# Hosting Environment (Daemon)

Generated by Doxygen 1.6.1

Thu Dec 3 07:40:19 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

# File Index

## 4.1 File List

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

# Chapter 5

# Namespace Documentation

## 5.1 Arc Namespace Reference

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

### Data Structures

- class **ARCJSDLParser**
- class **Broker**
- class **BrokerLoader**
- class **BrokerPluginArgument**
- class **ClientInterface**

  *Utility base class for* ***MCC*** *(*p. 278*).*

- class **ClientTCP**

  *Class for setting up a* ***MCC*** *(*p. 278*) chain for TCP communication.*

- struct **HTTPClientInfo**
- class **ClientHTTP**

  *Class for setting up a* ***MCC*** *(*p. 278*) chain for HTTP communication.*

- class **ClientSOAP**
- class **SecHandlerConfig**
- class **DNListHandlerConfig**
- class **ARCPolicyHandlerConfig**
- class **ClientHTTPwithSAML2SSO**
- class **ClientSOAPwithSAML2SSO**
- class **ClientX509Delegation**
- class **ConfusaCertHandler**
- class **ConfusaParserUtils**
- class **HakaClient**
- class **OpenIdpClient**
- class **OAuthConsumer**
- class **SAML2LoginClient**
- class **SAML2SSOHTTPClient**

- class **ApplicationEnvironment**

  ***ApplicationEnvironment** (p. 65).*

- class **ExecutionTarget**

  ***ExecutionTarget** (p. 207).*

- class **JDLParser**
- class **Job**

  ***Job** (p. 251).*

- class **JobController**

  *Base class for the JobControllers.*

- class **JobControllerLoader**
- class **JobControllerPluginArgument**
- class **Range**
- class **ScalableTime**
- class **ScalableTime< int >**
- class **JobIdentificationType**
- class **ExecutableType**
- class **ApplicationType**
- class **ResourceSlotType**
- class **DiskSpaceRequirementType**
- class **ResourceTargetType**
- class **ResourcesType**
- class **DataSourceType**
- class **DataTargetType**
- class **DataType**
- class **FileType**
- class **DirectoryType**
- class **DataStagingType**
- class **JobMetaType**
- class **JobDescription**
- class **JobDescriptionParser**
- class **JobState**
- class **JobSupervisor**

  *% **JobSupervisor** (p. 262) class*

- class **RSLValue**
- class **RSLLiteral**
- class **RSLVariable**
- class **RSLConcat**
- class **RSLList**
- class **RSLSequence**
- class **RSL**
- class **RSLBoolean**
- class **RSLCondition**
- class **RSLParser**
- class **Software**

  *Used to represent software (names and version) and comparison.*

- class **SoftwareRequirement**

  *Class used to express and resolve version requirements on software.*

- class **Submitter**

  *Base class for the Submitters.*

- class **SubmitterLoader**
- class **SubmitterPluginArgument**
- class **TargetGenerator**

  *Target generation class*

- class **TargetRetriever**

  *TargetRetriever base class*

- class **TargetRetrieverLoader**
- class **TargetRetrieverPluginArgument**
- 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 **IniConfig**
- class **IntraProcessCounter**

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

- class **PrintFBase**
- class **PrintF**
- class **IString**
- struct **LoggerFormat**
- class **LogMessage**

  *A class for log messages.*

- class **LogDestination**

  *A base class for log destinations.*

- class **LogStream**

  *A class for logging to ostreams.*

- class **LogFile**

  *A class for logging to files.*

- class **Logger**

  *A logger class.*

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

  *Helper function to create simple thread.*

- class **ThreadRegistry**
- class **ThreadInitializer**
- class **URL**
- class **URLLocation**

  *Class to hold a resolved **URL** (p. 456) location.*

- class **PathIterator**

  *Class to iterate through elements of path.*

- class **User**
- class **UserSwitch**
- class **initializeCredentialsType**
- class **UserConfig**

  *User configuration class*

- 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 **Adler32Sum**

  *Implementation of Adler32 checksum.*

- class **CheckSumAny**

  *Wraper for **CheckSum** (p. 96) class.*

- class **DataBuffer**

  *Represents set of buffers.*

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

  *This class is a wrapper around the **DataPoint** (p. 151) class.*

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

  *This base class is an abstraction of **URL** (p. 456).*

- class **DataPointLoader**
- class **DataPointPluginArgument**
- class **DataPointDirect**

  *This is a kind of generalized file handle.*

- class **DataPointIndex**

  *Complements **DataPoint** (p. 151) with attributes common for Indexing **Service** (p. 409) URLs.*

- class **DataSpeed**

  *Keeps track of average and instantaneous transfer speed.*

- class **DataStatus**

- struct **CacheParameters**
- class **FileCache**
- class **FileInfo**

  ***FileInfo*** *(p. 217) 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**

  ***RegisteredService*** *(p. 365) - extension of* ***Service*** *(p. 409) performing self-registration.*

- class **FinderLoader**

- 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. *278),* ***Service*** *(*p. *409) and **Plexer** (*p. *340) objects.*

- class **MCC**

  ***Message*** *(*p. *290) Chain Component - base class for every **MCC** (*p. *278) plugin.*

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

  *A class for communication of **MCC** (*p. *278) processing results.*

- class **MCCLoader**

  *Creator of **Message** (*p. *290) Component Chains (**MCC** (*p. *278)).*

- 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. *290) objects.*

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

  *Raw byte multi-buffer.*

- class **PayloadSOAP**

  *Payload of* ***Message*** (p. *290) with SOAP content.*

- class **PayloadStreamInterface**

  *Stream-like Payload for* ***Message*** (p. *290) object.*

- class **PayloadStream**

  *POSIX handle as Payload.*

- class **PlexerEntry**

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

- class **Plexer**

  *The* ***Plexer*** (p. *340) 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. *400) attributes.*

- class **Service**

  ***Service*** (p. *409) - last component in a* ***Message*** (p. *290) Chain.*

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

  ***Message*** *(p. 290) 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*** *(p. 357).*

- class **UsernameToken**

  *Interface for manipulation of WS-Security according to Username Token* ***Profile*** *(p. 357).*

- class **X509Token**

  *Class for manipulating X.509 Token* ***Profile*** *(p. 357).*

- class **PayloadWSRF**

  *This class combines* ***MessagePayload*** *(p. 300) with* ***WSRF*** *(p. 503).*

- 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. *503) message.*

- class **WSRFBaseFault**

  *Base class for **WSRF** (*p. *503) fault messages.*

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

  *Extends **XMLNode** (*p. *547) 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)
- **LogLevel** **string_to_level** (const std::string &str)
- bool **istring_to_level** (const std::string &llStr, **LogLevel** &ll)
- std::string **level_to_string** (const **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)
- std::string **convert_to_rdn** (const std::string &dn)
- bool **CreateThreadFunction** (void(∗func)(void ∗), void ∗arg)
- std::list< **URL** > **ReadURLList** (const **URL** &urllist)
- std::string **GetEnv** (const std::string &var)
- std::string **GetEnv** (const std::string &var, bool &found)
- bool **SetEnv** (const std::string &var, const std::string &value)
- 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 &acorder, 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)
- bool **OpenSSLInit** (void)
- void **HandleOpenSSLError** (void)
- void **HandleOpenSSLError** (int code)
- 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**

## 5.1.1 Detailed Description

Some utility methods for using xml security library (`http://www.aleksey.com/xmlsec/`). **ARCJSDLParser** (p. 67) The **ARCJSDLParser** (p. 67) class, derived from the **JobDescriptionParser** (p. 258) class, is primarily a job description parser for the consolidated job description language (ARCJSDL), derived from JSDL, described in the following document <`http://svn.nordugrid.org/trac/nordugrid/browser/arc1/trunk/doc/tech_-doc/client/job_description.odt`>. However it is also capable of parsing regular JSDL (GFD 136), the POSIX-JSDL extension (GFD 136) and the JSDL HPC **Profile** (p. 357) Application Extension (GFD 111 and GFD 114). When parsing ARCJSDL takes precedence over other non-ARCJSDL, so if a non-ARCJSDL element specifies the same attribute as ARCJSDL, the ARCJSDL element will be saved. The output generated by the ARCJSDLParser::UnParse method will follow that of the ARCJSDL document, see reference above.

**JDLParser** (p. 250) The **JDLParser** (p. 250) class, derived from the **JobDescriptionParser** (p. 258) class, is a job description parser for the **Job** (p. 251) Description Language (JDL) specified in CREAM **Job** (p. 251) Description Language Attributes Specification for the EGEE middleware (EGEE-JRA1-TEC-592336) and **Job** (p. 251) Description Language Attributes Specification for the gLite middleware (EGEE-JRA1-TEC-590869-JDL-Attributes-v0-8).

**JobDescription** (p. 257) The **JobDescription** (p. 257) class is the internal representation of a job description in the ARC-lib. It is structured into a number of other classes/objects which should strictly follow the description given in the job description document <`http://svn.nordugrid.org/trac/nordugrid/browser/arc1/trunk/doc/tech_-doc/client/job_description.odt`>.

The class consist of a parsing method JobDescription::Parse which tries to parse the passed source using a number of different parsers. The parser method is complemented by the JobDescription::UnParse method, a method to generate a job description document in one of the supported formats. Additionally the internal representation is contained in public members which makes it directly accessible and modifiable from outside the scope of the class.

**JobDescriptionParser** (p. 258) The **JobDescriptionParser** (p. 258) class is abstract which provide a interface for job description parsers. A job description parser should inherit this class and overwrite the JobDescriptionParser::Parse and JobDescriptionParser::UnParse methods.

**XRSLParser** (p. 562) The **XRSLParser** (p. 562) class, derived from the **JobDescriptionParser** (p. 258) class, is a job description parser for the Extended Resource Specification Language (XRSL) specified in the NORDUGRID-MANUAL-4 document.

