::WSAEndpointReference (const WSAEndpointReference & *wsa*)

Copy constructor

5.136.2.3 Arc::WSAEndpointReference::WSAEndpointReference (const std::string & *address*)

Creating independent EPR - not implemented

5.136.2.4 Arc::WSAEndpointReference::WSAEndpointReference (void)

Dummy constructor - creates invalid instance

5.136.2.5 Arc::WSAEndpointReference::~WSAEndpointReference (void)

Destructor. All empty elements of EPR XML are destroyed here too

5.136.3 Member Function Documentation

5.136.3.1 void Arc::WSAEndpointReference::Address (const std::string & *uri*)

Assigns new Address value. If EPR had no Address element it is created.

5.136.3.2 std::string Arc::WSAEndpointReference::Address (void) const

Returns Address (**URL** (p. 286)) encoded in EPR

5.136.3.3 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.

5.136.3.4 Arc::WSAEndpointReference::operator XMLNode (void)

Returns reference to EPR top XML node

5.136.3.5 WSAEndpointReference& Arc::WSAEndpointReference::operator= (const std::string & *address*)

Same as Address(uri)

5.136.3.6 XMLNode Arc::WSAEndpointReference::ReferenceParameters (void)

Access to ReferenceParameters element of EPR. Obtained XML element should be manipulated directly in application-dependent way. If EPR had no ReferenceParameters element it is created.

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

- WSA.h

5.137 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

- bool **header_allocated_**

5.137.1 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.

5.137.2 Constructor & Destructor Documentation

5.137.2.1 Arc::WSAHeader::WSAHeader (SOAPEnvelope & *soap*)

Linking to a header of existing SOAP message

5.137.2.2 Arc::WSAHeader::WSAHeader (const std::string & *action*)

Creating independent SOAP header - not implemented

5.137.3 Member Function Documentation

5.137.3.1 void Arc::WSAHeader::Action (const std::string & *uri*)

Set content of Action element of SOAP Header. If such element does not exist it's created.

5.137.3.2 std::string Arc::WSAHeader::Action (void) const

Returns content of Action element of SOAP Header.

5.137.3.3 static bool Arc::WSAHeader::Check (SOAPEnvelope & *soap*) [static]

Tells if specified SOAP message has WSA header

5.137.3.4 WSAEndpointReference Arc::WSAHeader::FaultTo (void)

Returns FaultTo element of SOAP Header. If such element does not exist it's created. Obtained element may be manipulted.

5.137.3.5 WSAEndpointReference Arc::WSAHeader::From (void)

Returns From element of SOAP Header. If such element does not exist it's created. Obtained element may be manipulted.

5.137.3.6 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.

5.137.3.7 std::string Arc::WSAHeader::MessageID (void) const

Returns content of MessageID element of SOAP Header.

5.137.3.8 XMLNode Arc::WSAHeader::NewReferenceParameter (const std::string & *name*)

Creates new ReferenceParameter element with specified name. Returns reference to created element.

5.137.3.9 Arc::WSAHeader::operator XMLNode (void)

Returns reference to SOAP Header - not implemented

5.137.3.10 XMLNode Arc::WSAHeader::ReferenceParameter (const std::string & *name*)

Returns first ReferenceParameter element with specified name

5.137.3.11 XMLNode Arc::WSAHeader::ReferenceParameter (int *n*)

Return n-th ReferenceParameter element

5.137.3.12 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.

5.137.3.13 std::string Arc::WSAHeader::RelatesTo (void) const

Returns content of RelatesTo element of SOAP Header.

5.137.3.14 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.

5.137.3.15 std::string Arc::WSAHeader::RelationshipType (void) const

Returns content of RelationshipType element of SOAP Header.

5.137.3.16 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.

5.137.3.17 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.

5.137.3.18 std::string Arc::WSAHeader::To (void) const

Returns content of To element of SOAP Header.

5.137.4 Field Documentation

5.137.4.1 bool Arc::WSAHeader::header_allocated_ [protected]

SOAP header element

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

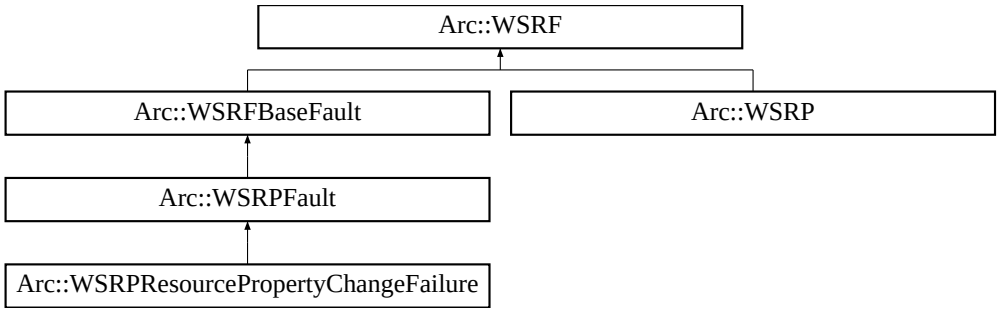
- WSA.h

5.138 Arc::WSRF Class Reference

Base class for every **WSRF** (p. 306) message.

`#include <WSRF.h>`

Inheritance diagram for Arc::WSRF::



Public Member Functions

- **WSRF** (SOAPEnvelope &soap, const std::string &action="")
- **WSRF** (bool fault=false, const std::string &action="")
- virtual SOAPEnvelope & **SOAP** (void)
- virtual **operator bool** (void)

Protected Member Functions

- void **set_namespaces** (void)

Protected Attributes

- bool **allocated_**
- bool **valid_**

5.138.1 Detailed Description

Base class for every **WSRF** (p. 306) message.

This class is not intended to be used directly. Use it like reference while passing through unknown **WSRF** (p. 306) message or use classes derived from it.

5.138.2 Constructor & Destructor Documentation

5.138.2.1 Arc::WSRF::WSRF (SOAPEnvelope & *soap*, const std::string & *action* = "")

Constructor - creates object out of supplied SOAP tree.

5.138.2.2 Arc::WSRF::WSRF (bool *fault* = false, const std::string & *action* = "")

Constructor - creates new **WSRF** (p. 306) object

5.138.3 Member Function Documentation

5.138.3.1 virtual Arc::WSRF::operator bool (void) [inline, virtual]

Returns true if instance is valid

References valid_.

5.138.3.2 void Arc::WSRF::set_namespaces (void) [protected]

true if object represents valid **WSRF** (p. 306) message set WS Resource namespaces and default prefixes in SOAP message

Reimplemented in **Arc::WSRP** (p. 310), and **Arc::WSRFBaseFault** (p. 308).

5.138.3.3 virtual SOAPEnvelope& Arc::WSRF::SOAP (void) [inline, virtual]

Direct access to underlying SOAP element

5.138.4 Field Documentation

5.138.4.1 bool Arc::WSRF::allocated_ [protected]

Associated SOAP message - it's SOAP message after all

5.138.4.2 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:

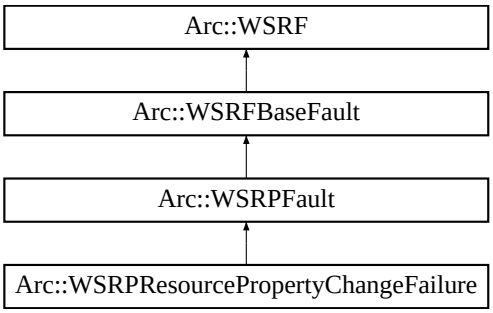
- WSRF.h

5.139 Arc::WSRFBaseFault Class Reference

Base class for **WSRF** (p. 306) fault messages.

`#include <WSRFBaseFault.h>`

Inheritance diagram for Arc::WSRFBaseFault::



Public Member Functions

- **WSRFBaseFault** (SOAPEnvelope &soap)
- **WSRFBaseFault** (const std::string &type)

Protected Member Functions

- void **set_namespaces** (void)

5.139.1 Detailed Description

Base class for **WSRF** (p. 306) fault messages.

Use classes inherited from it for specific faults.

5.139.2 Constructor & Destructor Documentation

5.139.2.1 Arc::WSRFBaseFault::WSRFBaseFault (SOAPEnvelope & *soap*)

Constructor - creates object out of supplied SOAP tree.