**Credential** (p. 129) 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** (p. 129) 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

## 5.1.2 Typedef Documentation

### 5.1.2.1 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. 293) class as well as the AttributesIterator class, but is not visible externally.

### 5.1.2.2 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. 293) class, but is not visible externally.

### 5.1.2.3 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.

### 5.1.2.4 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.

### 5.1.3 Enumeration Type Documentation

#### 5.1.3.1 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. VERBOSE level designates fine-grained informational events that will give additional information about the application. DEBUG level designates finer-grained informational events which should only be used for debugging purposes.

#### 5.1.3.2 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. 290) does not fit into expected protocol.

***BUSY_ERROR*** There is no destination configured for this message.

***SESSION_CLOSE*** **Message** (p. 290) can't be processed now.

#### 5.1.3.3 enum Arc::WSAFault

WS-Addressing possible faults.

**Enumerator:**

***WSAFaultUnknown*** This is not a fault

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

### 5.1.4 Function Documentation

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

Add decoded AC string into a list of AC objects

**Parameters:**

***aclist*** The list of AC objects (output)

***acorder*** The order of AC objects (output)

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

#### 5.1.4.2 const char∗ Arc::ContentFromPayload (const MessagePayload & *payload*)

Returns pointer to main memory chunk of **Message** (p. 290) payload. If no buffer is present or if payload is not of **PayloadRawInterface** (p. 321) type NULL is returned.

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

This macro behaves like function which makes thread of class' method. It accepts class instance and full name of method - like class::method. 'method' should not be static member of the class. Result is true if creation of thread succeeded. Specified instance must be valid during whole lifetime of thread. So probably it is safer to destroy 'instance' in 'method' just before exiting. 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.

#### 5.1.4.4 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

#### 5.1.4.5 bool Arc::final_xmlsec (void)

Finalize the xml security library

#### 5.1.4.6 std::string Arc::get_cert_str (const char ∗ *certfile*)

Get certificate in string format from certificate file

#### 5.1.4.7 std::string Arc::get_key_from_certfile (const char ∗ *certfile*)

Get public key in string format from certificate file

#### 5.1.4.8 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-----")

#### 5.1.4.9 xmlSecKey∗ Arc::get_key_from_keyfile (const char ∗ *keyfile*)

Get key in xmlSecKey structure from key file

#### 5.1.4.10 xmlSecKey∗ Arc::get_key_from_keystr (const std::string & *value*)

Get key in xmlSecKey structure from key in string format

#### 5.1.4.11 XMLNode Arc::get_node (XMLNode & *parent*, const char ∗ *name*)

Generate a new child **XMLNode** (p. 547) with specified name

#### 5.1.4.12 bool Arc::init_xmlsec (void)

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

#### 5.1.4.13 bool Arc::istring_to_level (const std::string & *llStr*, LogLevel & *ll*)

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

**Parameters:**

*llStr* a string which should be parsed to a **Arc::LogLevel** (p. 36).

*ll* a **Arc::LogLevel** (p. 36) reference which will be set to the matching **Arc::LogLevel** (p. 36) upon successful parsing.

**Returns:**

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

**See also:**

**LogLevel** (p. 36)

#### 5.1.4.14 xmlSecKeysMngrPtr Arc::load_key_from_certfile (xmlSecKeysMngrPtr ∗ *keys_manager*, const char ∗ *certfile*)

Load public key from a certificate file into key manager

#### 5.1.4.15 xmlSecKeysMngrPtr Arc::load_key_from_certstr (xmlSecKeysMngrPtr ∗ *keys_manager*, const std::string & *certstr*)

Load public key from a certificate string into key manager

#### 5.1.4.16 xmlSecKeysMngrPtr Arc::load_key_from_keyfile (xmlSecKeysMngrPtr ∗ *keys_manager*, const char ∗ *keyfile*)

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

#### 5.1.4.17 xmlSecKeysMngrPtr Arc::load_trusted_cert_file (xmlSecKeysMngrPtr ∗ *keys_manager*, const char ∗ *cert_file*)

Load trusted certificate from certificate file into key manager

#### 5.1.4.18 xmlSecKeysMngrPtr Arc::load_trusted_cert_str (xmlSecKeysMngrPtr ∗ *keys_manager*, const std::string & *cert_str*)

Load trusted certificate from cetrtificate string into key manager

#### 5.1.4.19 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

#### 5.1.4.20 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

#### 5.1.4.21 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

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

Returns true if underlying XML elements have same names

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

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

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

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

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

Returns true if underlying XML elements belong to same namespaces

#### 5.1.4.26 bool Arc::OpenSSLInit (void)

This module contains various convenience utilities for using OpenSSL. Application may be linked to this module instead of OpenSSL libraries directly. This function initializes OpenSSL library. It may be called multiple times and makes sure everything is done properly and OpenSSL may be used in multi-threaded environment. Because this function makes use of **ArcLocation** (p. 68) it is advisable to call it after **ArcLocation::Init()** (p. 68).

#### 5.1.4.27 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.

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

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

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

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

#### 5.1.4.30 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

#### 5.1.4.31 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: /von-ame=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.

#### 5.1.4.32 int Arc::passphrase_callback (char ∗ *buf*, int *size*, int *rwflag*, void ∗)

callback method for inputing passphrase of key file

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

Conversion to string. Conversion from StatusKind to string.

**Parameters:**

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

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

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

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

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

#### 5.1.4.36 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

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

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

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

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

## 5.1.5 Variable Documentation

### 5.1.5.1 Logger Arc::CredentialLogger

**Logger** (p. 268) to be used by all modules of credentials library

#### 5.1.5.2 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. 346) elements. The array is terminated by element with all components set to NULL.

#### 5.1.5.3 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. Always use it instead of glibmm/thread.h and keep among first includes. It safe to use it multiple times and to include it both from source files and other include files. Defines size of stack assigned to every new thread.

## 5.2 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** }

### 5.2.1 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.

### 5.2.2 Enumeration Type Documentation

#### 5.2.2.1 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 6

# Data Structure Documentation

## 6.1 ArcCredential::ACACI Struct Reference

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

- VOMSAttribute.h

## 6.2 ArcCredential::ACATTHOLDER Struct Reference

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

- VOMSAttribute.h

## 6.3 ArcCredential::ACATTR Struct Reference

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

- VOMSAttribute.h

## 6.4 ArcCredential::ACATTRIBUTE Struct Reference

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

- VOMSAttribute.h

## 6.5 ArcCredential::ACC Struct Reference

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

- VOMSAttribute.h

## 6.6 ArcCredential::ACCERTS Struct Reference

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

- VOMSAttribute.h

## 6.7 ArcCredential::ACDIGEST Struct Reference

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

- VOMSAttribute.h

## 6.8 ArcCredential::ACFORM Struct Reference

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

- VOMSAttribute.h

## 6.9 ArcCredential::ACFULLATTRIBUTES Struct Reference

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

- VOMSAttribute.h

## 6.10 ArcCredential::ACHOLDER Struct Reference

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

- VOMSAttribute.h

## 6.11 ArcCredential::ACIETFATTR Struct Reference

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

- VOMSAttribute.h

## 6.12 ArcCredential::ACINFO Struct Reference

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

- VOMSAttribute.h

## 6.13 ArcCredential::ACIS Struct Reference

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

- VOMSAttribute.h

## 6.14 ArcCredential::ACSEQ Struct Reference

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

- VOMSAttribute.h

## 6.15 ArcCredential::ACTARGET Struct Reference

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

- VOMSAttribute.h

## 6.16 ArcCredential::ACTARGETS Struct Reference

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

- VOMSAttribute.h

## 6.17 ArcCredential::ACVAL Struct Reference

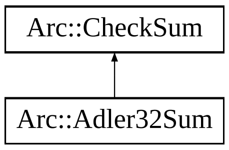
The documentation for this struct was generated from the following file:

- VOMSAttribute.h

## 6.18 Arc::Adler32Sum Class Reference

Implementation of Adler32 checksum.

`#include <CheckSum.h>`Inheritance diagram for Arc::Adler32Sum::



### 6.18.1 Detailed Description

Implementation of Adler32 checksum.

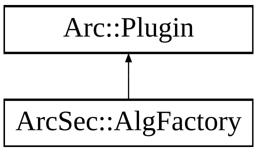
The documentation for this class was generated from the following file:

- CheckSum.h

## 6.19 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

### 6.19.1 Detailed Description

Interface for algorithm factory class. **AlgFactory** (p. 63) is in charge of creating **CombiningAlg** (p. 111) 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.

### 6.19.2 Member Function Documentation

#### 6.19.2.1 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:**

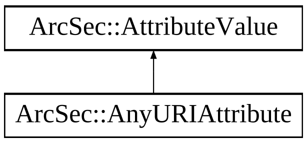
The object of **CombiningAlg** (p. 111)

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

- AlgFactory.h

## 6.20 ArcSec::AnyURIAttribute Class Reference

Inheritance diagram for ArcSec::AnyURIAttribute::



### Public Member Functions

- virtual std::string **encode** ()
- std::string **getId** ()
- virtual std::string **getType** ()

### 6.20.1 Member Function Documentation

#### 6.20.1.1 virtual std::string ArcSec::AnyURIAttribute::encode () `[inline, virtual]`

encode the value in a string format

Implements **ArcSec::AttributeValue** (p. 78).

#### 6.20.1.2 std::string ArcSec::AnyURIAttribute::getId () `[inline, virtual]`

Get the AttributeId of the <Attribute>

Implements **ArcSec::AttributeValue** (p. 78).

#### 6.20.1.3 virtual std::string ArcSec::AnyURIAttribute::getType () `[inline, virtual]`

Get the DataType of the <Attribute>

Implements **ArcSec::AttributeValue** (p. 78).

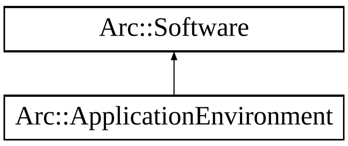
The documentation for this class was generated from the following file:

- AnyURIAttribute.h

## 6.21 Arc::ApplicationEnvironment Class Reference

**ApplicationEnvironment** (p. 65).

`#include <ExecutionTarget.h>`Inheritance diagram for Arc::ApplicationEnvironment::



### 6.21.1 Detailed Description

**ApplicationEnvironment** (p. 65). The ApplicationEnviroment is closely related to the definition given in GLUE2. By extending the **Software** (p. 417) class the two GLUE2 attributes AppName and AppVersion are mapped to two private members. However these can be obtained through the inheriated member methods getName and getVersion.

GLUE2 description: A description of installed application software or software environment characteristics available within one or more Execution Environments.

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

- ExecutionTarget.h

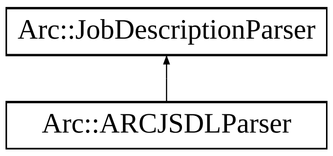
## 6.22 Arc::ApplicationType Class Reference

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

- JobDescription.h

## 6.23 Arc::ARCJSDLParser Class Reference

Inheritance diagram for Arc::ARCJSDLParser::



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

- ARCJSDLParser.h

## 6.24 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** ()

### 6.24.1 Detailed Description

Determines ARC installation location.

### 6.24.2 Member Function Documentation

#### 6.24.2.1 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.

#### 6.24.2.2 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

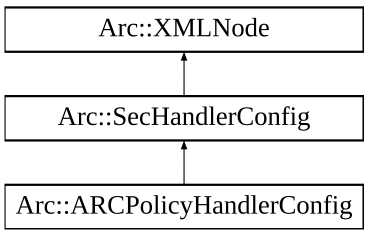
## 6.25 ArcSec::ArcPeriod Struct Reference

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

- DateTimeAttribute.h

## 6.26 Arc::ARCPolicyHandlerConfig Class Reference

Inheritance diagram for Arc::ARCPolicyHandlerConfig::



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

- ClientInterface.h

## 6.27 ArcSec::Attr Struct Reference

**Attr** (p. 71) contains a tuple of attribute type and value.

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

### 6.27.1 Detailed Description

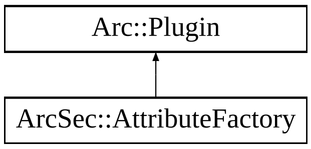
**Attr** (p. 71) contains a tuple of attribute type and value.

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

- Request.h

## 6.28 ArcSec::AttributeFactory Class Reference

`#include <AttributeFactory.h>`Inheritance diagram for ArcSec::AttributeFactory::



### 6.28.1 Detailed Description

Base attribute factory class

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

- AttributeFactory.h

## 6.29 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**

### 6.29.1 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. 293) class returns an **AttributeIterator** (p. 73) 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;
```

### 6.29.2 Constructor & Destructor Documentation

#### 6.29.2.1 Arc::AttributeIterator::AttributeIterator ()

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

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

Protected constructor used by the **MessageAttributes** (p. 293) 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. 293) class.

**Parameters:**

***begin*** A const_iterator pointing to the first matching key-value pair in the internal multimap of the **MessageAttributes** (p. 293) class.

***end*** A const_iterator pointing to the first key-value pair in the internal multimap of the **MessageAttributes** (p. 293) class where the key is larger than the key searched for.

### 6.29.3 Member Function Documentation

#### 6.29.3.1 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.

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

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

#### 6.29.3.3 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.

#### 6.29.3.4 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.

#### 6.29.3.5 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.

#### 6.29.3.6 const std::string∗ Arc::AttributeIterator::operator-> () const

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

### 6.29.4 Friends And Related Function Documentation

#### 6.29.4.1 friend class MessageAttributes `[friend]`

The **MessageAttributes** (p. 293) class is a friend. The constructor that creates an **AttributeIterator** (p. 73) that is connected to the internal multimap of the **MessageAttributes** (p. 293) class should not be exposed to the outside, but it still needs to be accessible from the getAll() method of the **MessageAttributes** (p. 293) class. Therefore, that class is a friend.

### 6.29.5 Field Documentation

#### 6.29.5.1 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. 293) class.

#### 6.29.5.2 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. 293) class where the key is larger than the key searched for.

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

- MessageAttributes.h

## 6.30 ArcSec::AttributeProxy Class Reference

Interface for creating the **AttributeValue** (p. 77) object, it will be used by **AttributeFactory** (p. 72).

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

### Public Member Functions

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

### 6.30.1 Detailed Description

Interface for creating the **AttributeValue** (p. 77) object, it will be used by **AttributeFactory** (p. 72). The **AttributeProxy** (p. 76) 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. 77)

### 6.30.2 Member Function Documentation

#### 6.30.2.1 virtual AttributeValue∗ ArcSec::AttributeProxy::getAttribute (const Arc::XMLNode & *node*) `[pure virtual]`

Create a **AttributeValue** (p. 77) object according to the information inside the XMLNode as parameter.

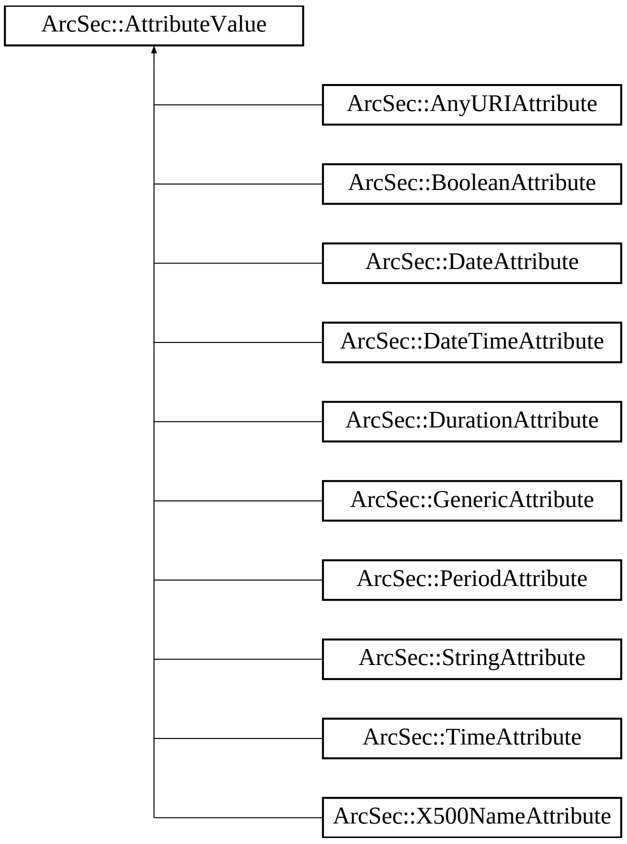
The documentation for this class was generated from the following file:

- AttributeProxy.h

## 6.31 ArcSec::AttributeValue Class Reference

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

`#include <AttributeValue.h>`Inheritance diagram for ArcSec::AttributeValue::



### Public Member Functions

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

### 6.31.1 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** (p. 435), **DateAttribute** (p. 177), **TimeAttribute** (p. 455), **DurationAttribute** (p. 195), **PeriodAttribute** (p. 337), **AnyURIAttribute** (p. 64), **X500NameAttribute** (p. 542)

## 6.31.2 Member Function Documentation

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

encode the value in a string format

Implemented in **ArcSec::AnyURIAttribute** (p. 64), **ArcSec::BooleanAttribute** (p. 86), **ArcSec::DateTimeAttribute** (p. 178), **ArcSec::TimeAttribute** (p. 455), **ArcSec::DateAttribute** (p. 177), **ArcSec::DurationAttribute** (p. 195), **ArcSec::PeriodAttribute** (p. 337), **ArcSec::GenericAttribute** (p. 223), **ArcSec::StringAttribute** (p. 435), and **ArcSec::X500NameAttribute** (p. 542).

### 6.31.2.2 virtual bool ArcSec::AttributeValue::equal (AttributeValue ∗ *value*, bool *check_id* = `true`) `[pure virtual]`

Evluate whether "this" equale to the parameter value

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

Get the AttributeId of the <Attribute>

Implemented in **ArcSec::AnyURIAttribute** (p. 64), **ArcSec::BooleanAttribute** (p. 86), **ArcSec::DateTimeAttribute** (p. 178), **ArcSec::TimeAttribute** (p. 455), **ArcSec::DateAttribute** (p. 177), **ArcSec::DurationAttribute** (p. 195), **ArcSec::PeriodAttribute** (p. 337), **ArcSec::GenericAttribute** (p. 223), **ArcSec::StringAttribute** (p. 435), and **ArcSec::X500NameAttribute** (p. 542).

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

Get the DataType of the <Attribute>

Implemented in **ArcSec::AnyURIAttribute** (p. 64), **ArcSec::BooleanAttribute** (p. 86), **ArcSec::DateTimeAttribute** (p. 178), **ArcSec::TimeAttribute** (p. 455), **ArcSec::DateAttribute** (p. 177), **ArcSec::DurationAttribute** (p. 195), **ArcSec::PeriodAttribute** (p. 337), **ArcSec::GenericAttribute** (p. 223), **ArcSec::StringAttribute** (p. 435), and **ArcSec::X500NameAttribute** (p. 542).

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

- AttributeValue.h

## 6.32 ArcSec::Attrs Class Reference

**Attrs** (p. 79) is a container for one or more **Attr** (p. 71).

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

### 6.32.1 Detailed Description

**Attrs** (p. 79) is a container for one or more **Attr** (p. 71). **Attrs** (p. 79) 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

## 6.33 ArcSec::AuthzRequest Struct Reference

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

- PDP.h

## 6.34 ArcSec::AuthzRequestSection Struct Reference

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

### 6.34.1 Detailed Description

These structure are based on the request schema for **PDP** (p. 332), 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

## 6.35 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

### 6.35.1 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.

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

- Utils.h

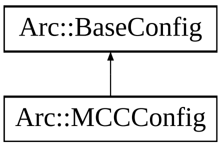
## 6.36 Arc::Base64 Class Reference

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

- Base64.h

## 6.37 Arc::BaseConfig Class Reference

`#include <ArcConfig.h>`Inheritance diagram for Arc::BaseConfig::



### 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

### 6.37.1 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.

### 6.37.2 Member Function Documentation

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

Add CA directory

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

Add CA file

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

Add certificate

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

Add configuration overlay

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

Adds non-standard location of plugins

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

Add private key

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

Add credentials proxy

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

Read overlay from file

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

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

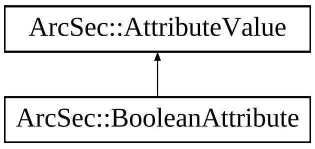
Reimplemented in **Arc::MCCConfig** (p. 283).

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

- ArcConfig.h

## 6.38 ArcSec::BooleanAttribute Class Reference

Inheritance diagram for ArcSec::BooleanAttribute::



### Public Member Functions

- virtual std::string **encode** ()
- std::string **getId** ()
- std::string **getType** ()

### 6.38.1 Member Function Documentation

#### 6.38.1.1 virtual std::string ArcSec::BooleanAttribute::encode () `[inline, virtual]`

encode the value in a string format

Implements **ArcSec::AttributeValue** (p. 78).

#### 6.38.1.2 std::string ArcSec::BooleanAttribute::getId () `[inline, virtual]`

Get the AttributeId of the <Attribute>

Implements **ArcSec::AttributeValue** (p. 78).

#### 6.38.1.3 std::string ArcSec::BooleanAttribute::getType () `[inline, virtual]`

Get the DataType of the <Attribute>

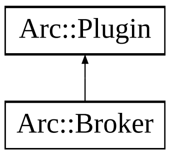
Implements **ArcSec::AttributeValue** (p. 78).

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

- BooleanAttribute.h

## 6.39 Arc::Broker Class Reference

Inheritance diagram for Arc::Broker::



### Public Member Functions

- const **ExecutionTarget** ∗ **GetBestTarget** ()
- void **PreFilterTargets** (std::list< **ExecutionTarget** > &targets, const **JobDescription** &job)
- void **RegisterJobsubmission** ()

### Protected Member Functions

- virtual void **SortTargets** ()=0

### Protected Attributes

- std::list< **ExecutionTarget** ∗ > **PossibleTargets**
- bool **TargetSortingDone**

### 6.39.1 Member Function Documentation

#### 6.39.1.1 const ExecutionTarget∗ Arc::Broker::GetBestTarget ()

Returns next target from the list of **ExecutionTarget** (p. 207) objects. When first called this method will sort its list of **ExecutionTarget** (p. 207) objects, which have been filled by the PreFilterTargets method, and then the first target in the list will be returned.

If this is not the first call then the next target in the list is simply returned.

If there are no targets in the list or the end of the target list have been reached the NULL pointer is returned.

**Returns:**

The pointer to the next **ExecutionTarget** (p. 207) in the list is returned.

#### 6.39.1.2 void Arc::Broker::PreFilterTargets (std::list< ExecutionTarget > & *targets*, const JobDescription & *job*)

**ExecutionTarget** (p. 207) filtering, view-point: enought memory, diskspace, CPUs, etc. The "bad" targets will be ignored and only the good targets will be added to to the list of **ExecutionTarget** (p. 207) objects which be used for brokering.

**Parameters:**

***targets*** A list of **ExecutionTarget** (p. 207) objects to be considered for addition to the **Broker** (p. 87).

***jd*** **JobDescription** (p. 257) object of the actual job.

#### 6.39.1.3 virtual void Arc::Broker::SortTargets () `[protected, pure virtual]`

Custom Brokers should implement this method. The task is to sort the PossibleTargets list by "custom" way, for example: FastestQueueBroker, **ExecutionTarget** (p. 207) which has the shortest queue lenght will be at the begining of the PossibleTargets list

## 6.39.2 Field Documentation

#### 6.39.2.1 std::list<ExecutionTarget∗> Arc::Broker::PossibleTargets `[protected]`

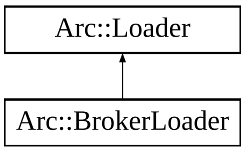
This content the Prefilteres ExecutionTargets. If an Execution Tartget has enought memory, CPU, diskspace, etc. for the actual job requirement than it will be added to the PossibleTargets list

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

- Broker.h

## 6.40 Arc::BrokerLoader Class Reference

`#include <Broker.h>`Inheritance diagram for Arc::BrokerLoader::



### Public Member Functions

- **BrokerLoader** ()
- ∼**BrokerLoader** ()
- **Broker** ∗ **load** (const std::string &name, const **UserConfig** &usercfg)
- const std::list< **Broker** ∗ > & **GetBrokers** () const

### 6.40.1 Detailed Description

Class responsible for loading **Broker** (p. 87) plugins The **Broker** (p. 87) objects returned by a **Broker-Loader** (p. 89) must not be used after the **BrokerLoader** (p. 89) goes out of scope.

### 6.40.2 Constructor & Destructor Documentation

#### 6.40.2.1 Arc::BrokerLoader::BrokerLoader ()

Constructor Creates a new **BrokerLoader** (p. 89).

#### 6.40.2.2 Arc::BrokerLoader::∼BrokerLoader ()

Destructor Calling the destructor destroys all Brokers loaded by the **BrokerLoader** (p. 89) instance.

### 6.40.3 Member Function Documentation

#### 6.40.3.1 const std::list<Broker∗>& Arc::BrokerLoader::GetBrokers () const `[inline]`

Retrieve the list of loaded Brokers.

**Returns:**

A reference to the list of Brokers.

#### 6.40.3.2 Broker∗ Arc::BrokerLoader::load (const std::string & *name*, const UserConfig & *usercfg*)

Load a new **Broker** (p. 87)

**Parameters:**

***name*** The name of the **Broker** (p. 87) to load.

***usercfg*** The **UserConfig** (p. 468) object for the new **Broker** (p. 87).

**Returns:**

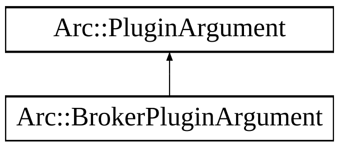
A pointer to the new **Broker** (p. 87) (NULL on error).

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

- Broker.h

## 6.41 Arc::BrokerPluginArgument Class Reference

Inheritance diagram for Arc::BrokerPluginArgument::



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

- Broker.h

## 6.42 Arc::ByteArray Class Reference

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

- ByteArray.h

## 6.43 Arc::CacheParameters Struct Reference