5.139.2.2 Arc::WSRFBaseFault::WSRFBaseFault (const std::string & *type*)

Constructor - creates new **WSRF** (p. 306) fault

5.139.3 Member Function Documentation

5.139.3.1 void Arc::WSRFBaseFault::set_namespaces (void) [protected]

set WS-ResourceProperties namespaces and default prefixes in SOAP message

Reimplemented from **Arc::WSRF** (p. 307).

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

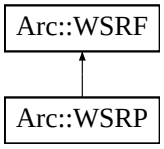
- WSRFBaseFault.h

5.140 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)

5.140.1 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.

5.140.2 Constructor & Destructor Documentation

5.140.2.1 Arc::WSRP::WSRP (bool *fault* = `false`, const std::string & *action* = `""`)

Constructor - prepares object for creation of new **WSRP** (p. 310) request/response/fault

5.140.2.2 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.

5.140.3 Member Function Documentation

5.140.3.1 void Arc::WSRP::set_namespaces (void) `[protected]`

set WS-ResourceProperties namespaces and default prefixes in SOAP message

Reimplemented from **Arc::WSRF** (p. 307).

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

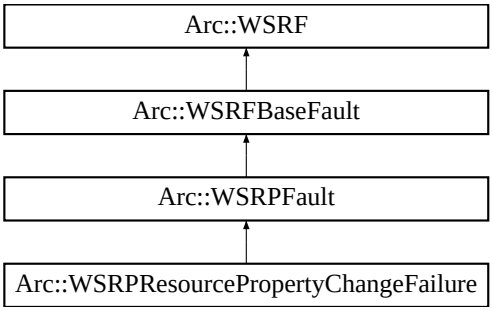
- WSResourceProperties.h

5.141 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)

5.141.1 Detailed Description

Base class for WS-ResourceProperties faults.

5.141.2 Constructor & Destructor Documentation

5.141.2.1 Arc::WSRPFault::WSRPFault (SOAPEnvelope & *soap*)

Constructor - creates object out of supplied SOAP tree.

5.141.2.2 Arc::WSRPFault::WSRPFault (const std::string & *type*)

Constructor - creates new **WSRP** (p. 310) fault

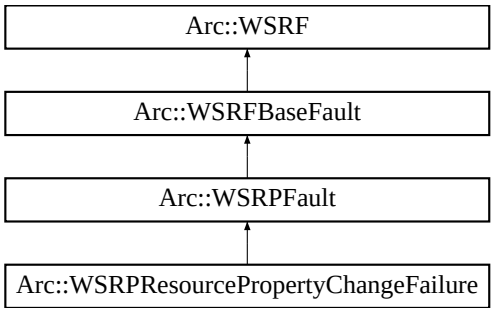
The documentation for this class was generated from the following file:

- WSResourceProperties.h

5.142 Arc::WSRPResourcePropertyChangeFailure Class Reference

`#include <WSResourceProperties.h>`

Inheritance diagram for Arc::WSRPResourcePropertyChangeFailure::



Public Member Functions

- **WSRPResourcePropertyChangeFailure** (SOAPEnvelope &soap)
- **WSRPResourcePropertyChangeFailure** (const std::string &type)

5.142.1 Detailed Description

Base class for WS-ResourceProperties faults which contain ResourcePropertyChangeFailure

5.142.2 Constructor & Destructor Documentation

5.142.2.1 Arc::WSRPResourcePropertyChangeFailure::WSRPResourcePropertyChangeFailure (SOAPEnvelope & *soap*) [inline]

Constructor - creates object out of supplied SOAP tree.

5.142.2.2 Arc::WSRPResourcePropertyChangeFailure::WSRPResourcePropertyChangeFailure (const std::string & *type*) [inline]

Constructor - creates new **WSRP** (p. 310) fault

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

- WSResourceProperties.h

5.143 Arc::X509Token Class Reference

Class for manipulating X.509 Token Profile.

`#include <X509Token.h>`

Public Types

- enum **X509TokenType**

Public Member Functions

- **X509Token** (SOAPEnvelope &soap)
- **X509Token** (SOAPEnvelope &soap, const std::string &certfile, const std::string &keyfile, **X509TokenType** token_type=Signature)
- ~**X509Token** (void)
- **operator bool** (void)
- bool **Authenticate** (const std::string &cafile, const std::string &capath)
- bool **Authenticate** (void)

5.143.1 Detailed Description

Class for manipulating X.509 Token Profile.