```
#include <FileCache.h>
```

### 6.43.1 Detailed Description

Contains data on the parameters of a cache.

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

- FileCache.h

## 6.44 ArcCredential::cert_verify_context Struct Reference

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

- CertUtil.h

## 6.45 Arc::ChainContext Class Reference

Interface to chain specific functionality.

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

### Public Member Functions

- **operator PluginsFactory** $*$ ()

### 6.45.1 Detailed Description

Interface to chain specific functionality. Object of this class is associated with every **MCCLoader** (p. 285) object. It is accessible for **MCC** (p. 278) and **Service** (p. 409) components and provides an interface to manipulate chains stored in **Loader** (p. 264). This makes it possible to modify chains dynamically - like deploying new services on demand.

### 6.45.2 Member Function Documentation

#### 6.45.2.1 Arc::ChainContext::operator PluginsFactory $*$ () `[inline]`

Returns associated **PluginsFactory** (p. 347) object

References Arc::Loader::factory_.

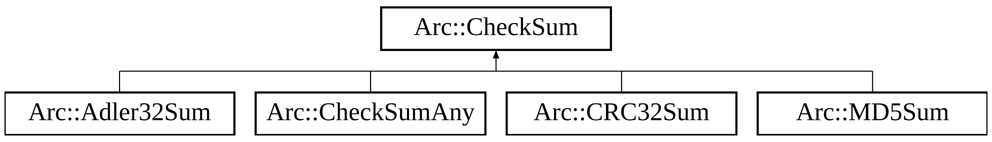
The documentation for this class was generated from the following file:

- MCCLoader.h

## 6.46 Arc::CheckSum Class Reference

Defines interface for variuos checksum manipulations.

`#include <CheckSum.h>`Inheritance diagram for Arc::CheckSum::



### 6.46.1 Detailed Description

Defines interface for variuos checksum manipulations. This class is used during data transfers through **DataBuffer** (p. 140) class

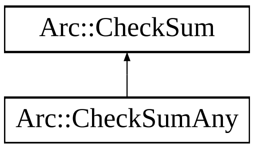
The documentation for this class was generated from the following file:

- CheckSum.h

## 6.47 Arc::CheckSumAny Class Reference

Wraper for **CheckSum** (p. 96) class.

`#include <CheckSum.h>`Inheritance diagram for Arc::CheckSumAny::



### 6.47.1 Detailed Description

Wraper for **CheckSum** (p. 96) class. To be used for manipulation of any supported checksum type in a transparent way.

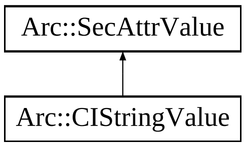
The documentation for this class was generated from the following file:

- CheckSum.h

## 6.48 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)

### 6.48.1 Detailed Description

This class implements case insensitive strings as security attributes. This is an example of how to inherit **SecAttrValue** (p. 403). The class is meant to implement security attributes that are case insensitive strings.

### 6.48.2 Constructor & Destructor Documentation

#### 6.48.2.1 Arc::CIStringValue::CIStringValue ()

Default constructor

#### 6.48.2.2 Arc::CIStringValue::CIStringValue (const char ∗ *ss*)

This is a constructor that takes a string litteral.

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

This is a constructor that takes a string object.

### 6.48.3 Member Function Documentation

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

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

#### 6.48.3.2 virtual Arc::CIStringValue::operator bool () `[virtual]`

This function returns false if the string is empty or uninitialized

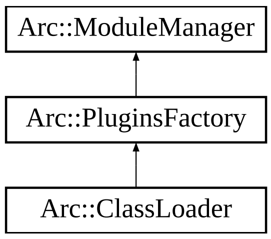
Reimplemented from **Arc::SecAttrValue** (p. 403).

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

- CIStringValue.h

## 6.49 Arc::ClassLoader Class Reference

Inheritance diagram for Arc::ClassLoader::

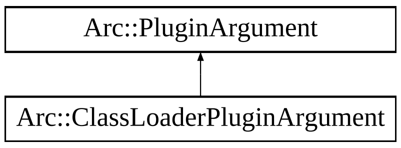


The documentation for this class was generated from the following file:

- ClassLoader.h

## 6.50 Arc::ClassLoaderPluginArgument Class Reference

Inheritance diagram for Arc::ClassLoaderPluginArgument::



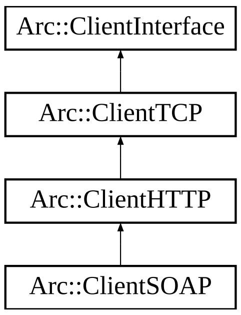
The documentation for this class was generated from the following file:

- ClassLoader.h

## 6.51 Arc::ClientHTTP Class Reference

Class for setting up a **MCC** (p. 278) chain for HTTP communication.

`#include <ClientInterface.h>`Inheritance diagram for Arc::ClientHTTP::



### 6.51.1 Detailed Description

Class for setting up a **MCC** (p. 278) chain for HTTP communication. The **ClientHTTP** (p. 102) class inherits from the **ClientTCP** (p. 108) class and adds an HTTP **MCC** (p. 278) to the chain.

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

- ClientInterface.h

## 6.52 Arc::ClientHTTPwithSAML2SSO Class Reference

### Public Member Functions

- **ClientHTTPwithSAML2SSO** ()
- **MCC_Status process** (const std::string &method, **PayloadRawInterface** ∗request, **HTTPClientInfo** ∗info, **PayloadRawInterface** ∗∗response, const std::string &idp_name, const std::string &username, const std::string &password, const bool reuse_authn=false)

### 6.52.1 Constructor & Destructor Documentation

#### 6.52.1.1 Arc::ClientHTTPwithSAML2SSO::ClientHTTPwithSAML2SSO () `[inline]`

Constructor creates **MCC** (p. 278) chain and connects to server.

### 6.52.2 Member Function Documentation

#### 6.52.2.1 MCC_Status Arc::ClientHTTPwithSAML2SSO::process (const std::string & *method*, PayloadRawInterface ∗ *request*, HTTPClientInfo ∗ *info*, PayloadRawInterface ∗∗ *response*, const std::string & *idp_name*, const std::string & *username*, const std::string & *password*, const bool *reuse_authn* = `false`)

Send HTTP request and receive response.

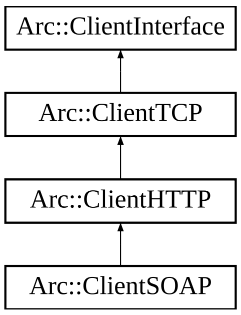
The documentation for this class was generated from the following file:

- ClientSAML2SSO.h

## 6.53 Arc::ClientInterface Class Reference

Utility base class for **MCC** (p. 278).

`#include <ClientInterface.h>`Inheritance diagram for Arc::ClientInterface::



### 6.53.1 Detailed Description

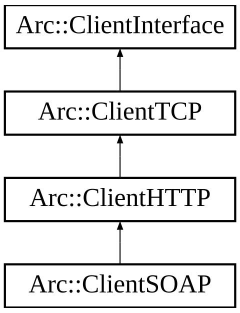
Utility base class for **MCC** (p. 278). The **ClientInterface** (p. 104) class is a utility base class used for configuring a client side **Message** (p. 290) Chain Component (**MCC** (p. 278)) chain and loading it into memory. It has several specializations of increasing complexity of the **MCC** (p. 278) chains.

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

- ClientInterface.h

## 6.54 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** ()

### 6.54.1 Detailed Description

Class with easy interface for sending/receiving SOAP messages over HTTP(S/G). It takes care of configuring **MCC** (p. 278) chain and making an entry point.

### 6.54.2 Constructor & Destructor Documentation

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

Constructor creates **MCC** (p. 278) chain and connects to server.

### 6.54.3 Member Function Documentation

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

Adds security handler to configuration of SOAP **MCC** (p. 278)

Reimplemented from **Arc::ClientHTTP** (p. 102).

#### 6.54.3.2 MCC∗ Arc::ClientSOAP::GetEntry () `[inline]`

Returns entry point to SOAP **MCC** (p. 278) in configured chain. To initialize entry point **Load()** (p. 106) method must be called.

Reimplemented from **Arc::ClientHTTP** (p. 102).

#### 6.54.3.3 virtual void Arc::ClientSOAP::Load () `[virtual]`

Instantiates pluggable elements according to generated configuration

Reimplemented from **Arc::ClientHTTP** (p. 102).

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

Send SOAP request with specified SOAP action and receive response.

#### 6.54.3.5 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

## 6.55 Arc::ClientSOAPwithSAML2SSO Class Reference

### Public Member Functions

- **ClientSOAPwithSAML2SSO** ()
- **MCC_Status process** (**PayloadSOAP** ∗request, **PayloadSOAP** ∗∗response, const std::string &idp_name, const std::string &username, const std::string &password, const bool reuse_-authn=false)
- **MCC_Status process** (const std::string &action, **PayloadSOAP** ∗request, **PayloadSOAP** ∗∗response, const std::string &idp_name, const std::string &username, const std::string &password, const bool reuse_authn=false)

### 6.55.1 Constructor & Destructor Documentation

#### 6.55.1.1 Arc::ClientSOAPwithSAML2SSO::ClientSOAPwithSAML2SSO () `[inline]`

Constructor creates **MCC** (p. 278) chain and connects to server.

### 6.55.2 Member Function Documentation

#### 6.55.2.1 MCC_Status Arc::ClientSOAPwithSAML2SSO::process (const std::string & *action*, PayloadSOAP ∗ *request*, PayloadSOAP ∗∗ *response*, const std::string & *idp_name*, const std::string & *username*, const std::string & *password*, const bool *reuse_authn* = `false`)

Send SOAP request with specified SOAP action and receive response.

#### 6.55.2.2 MCC_Status Arc::ClientSOAPwithSAML2SSO::process (PayloadSOAP ∗ *request*, PayloadSOAP ∗∗ *response*, const std::string & *idp_name*, const std::string & *username*, const std::string & *password*, const bool *reuse_authn* = `false`)

Send SOAP request and receive response.

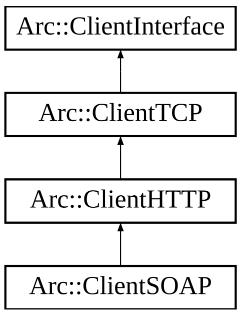
The documentation for this class was generated from the following file:

- ClientSAML2SSO.h

## 6.56 Arc::ClientTCP Class Reference

Class for setting up a **MCC** (p. 278) chain for TCP communication.

`#include <ClientInterface.h>`Inheritance diagram for Arc::ClientTCP::



### 6.56.1 Detailed Description

Class for setting up a **MCC** (p. 278) chain for TCP communication. The **ClientTCP** (p. 108) class is a specialization of the **ClientInterface** (p. 104) which sets up a client **MCC** (p. 278) chain for TCP communication, and optionally with a security layer on top which can be either TLS, GSI or SSL3.

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

- ClientInterface.h

## 6.57 Arc::ClientX509Delegation Class Reference

### Public Member Functions

- **ClientX509Delegation** ()
- bool **createDelegation** (DelegationType deleg, std::string &delegation_id)
- bool **acquireDelegation** (DelegationType deleg, std::string &delegation_cred, std::string &delegation_id, const std::string cred_identity="", const std::string cred_delegator_ip="", const std::string username="", const std::string password="")

### 6.57.1 Constructor & Destructor Documentation

#### 6.57.1.1 Arc::ClientX509Delegation::ClientX509Delegation () `[inline]`

Constructor creates **MCC** (p. 278) chain and connects to server.

### 6.57.2 Member Function Documentation

#### 6.57.2.1 bool Arc::ClientX509Delegation::acquireDelegation (DelegationType *deleg*, std::string & *delegation_cred*, std::string & *delegation_id*, const std::string *cred_identity* = `""`, const std::string *cred_delegator_ip* = `""`, const std::string *username* = `""`, const std::string *password* = `""`)

Acquire delegation credential from delegation service. This method should be called by intermediate service ('n+1' service as explained on above) in order to use this delegation credential on behalf of the EEC's holder.

**Parameters:**

***deleg*** Delegation type

***delegation_id*** delegation ID which is used to look up the credential by delegation service

***cred_identity*** the identity (in case of x509 credential, it is the DN of EEC credential).

***cred_delegator_ip*** the IP address of the credential delegator. Regard of delegation, an intermediate service should accomplish three tasks: 1. Acquire 'n' level delegation credential (which is delegated by 'n-1' level delegator) from delegation service; 1. Create 'n+1' level delegation credential to delegation service; 2. Use 'n' level delegation credential to act on behalf of the EEC's holder. In case of absense of delegation_id, the 'n-1' level delegator's IP address and credential's identity are supposed to be used for look up the delegation credential from delegation service.

#### 6.57.2.2 bool Arc::ClientX509Delegation::createDelegation (DelegationType *deleg*, std::string & *delegation_id*)

Create the delegation credential according to the different remote delegation service. This method should be called by holder of EEC(end entity credential) which would delegate its EEC credential, or by holder of delegated credential(normally, the holder is intermediate service) which would further delegate the credential (on behalf of the original EEC's holder) (for instance, the 'n' intermediate service creates a delegation credential, then the 'n+1' intermediate service aquires this delegation credential from the delegation service and also acts on behalf of the EEC's holder by using this delegation credential).

**Parameters:**

***deleg*** Delegation type

***delegation_id*** For gridsite delegation service, the delegation_id is supposed to be created by client side, and sent to service side; for ARC delegation service, the delegation_id is supposed to be created by service side, and returned back. So for gridsite delegation service, this parameter is treated as input, while for ARC delegation service, it is treated as output.

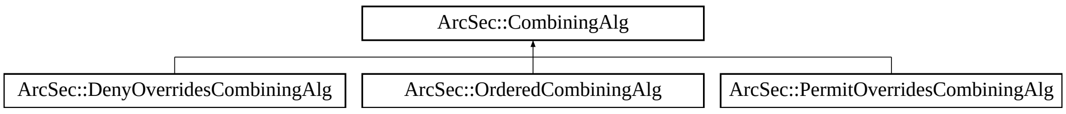
The documentation for this class was generated from the following file:

- ClientX509Delegation.h

## 6.58 ArcSec::CombiningAlg Class Reference

Interface for combining algrithm.

`#include <CombiningAlg.h>`Inheritance diagram for ArcSec::CombiningAlg::


ArcSec::CombiningAlg
ArcSec::DenyOverridesCombiningAlg
ArcSec::OrderedCombiningAlg
ArcSec::PermitOverridesCombiningAlg

### Public Member Functions

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

### 6.58.1 Detailed Description

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

### 6.58.2 Member Function Documentation

#### 6.58.2.1 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. 189), and **ArcSec::PermitOverridesCombiningAlg** (p. 338).

#### 6.58.2.2 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. 189), and **ArcSec::PermitOverridesCombiningAlg** (p. 338).

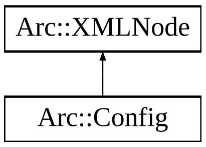
The documentation for this class was generated from the following file:

- CombiningAlg.h

## 6.59 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) const
- void **setFileName** (const std::string &filename)
- void **save** (const char ∗filename)

### 6.59.1 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.

### 6.59.2 Constructor & Destructor Documentation

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

Creates empty XML tree

#### 6.59.2.2 Arc::Config::Config (const char ∗ *filename*)

Loads configuration document from file 'filename'

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

Parse configuration document from memory

#### 6.59.2.4 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.

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

Copy constructor used by language bindings

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

Copy constructor used by language bindings

### 6.59.3 Member Function Documentation

#### 6.59.3.1 const std::string& Arc::Config::getFileName (void) const `[inline]`

Gives back file name of config file or empty string if it was generared from the **XMLNode** (p. 547) subtree

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

Parse configuration document from file 'filename'

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

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

#### 6.59.3.4 void Arc::Config::save (const char ∗ *filename*)

Save to file

#### 6.59.3.5 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

## 6.60 Arc::ConfusaCertHandler Class Reference

`#include <ConfusaCertHandler.h>`

### Public Member Functions

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

### 6.60.1 Detailed Description

Wrapper around **Credential** (p. 129) handling the Confusa specifics.

### 6.60.2 Constructor & Destructor Documentation

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

Create a new **ConfusaCertHandler** (p. 115) for DN dn and given keysize Basically Confusa cert handler wraps around **Credential** (p. 129)

### 6.60.3 Member Function Documentation

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

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

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

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

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

- ConfusaCertHandler.h

## 6.61 Arc::ConfusaParserUtils Class Reference

`#include <ConfusaParserUtils.h>`

### Static Public Member Functions

- static std::string **urlencode** (const std::string url)
- static std::string **urlencode_params** (const std::string url)
- static xmlDocPtr **get_doc** (const std::string xml_file)
- static void **destroy_doc** (xmlDocPtr doc)
- static std::string **extract_body_information** (const std::string html_string)
- static std::string **handle_redirect_step** (**Arc::MCCConfig** cfg, const std::string remote_url, std::string ∗cookies=NULL, std::multimap< std::string, std::string > ∗httpAttributes=NULL)
- static std::string **evaluate_path** (xmlDocPtr doc, const std::string xpathExpr, std::list< std::string > ∗contentList=NULL)

### 6.61.1 Detailed Description

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

### 6.61.2 Member Function Documentation

#### 6.61.2.1 static void Arc::ConfusaParserUtils::destroy_doc (xmlDocPtr *doc*) `[static]`

Destroy a libxml2 doc representation

#### 6.61.2.2 static std::string Arc::ConfusaParserUtils::evaluate_path (xmlDocPtr *doc*, const std::string *xpathExpr*, std::list< std::string > ∗ *contentList* = `NULL`) `[static]`

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

#### 6.61.2.3 static std::string Arc::ConfusaParserUtils::extract_body_information (const std::string *html_string*) `[static]`

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

#### 6.61.2.4 static xmlDocPtr Arc::ConfusaParserUtils::get_doc (const std::string *xml_file*) `[static]`

Construct a lixml2 doc representation from the xml file

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

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

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

urlencode the passed string

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

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

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

- ConfusaParserUtils.h

## 6.62 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

### 6.62.1 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

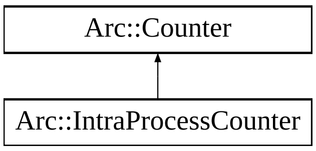
The documentation for this class was generated from the following file:

- Utils.h

## 6.63 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**

### 6.63.1 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(...);
```

## 6.63.2 Member Typedef Documentation

### 6.63.2.1 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. 126) class in order to be able to cancel and extend reservations.

## 6.63.3 Constructor & Destructor Documentation

### 6.63.3.1 Arc::Counter::Counter () `[protected]`

Default constructor. This is the default constructor. Since **Counter** (p. 119) is an abstract class, it should only be used by subclasses. Therefore it is protected. Furthermore, since the **Counter** (p. 119) class has no attributes, nothing needs to be initialized and thus this constructor is empty.

### 6.63.3.2 virtual Arc::Counter::∼Counter () `[virtual]`

The destructor. This is the destructor of the **Counter** (p. 119) class. Since the **Counter** (p. 119) class has no attributes, nothing needs to be cleaned up and thus the destructor is empty.

## 6.63.4 Member Function Documentation

### 6.63.4.1 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. 126) that corresponds to the reservation.

**Parameters:**

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

Implemented in **Arc::IntraProcessCounter** (p. 245).

### 6.63.4.2 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. 245).

#### 6.63.4.3 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. 245).

#### 6.63.4.4 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. 126) 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.

Implemented in **Arc::IntraProcessCounter** (p. 245).

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

A "relay method" for a constructor of the **CounterTicket** (p. 126) class. This method acts as a relay for one of the constructors of the **CounterTicket** (p. 126) class. That constructor is private, but needs to be accessible from the subclasses of **Counter** (p. 119) (bot not from anywhere else). In order not to have to declare every possible subclass of **Counter** (p. 119) as a friend of **CounterTicket** (p. 126), only the base class **Counter** (p. 119) 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. 126).

*expiryTime* the expiry time of the reservation corresponding to the **CounterTicket** (p. 126).

*counter* The **Counter** (p. 119) from which the reservation has been made.

**Returns:**

The counter ticket that has been created.

#### 6.63.4.6 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.

#### 6.63.4.7 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. 246).

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

A "relay method" for the constructor of **ExpirationReminder** (p. 210). This method acts as a relay for one of the constructors of the **ExpirationReminder** (p. 210) class. That constructor is private, but needs to be accessible from the subclasses of **Counter** (p. 119) (bot not from anywhere else). In order not to have to declare every possible subclass of **Counter** (p. 119) as a friend of **ExpirationReminder** (p. 210), only the base class **Counter** (p. 119) 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. 210).

***resID*** The identity number of the reservation corresponding to the **ExpirationReminder** (p. 210).

**Returns:**

The **ExpirationReminder** (p. 210) that has been created.

#### 6.63.4.9 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.

#### 6.63.4.10 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. 246).

#### 6.63.4.11 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. 246).

#### 6.63.4.12 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. 126) that can be queried about the status of the reservation as well as for cancellations and extensions.

Implemented in **Arc::IntraProcessCounter** (p. 246).