This class is for generating/consuming X.509 Token profile. Currently it is used by x509token handler (src/hed/pdc/x509tokensh/) It is not necessary to directly called this class. If we need to use X.509 Token functionality, we only need to configure the x509token handler into service and client.

5.143.2 Member Enumeration Documentation

5.143.2.1 enum Arc::X509Token::X509TokenType

X509TokeType is for distinguishing two types of operation. It is used as the parameter of constuctor.

5.143.3 Constructor & Destructor Documentation

5.143.3.1 Arc::X509Token::X509Token (SOAPEnvelope & *soap*)

Constructor.Parse X509 Token information from SOAP header. X509 Token related information is extracted from SOAP header and stored in class variables. And then it the **X509Token** (p. 314) object will be used for authentication if the tokentype is Signature; otherwise if the tokentype is Encryption, the encrypted soap body will be decrypted and replaced by decrypted message.

5.143.3.2 Arc::X509Token::X509Token (SOAPEnvelope & *soap*, const std::string & *certfile*, const std::string & *keyfile*, X509TokenType *token_type* = `Signature`)

Constructor. Add X509 Token information into the SOAP header. Generated token contains elements X509 token and signature, and is meant to be used for authentication on the consuming side.

Parameters:

soap The SOAP message to which the X509 Token will be inserted

certfile The certificate file which will be used to encrypt the SOAP body (if parameter tokentype is Encryption), or be used as <wsse:BinarySecurityToken/> (if parameter tokentype is Signature).

keyfile The key file which will be used to create signature. Not needed when create encryption.

tokentype Token type: Signature or Encryption.

5.143.3.3 Arc::X509Token::~X509Token (void)

Deconstructor. Nothing to be done except finalizing the xmlsec library.

5.143.4 Member Function Documentation

5.143.4.1 bool Arc::X509Token::Authenticate (void)

Check signature by using the cert information in soap message. Only the signature itself is checked, and it is not guranteed that the certificate which is supposed to check the signature is trusted.

5.143.4.2 bool Arc::X509Token::Authenticate (const std::string & *cafile*, const std::string & *capath*)

Check signature by using the certificare information in **X509Token** (p. 314) which is parsed by the constructor, and the trusted certificates specified as one of the two parameters. Not only the signature (in the **X509Token** (p. 314)) itself is checked, but also the certificate which is supposed to check the signature needs to be trused (which means the certificate is issued by the ca certificate from CA file or CA directory). At least one the the two parameters should be set.

Parameters:

cafile The CA file

capath The CA directory

Returns:

true if authentication passes; otherwise false

5.143.4.3 Arc::X509Token::operator bool (void)

Returns true of constructor succeeded

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

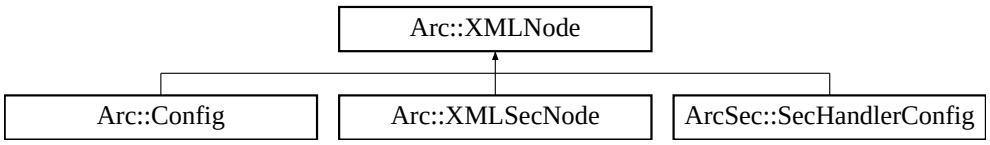
- X509Token.h

5.144 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 NS &ns, const char *name)
- ~**XMLNode** (void)
- void **New** (**XMLNode** &new_node) const
- **operator bool** (void) const
- bool **operator!** (void) const
- bool **operator==** (const **XMLNode** &node)
- bool **operator!=** (const **XMLNode** &node)
- bool **Same** (const **XMLNode** &node)
- bool **operator==** (bool val)
- bool **operator!=** (bool val)
- bool **operator==** (const std::string &str)
- bool **operator!=** (const std::string &str)
- bool **operator==** (const char *str)
- bool **operator!=** (const char *str)
- **XMLNode Child** (int n=0) 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 **GetXML** (std::string &out_xml_str, const std::string &encoding, bool user_friendly=false) const