#### 6.63.4.13 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. 247).

#### 6.63.4.14 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. 247).

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

- Counter.h

## 6.64 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**

### 6.64.1 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. 119), 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();
```

### 6.64.2 Constructor & Destructor Documentation

#### 6.64.2.1 Arc::CounterTicket::CounterTicket ()

The default constructor. This is the default constructor. It creates a **CounterTicket** (p. 126) 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. 119).

### 6.64.3 Member Function Documentation

#### 6.64.3.1 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.

#### 6.64.3.2 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.

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

Returns the validity of a **CounterTicket** (p. 126). This method checks whether a **CounterTicket** (p. 126) is valid. The ticket was probably returned earlier by the reserve() method of a **Counter** (p. 119) but the corresponding reservation may have expired.

**Returns:**

The validity of the ticket.

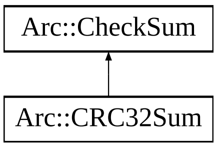
The documentation for this class was generated from the following file:

- Counter.h

## 6.65 Arc::CRC32Sum Class Reference

Implementation of CRC32 checksum.

`#include <CheckSum.h>`Inheritance diagram for Arc::CRC32Sum::



### 6.65.1 Detailed Description

Implementation of CRC32 checksum.

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

- CheckSum.h

## 6.66 Arc::Credential Class Reference

### Public Member Functions

- **Credential** ()
- **Credential** (int keybits)
- **Credential** (const std::string &CAfile, const std::string &CAkey, const std::string &CAserial, bool CAcreateserial, const std::string &extfile, const std::string &extsect, const std::string &passphrase4key="")
- **Credential** (**Time** start, **Period** lifetime=**Period**("PT12H"), int keybits=1024, std::string proxyversion="rfc", std::string policylang="inheritAll", std::string policy="", int pathlength=-1)
- **Credential** (const std::string &cert, const std::string &key, const std::string &cadir, const std::string &cafile, const std::string &passphrase4key="", const bool is_file=true)
- void **AddCertExtObj** (std::string &sn, std::string &oid)
- void **LogError** (void)
- bool **GetVerification** (void)
- EVP_PKEY ∗ **GetPrivKey** (void)
- EVP_PKEY ∗ **GetPubKey** (void)
- X509 ∗ **GetCert** (void)
- X509_REQ ∗ **GetCertReq** (void)
- **STACK_OF** (X509)∗GetCertChain(void)
- int **GetCertNumofChain** (void)
- Credformat **getFormat** (BIO ∗in, const bool is_file=true)
- std::string **GetDN** (void)
- std::string **GetIdentityName** (void)
- **ArcCredential::certType GetType** (void)
- std::string **GetProxyPolicy** (void)
- void **SetProxyPolicy** (const std::string &proxyversion, const std::string &policylang, const std::string &policy, int pathlength)
- bool **OutputPrivatekey** (std::string &content, bool encryption=false, const std::string &passphrase="")
- bool **OutputPublickey** (std::string &content)
- bool **OutputCertificate** (std::string &content, bool is_der=false)
- bool **OutputCertificateChain** (std::string &content, bool is_der=false)
- **Period GetLifeTime** (void)
- **Time GetStartTime** ()
- **Time GetEndTime** ()
- void **SetLifeTime** (const **Period** &period)
- void **SetStartTime** (const **Time** &start_time)
- bool **AddExtension** (std::string name, std::string data, bool crit=false)
- bool **AddExtension** (std::string name, char ∗∗binary, bool crit=false)
- bool **GenerateEECRequest** (BIO ∗reqbio, BIO ∗keybio, std::string dn="")
- bool **GenerateEECRequest** (std::string &reqcontent, std::string &keycontent, std::string dn="")
- bool **GenerateEECRequest** (const char ∗request_filename, const char ∗key_filename, std::string dn="")
- bool **GenerateRequest** (BIO ∗bio, bool if_der=false)
- bool **GenerateRequest** (std::string &content, bool if_der=false)
- bool **GenerateRequest** (const char ∗filename, bool if_der=false)
- bool **InquireRequest** (BIO ∗reqbio, bool if_eec=false, bool if_der=false)
- bool **InquireRequest** (std::string &content, bool if_eec=false, bool if_der=false)

- bool **InquireRequest** (const char ∗filename, bool if_eec=false, bool if_der=false)
- bool **SignRequest** (**Credential** ∗proxy, BIO ∗outputbio, bool if_der=false)
- bool **SignRequest** (**Credential** ∗proxy, std::string &content, bool if_der=false)
- bool **SignRequest** (**Credential** ∗proxy, const char ∗filename, bool foamat=false)
- bool **SignEECRequest** (**Credential** ∗eec, const std::string &DN, BIO ∗outputbio)
- bool **SignEECRequest** (**Credential** ∗eec, const std::string &DN, std::string &content)
- bool **SignEECRequest** (**Credential** ∗eec, const std::string &DN, const char ∗filename)

## Static Public Member Functions

- static void **InitProxyCertInfo** (void)

## 6.66.1 Constructor & Destructor Documentation

### 6.66.1.1 Arc::Credential::Credential ()

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

### 6.66.1.2 Arc::Credential::Credential (int *keybits*)

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

### 6.66.1.3 Arc::Credential::Credential (const std::string & *CAfile*, const std::string & *CAkey*, const std::string & *CAserial*, bool *CAcreateserial*, const std::string & *extfile*, const std::string & *extsect*, const std::string & *passphrase4key* = `""`)

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

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

Constructor, specific constructor for proxy certificate, only acts as a container for constraining certificate signing and/or generating certificate request(only keybits is useful for creating certificate request), is meaningless for any other use. The proxyversion and policylang is for specifying the proxy certificate type and the policy language inside proxy. The definition of proxyversion and policy language is based on `http://dev.globus.org/wiki/Security/ProxyCertTypes#RFC_3820_Proxy_-Certificates` The code is supposed to support proxy version: GSI2(legacy proxy), GSI3(Proxy draft) and RFC(RFC3820 proxy), and correspoding policy language. GSI2(GSI2, GSI2_LIMITED) GSI3 and RFC (IMPERSONATION_PROXY--1.3.6.1.5.5.7.21.1, INDEPENDENT_PROXY--1.3.6.1.5.5.7.21.2, LIMITED_PROXY--1.3.6.1.4.1.3536.1.1.1.9, RESTRICTED_PROXY--policy language undefined) In openssl>=098, there are three types of policy languages: id-ppl-inheritAll--1.3.6.1.5.5.7.21.1, id-ppl-independent--1.3.6.1.5.5.7.21.2, and id-ppl-anyLanguage-1.3.6.1.5.5.7.21.0

**Parameters:**

***start,start*** time of proxy certificate

***lifetime,lifetime*** of proxy certificate

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

#### 6.66.1.5 Arc::Credential::Credential (const std::string & *cert*, const std::string & *key*, const std::string & *cadir*, const std::string & *cafile*, const std::string & *passphrase4key* = `""`, const bool *is_file* = `true`)

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

**Parameters:**

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

### 6.66.2 Member Function Documentation

#### 6.66.2.1 void Arc::Credential::AddCertExtObj (std::string & *sn*, std::string & *oid*)

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

#### 6.66.2.2 bool Arc::Credential::AddExtension (std::string *name*, char ** *binary*, bool *crit* = `false`)

Add an extension to the extension part of the certificate

**Parameters:**

***binary,the*** data which will be inserted into certificate extension part as a specific extension there should be specific methods defined inside specific X509V3_EXT_METHOD structure to parse the specific extension format. For example, VOMS attribute certificate is a specific extension to proxy certificate. There is specific X509V3_EXT_METHOD defined in **VOMSAttribute.h** (p. **??**) and VOMSAttribute.c for parsing attribute certificate. In openssl, the specific X509V3_EXT_METHOD can be got according to the extension name/id, see X509V3_EXT_-get_nid(ext_nid)

#### 6.66.2.3 bool Arc::Credential::AddExtension (std::string *name*, std::string *data*, bool *crit* = `false`)

Add an extension to the extension part of the certificate

**Parameters:**

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

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

### 6.66.2.4 bool Arc::Credential::GenerateEECRequest (const char ∗ *request_filename*, const char ∗ *key_filename*, std::string *dn* = `""`)

Generate an EEC request, output the certificate request and the key to a file

### 6.66.2.5 bool Arc::Credential::GenerateEECRequest (std::string & *reqcontent*, std::string & *keycontent*, std::string *dn* = `""`)

Generate an EEC request, output the certificate request to a string

### 6.66.2.6 bool Arc::Credential::GenerateEECRequest (BIO ∗ *reqbio*, BIO ∗ *keybio*, std::string *dn* = `""`)

Generate an EEC request, based on the keybits and signing algorithm information inside this object output the certificate request to output BIO

The user will be asked for a private key password

### 6.66.2.7 bool Arc::Credential::GenerateRequest (const char ∗ *filename*, bool *if_der* = `false`)

Generate a proxy request, output the certificate request to a file

### 6.66.2.8 bool Arc::Credential::GenerateRequest (std::string & *content*, bool *if_der* = `false`)

Generate a proxy request, output the certificate request to a string

### 6.66.2.9 bool Arc::Credential::GenerateRequest (BIO ∗ *bio*, bool *if_der* = `false`)

Generate a proxy request, base on the keybits and signing algorithm information inside this object output the certificate request to output BIO

### 6.66.2.10 X509∗ Arc::Credential::GetCert (void)

Get the certificate attached to this object

### 6.66.2.11 int Arc::Credential::GetCertNumofChain (void)

Get the number of certificates in the certificate chain attached to this object

### 6.66.2.12 X509_REQ∗ Arc::Credential::GetCertReq (void)

Get the certificate request, if there is any

### 6.66.2.13 std::string Arc::Credential::GetDN (void)

Get the DN of the certificate attached to this object

#### 6.66.2.14 Time Arc::Credential::GetEndTime ()

Returns validity end time of certificate or proxy

#### 6.66.2.15 Credformat Arc::Credential::getFormat (BIO ∗ *in*, const bool *is_file* = `true`)

Get the certificate format, PEM PKCS12 or DER BIO could be memory or file, they should be processed differently.

#### 6.66.2.16 std::string Arc::Credential::GetIdentityName (void)

Get the Identity name of the certificate attached to this object, the result will not include proxy CN

#### 6.66.2.17 Period Arc::Credential::GetLifeTime (void)

Returns lifetime of certificate or proxy

#### 6.66.2.18 EVP_PKEY∗ Arc::Credential::GetPrivKey (void)

Get the private key attached to this object

#### 6.66.2.19 std::string Arc::Credential::GetProxyPolicy (void)

Get the proxy policy attached to the "proxy certificate information" extension of the proxy certicate

#### 6.66.2.20 EVP_PKEY∗ Arc::Credential::GetPubKey (void)

Get the public key attached to this object

#### 6.66.2.21 Time Arc::Credential::GetStartTime ()

Returns validity start time of certificate or proxy

#### 6.66.2.22 ArcCredential::certType Arc::Credential::GetType (void)

Get type of the certificate attached to this object

#### 6.66.2.23 bool Arc::Credential::GetVerification (void) `[inline]`

Get the verification result about certificate chain checking

#### 6.66.2.24 static void Arc::Credential::InitProxyCertInfo (void) `[static]`

Initiate nid for proxy certificate extension

#### 6.66.2.25 bool Arc::Credential::InquireRequest (const char ∗ *filename*, bool *if_eec* = `false`, bool *if_der* = `false`)

Inquire the certificate request from a file

#### 6.66.2.26 bool Arc::Credential::InquireRequest (std::string & *content*, bool *if_eec* = `false`, bool *if_der* = `false`)

Inquire the certificate request from a string

#### 6.66.2.27 bool Arc::Credential::InquireRequest (BIO ∗ *reqbio*, bool *if_eec* = `false`, bool *if_der* = `false`)

Inquire the certificate request from BIO, and put the request information to X509_REQ inside this object, and parse the certificate type from the PROXYCERTINFO of request' extension

**Parameters:**

*if_der* false for PEM; true for DER

#### 6.66.2.28 void Arc::Credential::LogError (void)

Log error information related with openssl

#### 6.66.2.29 bool Arc::Credential::OutputCertificate (std::string & *content*, bool *is_der* = `false`)

Output the certificate into string

**Parameters:**

*is_der* false for PEM, true for DER

#### 6.66.2.30 bool Arc::Credential::OutputCertificateChain (std::string & *content*, bool *is_der* = `false`)

Output the certificate chain into string

**Parameters:**

*is_der* false for PEM, true for DER

#### 6.66.2.31 bool Arc::Credential::OutputPrivatekey (std::string & *content*, bool *encryption* = `false`, const std::string & *passphrase* = `""`)

Output the private key into string

**Parameters:**

***encryption,whether*** encrypt the output private key or not

***passphrase,the*** passphrase to encrypt the output private key

#### 6.66.2.32 bool Arc::Credential::OutputPublickey (std::string & *content*)

Output the public key into string

#### 6.66.2.33 void Arc::Credential::SetLifeTime (const Period & *period*)

Set lifetime of certificate or proxy

#### 6.66.2.34 void Arc::Credential::SetProxyPolicy (const std::string & *proxyversion*, const std::string & *policylang*, const std::string & *policy*, int *pathlength*)

Set the proxy policy attached to the "proxy certificate information" extension of the proxy certicate

#### 6.66.2.35 void Arc::Credential::SetStartTime (const Time & *start_time*)

Set start time of certificate or proxy

#### 6.66.2.36 bool Arc::Credential::SignEECRequest (Credential ∗ *eec*, const std::string & *DN*, const char ∗ *filename*)

Sign request and output the signed certificate to a file

#### 6.66.2.37 bool Arc::Credential::SignEECRequest (Credential ∗ *eec*, const std::string & *DN*, std::string & *content*)

Sign request and output the signed certificate to a string

#### 6.66.2.38 bool Arc::Credential::SignEECRequest (Credential ∗ *eec*, const std::string & *DN*, BIO ∗ *outputbio*)

Sign eec request, and output the signed certificate to output BIO

#### 6.66.2.39 bool Arc::Credential::SignRequest (Credential ∗ *proxy*, const char ∗ *filename*, bool *foamat* = `false`)

Sign request and output the signed certificate to a file

**Parameters:**

*if_der* false for PEM, true for DER

#### 6.66.2.40 bool Arc::Credential::SignRequest (Credential ∗ *proxy*, std::string & *content*, bool *if_der* = `false`)

Sign request and output the signed certificate to a string

**Parameters:**

*if_der* false for PEM, true for DER

#### 6.66.2.41 bool Arc::Credential::SignRequest (Credential ∗ *proxy*, BIO ∗ *outputbio*, bool *if_der* = `false`)

Sign request based on the information inside proxy, and output the signed certificate to output BIO

**Parameters:**

***if_der*** false for PEM, true for DER

#### 6.66.2.42 Arc::Credential::STACK_OF (X509)

Get the certificate chain attached to this object

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

- Credential.h

## 6.67 Arc::CredentialError Class Reference

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

### Public Member Functions

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

### 6.67.1 Detailed Description

This is an exception class that is used to handle runtime errors discovered in the **Credential** (p. 129) class.

### 6.67.2 Constructor & Destructor Documentation

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

This is the constructor of the **CredentialError** (p. 137) class.

**Parameters:**

***what*** An explanation of the error.

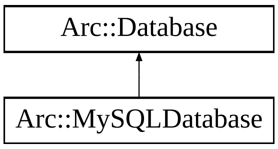
The documentation for this class was generated from the following file:

- Credential.h

## 6.68 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

### 6.68.1 Detailed Description

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

### 6.68.2 Constructor & Destructor Documentation

#### 6.68.2.1 Arc::Database::Database () `[inline]`

Default constructor

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

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

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

Copy constructor

#### 6.68.2.4 virtual Arc::Database::∼Database () `[inline, virtual]`

Deconstructor

### 6.68.3 Member Function Documentation

#### 6.68.3.1 virtual void Arc::Database::close () `[pure virtual]`

Close the connection with database server

Implemented in **Arc::MySQLDatabase** (p. 304).

#### 6.68.3.2 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. 304).

#### 6.68.3.3 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. 304).

#### 6.68.3.4 virtual bool Arc::Database::isconnected () const `[pure virtual]`

Get the connection status

Implemented in **Arc::MySQLDatabase** (p. 305).

#### 6.68.3.5 virtual bool Arc::Database::shutdown () `[pure virtual]`

Ask database server to shutdown

Implemented in **Arc::MySQLDatabase** (p. 305).

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

- DBInterface.h

## 6.69 Arc::DataBuffer Class Reference

Represents set of buffers.

`#include <DataBuffer.h>`

### Data Structures

- struct **buf_desc**
- class **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**

### 6.69.1 Detailed Description

Represents set of buffers. This class is used used during data transfer using **DataPoint** (p. 151) classes.

### 6.69.2 Constructor & Destructor Documentation

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

Contructor

**Parameters:**

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

***blocks*** number of buffers.

#### 6.69.2.2 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. 140) itself.

### 6.69.3 Member Function Documentation

#### 6.69.3.1 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. 140) itself.

**Returns:**

integer position in the list of checksum objects.

#### 6.69.3.2 unsigned int Arc::DataBuffer::buffer_size () const

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

#### 6.69.3.3 const CheckSum∗ Arc::DataBuffer::checksum_object () const

Returns **CheckSum** (p. 96) object specified in constructor, returns NULL if index is not in list.

**Parameters:**

***index*** of the checksum in question.

#### 6.69.3.4 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.

#### 6.69.3.5 bool Arc::DataBuffer::eof_read ()

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

#### 6.69.3.6 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.

#### 6.69.3.7 bool Arc::DataBuffer::eof_write ()

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

#### 6.69.3.8 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.

#### 6.69.3.9 bool Arc::DataBuffer::error ()

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

#### 6.69.3.10 void Arc::DataBuffer::error_read (bool *v*)

Informs object if error accured on 'read' side.

**Parameters:**

***v*** true if error.

#### 6.69.3.11 void Arc::DataBuffer::error_write (bool *v*)

Informs object if error accured on 'write' side.

**Parameters:**

***v*** true if error.

#### 6.69.3.12 bool Arc::DataBuffer::for_read ()

Check if there are buffers which can be taken by **for_read()** (p. 143). This function checks only for buffers and does not take eof and error conditions into account.

#### 6.69.3.13 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

#### 6.69.3.14 bool Arc::DataBuffer::for_write ()

Check if there are buffers which can be taken by **for_write()** (p. 143). This function checks only for buffers and does not take eof and error conditions into account.

#### 6.69.3.15 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.

#### 6.69.3.16 bool Arc::DataBuffer::is_notwritten (char $*$ *buf*)

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

**Parameters:**

***buf*** - address of buffer

#### 6.69.3.17 bool Arc::DataBuffer::is_notwritten (int *handle*)

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

**Parameters:**

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

#### 6.69.3.18 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.

#### 6.69.3.19 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.

#### 6.69.3.20 bool Arc::DataBuffer::is_written (char ∗ *buf*)

Informs object that data was written from buffer.

**Parameters:**

***buf*** - address of buffer

#### 6.69.3.21 bool Arc::DataBuffer::is_written (int *handle*)

Informs object that data was written from buffer.

**Parameters:**

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

#### 6.69.3.22 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. 140) itself.

#### 6.69.3.23 bool Arc::DataBuffer::wait_any ()

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

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

- DataBuffer.h

## 6.70 Arc::DataCallback Class Reference