- void **GetDoc** (std::string &out_xml_str, bool user_friendly=false) const
- **operator std::string** (void) const
- **XMLNode** & **operator=** (const char *content)
- **XMLNode** & **operator=** (const std::string &content)
- void **Set** (const std::string &content)
- **XMLNode** & **operator=** (const **XMLNode** &node)
- **XMLNode Attribute** (int n=0) 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 **Path** (const std::string &path) const
- XMLNodeList **XPathLookup** (const std::string &xpathExpr, const NS &nsList) const
- **XMLNode GetRoot** (void)
- **XMLNode Parent** (void)
- bool **SaveToFile** (const std::string &file_name) const
- bool **SaveToStream** (std::ostream &out) const
- bool **ReadFromFile** (const std::string &file_name)
- bool **ReadFromStream** (std::istream &in)

Protected Member Functions

- **XMLNode** (xmlNodePtr node)

Protected Attributes

- bool **is_owner_**
- bool **is_temporary_**

Friends

- bool **MatchXMLName** (const **XMLNode** &node1, const **XMLNode** &node2)
- bool **MatchXMLName** (const **XMLNode** &node, const char *name)
- bool **MatchXMLName** (const **XMLNode** &node, const std::string &name)
- bool **MatchXMLNamespace** (const **XMLNode** &node1, const **XMLNode** &node2)
- bool **MatchXMLNamespace** (const **XMLNode** &node, const char *uri)
- bool **MatchXMLNamespace** (const **XMLNode** &node, const std::string &uri)

5.144.1 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.

5.144.2 Constructor & Destructor Documentation

5.144.2.1 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.

5.144.2.2 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.

5.144.2.3 Arc::XMLNode::XMLNode (const XMLNode & *node*) [inline]

Copies existing instance. Underlying XML element is NOT copied. Ownership is NOT inherited.

5.144.2.4 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

5.144.2.5 Arc::XMLNode::XMLNode (const char * *xml*, int *len* = -1)

Same as previous

5.144.2.6 Arc::XMLNode::XMLNode (const NS & *ns*, const char * *name*)

Creates empty XML document structure with specified namespaces. Created XML contains only root element named 'name'. Created structure is pointed and owned by constructed instance

5.144.2.7 Arc::XMLNode::~XMLNode (void)

Destructor Also destroys underlying XML document if owned by this instance

5.144.3 Member Function Documentation

5.144.3.1 XMLNode Arc::XMLNode::Attribute (const std::string & *name*) const [inline]

Returns **XMLNode** (p. 316) instance representing first attribute of node with specified by name

References Attribute().

5.144.3.2 XMLNode Arc::XMLNode::Attribute (const char * *name*) const

Returns **XMLNode** (p. 316) instance representing first attribute of node with specified by name

5.144.3.3 XMLNode Arc::XMLNode::Attribute (int *n* = 0) const

Returns list of all attributes of node.

Returns **XMLNode** (p. 316) instance reresenting n-th attribute of node.

Referenced by Attribute().

5.144.3.4 int Arc::XMLNode::AttributesSize (void) const

Returns number of attributes of node

5.144.3.5 XMLNode Arc::XMLNode::Child (int *n* = 0) const

Returns **XMLNode** (p. 316) instance representing n-th child of XML element. If such does not exist invalid **XMLNode** (p. 316) instance is returned Returns **XMLNode** (p. 316) instance representing n-th child of XML element. If such does not exist invalid **XMLNode** (p. 316) instance is returned

5.144.3.6 void Arc::XMLNode::Destroy (void)

Destroys underlying XML element. XML element is unlinked from XML tree and destroyed. After this operation **XMLNode** (p. 316) instance becomes invalid

5.144.3.7 std::string Arc::XMLNode::FullName (void) const [inline]

Returns prefix:name of XML node

References Name(), and Prefix().

5.144.3.8 XMLNode Arc::XMLNode::Get (const std::string & *name*) const [inline]

Same as operator[]

References operator[]().

5.144.3.9 void Arc::XMLNode::GetDoc (std::string & *out_xml_str*, bool *user_friendly* = false) const

Fills argument with whole XML document textual representation

5.144.3.10 XMLNode Arc::XMLNode::GetRoot (void)

Get the root node from any child node of the tree

5.144.3.11 void Arc::XMLNode::GetXML (std::string & *out_xml_str*, const std::string & *encoding*, bool *user_friendly* = `false`) const

Fills argument with this instance XML subtree textual representation if the XML subtree is corresponding to the encoding format specified in the argument, e.g. utf-8

5.144.3.12 void Arc::XMLNode::GetXML (std::string & *out_xml_str*, bool *user_friendly* = `false`) const

Fills argument with this instance XML subtree textual representation

5.144.3.13 void Arc::XMLNode::Name (const std::string & *name*) `[inline]`

Assigns new name to XML node

References Name().

5.144.3.14 void Arc::XMLNode::Name (const char * *name*)

Assigns new name to XML node

5.144.3.15 std::string Arc::XMLNode::Name (void) const

Returns name of XML node

Referenced by FullName(), and Name().

5.144.3.16 std::string Arc::XMLNode::Namespace (void) const

Returns namespace URI of XML node

5.144.3.17 std::string Arc::XMLNode::NamespacePrefix (const char * *urn*)

Returns prefix of specified namespace. Empty string if no such namespace.

5.144.3.18 NS Arc::XMLNode::Namespaces (void)

Returns namespaces known at this node

5.144.3.19 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 useful to apply this method to XML being processed in order to refer to it's elements by known prefix.

5.144.3.20 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.

5.144.3.21 XMLNode Arc::XMLNode::NewAttribute (const std::string & *name*) [inline]

Creates new attribute with specified name.

References NewAttribute().

5.144.3.22 XMLNode Arc::XMLNode::NewAttribute (const char * *name*)

Creates new attribute with specified name.

Referenced by NewAttribute().

5.144.3.23 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

5.144.3.24 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)** (p. 321)

References NewChild().

5.144.3.25 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)** (p. 321)

5.144.3.26 XMLNode Arc::XMLNode::NewChild (const std::string & *name*, int *n* = -1, bool *global_order* = false) [inline]

Same as **NewChild(const char*,int,bool)** (p. 321)

References NewChild().

5.144.3.27 XMLNode Arc::XMLNode::NewChild (const char * *name*, int *n* = -1, bool *global_order* = false)

Creates new child XML element at specified position with specified name. Default is to put it at end of list. If global order is true position applies to whole set of children, otherwise only to children of same name. Returns created node.

Referenced by NewChild().

5.144.3.28 Arc::XMLNode::operator bool (void) const [inline]

Returns true if instance points to XML element - valid instance

References is_temporary_.

5.144.3.29 Arc::XMLNode::operator std::string (void) const

Returns textual content of node excluding content of children nodes

5.144.3.30 bool Arc::XMLNode::operator! (void) const [inline]

Returns true if instance does not point to XML element - invalid instance

References is_temporary_.

5.144.3.31 bool Arc::XMLNode::operator!= (const char * *str*) [inline]

This operator is needed to avoid ambiguity

5.144.3.32 bool Arc::XMLNode::operator!= (const std::string & *str*) [inline]

This operator is needed to avoid ambiguity

5.144.3.33 bool Arc::XMLNode::operator!= (bool *val*) [inline]

This operator is needed to avoid ambiguity

5.144.3.34 bool Arc::XMLNode::operator!= (const XMLNode & *node*) [inline]

Returns false if 'node' represents same XML element

References node_.

5.144.3.35 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.

5.144.3.36 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.

5.144.3.37 XMLNode& Arc::XMLNode::operator= (const XMLNode & *node*)

Make instance refer to another XML node. Ownership is not inherited.

5.144.3.38 XMLNode& Arc::XMLNode::operator= (const std::string & *content*) [inline]

Sets textual content of node. All existing children nodes are discarded.

References operator=().

5.144.3.39 XMLNode& Arc::XMLNode::operator= (const char * *content*)

Sets textual content of node. All existing children nodes are discarded.

Referenced by operator=(), and Set().

5.144.3.40 bool Arc::XMLNode::operator== (const char * *str*) [inline]

This operator is needed to avoid ambiguity

5.144.3.41 bool Arc::XMLNode::operator== (const std::string & *str*) [inline]

This operator is needed to avoid ambiguity

5.144.3.42 bool Arc::XMLNode::operator== (bool *val*) [inline]

This operator is needed to avoid ambiguity

5.144.3.43 bool Arc::XMLNode::operator== (const XMLNode & *node*) [inline]

Returns true if 'node' represents same XML element

References node_.

Referenced by Same().

5.144.3.44 XMLNode Arc::XMLNode::operator[ ] (int *n*) const

Returns **XMLNode** (p. 316) 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]

5.144.3.45 XMLNode Arc::XMLNode::operator[ ] (const std::string & *name*) const [inline]

Similar to previous method

References operator[ ]().

5.144.3.46 XMLNode Arc::XMLNode::operator[ ] (const char * *name*) const

Returns **XMLNode** (p. 316) 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** (p. 316) instance is returned

Referenced by Get(), and operator[ ]().

5.144.3.47 XMLNode Arc::XMLNode::Parent (void)

Get the parent node from any child node of the tree

5.144.3.48 XMLNodeList Arc::XMLNode::Path (const std::string & *path*) const

Collects nodes corresponding to specified path. This is a convenience function to cover common use of XPath but without performance hit. Path is made of node_name[/node_name[...]] and is relative to current node. node_names are treated in same way as in operator[]. Returns all nodes which are represented by path.

5.144.3.49 std::string Arc::XMLNode::Prefix (void) const

Returns namespace prefix of XML node

Referenced by FullName().

5.144.3.50 bool Arc::XMLNode::ReadFromFile (const std::string & *file_name*)

Read XML document from file and associate it with this node

5.144.3.51 bool Arc::XMLNode::ReadFromStream (std::istream & *in*)

Read XML document from stream and associate it with this node

5.144.3.52 void Arc::XMLNode::Replace (const XMLNode & *node*)

Makes a copy of supplied XML node and makes this instance refere to it

5.144.3.53 bool Arc::XMLNode::Same (const XMLNode & *node*) [inline]

Returns true if 'node' represents same XML element - for bindings

References operator==().

5.144.3.54 bool Arc::XMLNode::SaveToFile (const std::string & *file_name*) const

Save string representation of node to file

5.144.3.55 bool Arc::XMLNode::SaveToStream (std::ostream & *out*) const

Save string representation of node to stream

5.144.3.56 void Arc::XMLNode::Set (const std::string & *content*) [inline]

Same as operator=. Used for bindings.

References operator=().

5.144.3.57 int Arc::XMLNode::Size (void) const

Returns number of children nodes

5.144.3.58 XMLNodeList Arc::XMLNode::XPathLookup (const std::string & *xpathExpr*, const NS & *nsList*) const

Uses xPath to look up the whole xml structure, Returns a list of **XMLNode** (p. 316) 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.

5.144.4 Friends And Related Function Documentation

5.144.4.1 bool MatchXMLName (const XMLNode & *node*, const std::string & *name*) [friend]

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

5.144.4.2 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

5.144.4.3 bool MatchXMLName (const XMLNode & *node1*, const XMLNode & *node2*) [friend]

Returns true if underlying XML elements have same names

5.144.4.4 bool MatchXMLNamespace (const XMLNode & *node*, const std::string & *uri*) [friend]

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

5.144.4.5 bool MatchXMLNamespace (const XMLNode & *node*, const char * *uri*) [friend]

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

5.144.4.6 bool MatchXMLNamespace (const XMLNode & *node1*, const XMLNode & *node2*) [friend]

Returns true if underlying XML elements belong to same namespaces

5.144.5 Field Documentation

5.144.5.1 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.

5.144.5.2 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

5.145 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)

5.145.1 Detailed Description

Container for multiple **XMLNode** (p. 316) elements

5.145.2 Constructor & Destructor Documentation

5.145.2.1 Arc::XMLNodeContainer::XMLNodeContainer (void)

Default constructor

5.145.2.2 Arc::XMLNodeContainer::XMLNodeContainer (const XMLNodeContainer &)

Copy constructor. Add nodes from argument. Nodes owning XML document are copied using **AddNew()** (p. 328). Not owning nodes are linked using **Add()** (p. 327) method.

5.145.3 Member Function Documentation

5.145.3.1 void Arc::XMLNodeContainer::Add (const std::list< XMLNode > &)

Link multiple XML subtrees to container.

5.145.3.2 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.

5.145.3.3 void Arc::XMLNodeContainer::AddNew (const std::list< XMLNode > &)

Copy multiple XML subtrees to container.

5.145.3.4 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.

5.145.3.5 std::list<XMLNode> Arc::XMLNodeContainer::Nodes (void)

Returns all stored nodes.

5.145.3.6 XMLNodeContainer& Arc::XMLNodeContainer::operator= (const XMLNodeContainer &)

Same as copy constructor with current nodes being deleted first.

5.145.3.7 XMLNode Arc::XMLNodeContainer::operator[] (int)

Returns n-th node in a store.

5.145.3.8 int Arc::XMLNodeContainer::Size (void)

Return number of refered/stored nodes.

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

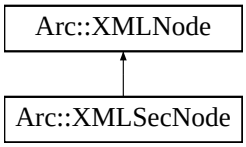
- XMLNode.h

5.146 Arc::XMLSecNode Class Reference

Extends **XMLNode** (p. 316) class to support XML security operation.

`#include <XMLSecNode.h>`

Inheritance diagram for Arc::XMLSecNode::



Public Member Functions

- **XMLSecNode** (**XMLNode** &node)
- void **AddSignatureTemplate** (const std::string &id_name, const SignatureMethod sign_method, const std::string &incl_namespaces="")
- bool **SignNode** (const std::string &privkey_file, const std::string &cert_file)
- bool **VerifyNode** (const std::string &id_name, const std::string &ca_file, const std::string &ca_path, bool verify_trusted=true)
- bool **EncryptNode** (const std::string &cert_file, const SymEncryptionType encrpt_type)
- bool **DecryptNode** (const std::string &privkey_file, **XMLNode** &decrypted_node)

5.146.1 Detailed Description

Extends **XMLNode** (p. 316) class to support XML security operation.

All **XMLNode** (p. 316) methods are exposed by inheriting from **XMLNode** (p. 316). **XMLSecNode** (p. 329) itself does not own node, instead it uses the node from the base class **XMLNode** (p. 316).

5.146.2 Constructor & Destructor Documentation

5.146.2.1 Arc::XMLSecNode::XMLSecNode (XMLNode & *node*)

Create a object based on an **XMLNode** (p. 316) instance.

5.146.3 Member Function Documentation

5.146.3.1 void Arc::XMLSecNode::AddSignatureTemplate (const std::string & *id_name*, const SignatureMethod *sign_method*, const std::string & *incl_namespaces* = "")

Add the signature template for later signing.

Parameters:

id_name The identifier name under this node which will be used for the <Signature> to refer to.

sign_method The sign method for signing. Two options now, RSA_SHA1, DSA_SHA1

5.146.3.2 bool Arc::XMLSecNode::DecryptNode (const std::string & *privkey_file*, XMLNode & *decrypted_node*)

Decrypt the <xenc:EncryptedData/> under this node, the decrypted node will be output in the second argument of DecryptNode method. And the <xenc:EncryptedData/> under this node will be removed after decryption.

Parameters:

privkey_file The private key file, which is used for decrypting

decrypted_node Output the decrypted node

5.146.3.3 bool Arc::XMLSecNode::EncryptNode (const std::string & *cert_file*, const SymEncryptionType *encrpt_type*)

Encrypt this node, after encryption, this node will be replaced by the encrypted node

Parameters:

cert_file The certificate file, the public key parsed from this certificate is used to encrypted the symmetric key, and then the symmetric key is used to encrypted the node

encrpt_type The encryption type when encrypting the node, four option in SymEncryptionType

verify_trusted Verify trusted certificates or not. If set to false, then only the signature will be checked (by using the public key from KeyInfo).

5.146.3.4 bool Arc::XMLSecNode::SignNode (const std::string & *privkey_file*, const std::string & *cert_file*)

Sign this node (identified by id_name).

Parameters:

privkey_file The private key file. The private key is used for signing

cert_file The certificate file. The certificate is used as the <KeyInfo> part of the <Signature>; <KeyInfo> will be used for the other end to verify this <Signature>

incl_namespaces InclusiveNamespaces for Tranform in Signature

5.146.3.5 bool Arc::XMLSecNode::VerifyNode (const std::string & *id_name*, const std::string & *ca_file*, const std::string & *ca_path*, bool *verify_trusted* = `true`)

Verify the signature under this node

Parameters:

id_name The id of this node, which is used for identifying the node

ca_file The CA file which used as trused certificate when verify the certificate in the <KeyInfo> part of <Signature>

ca_path The CA directory; either ca_file or ca_path should be set.

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

- XMLSecNode.h

Index