```
#include <DataCallback.h>
```

### 6.70.1 Detailed Description

This class is used by **DataHandle** (p. 147) 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

## 6.71 Arc::DataHandle Class Reference

This class is a wrapper around the **DataPoint** (p. 151) class.

```
#include <DataHandle.h>
```

### 6.71.1 Detailed Description

This class is a wrapper around the **DataPoint** (p. 151) class. It simplifies the construction, use and destruction of **DataPoint** (p. 151) objects.

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

- DataHandle.h

## 6.72 Arc::DataMover Class Reference

```
#include <DataMover.h>
```

### Public Member Functions

- **DataMover** ()
- ∼**DataMover** ()
- **DataStatus Transfer** (**DataPoint** &source, **DataPoint** &destination, **FileCache** &cache, const **URLMap** &map, 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, 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)

### 6.72.1 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. 149). Instance represents only attributes used during transfer.

### 6.72.2 Member Function Documentation

#### 6.72.2.1 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).

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

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

#### 6.72.2.3 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.

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

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

#### 6.72.2.5 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. 169) class.

#### 6.72.2.6 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. 169) class.

#### 6.72.2.7 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. 169) class.

#### 6.72.2.8 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*, 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.

#### 6.72.2.9 DataStatus Arc::DataMover::Transfer (DataPoint & *source*, DataPoint & *destination*, FileCache & *cache*, const URLMap & *map*, callback *cb* = `NULL`, void ∗ *arg* = `NULL`, const char ∗ *prefix* = `NULL`)

Initiates transfer from 'source' to 'destination'.

**Parameters:**

*source* source **URL** (p. 456).

***destination*** destination **URL** (p. 456).

***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. 456) mapping/convertion table (for 'source' **URL** (p. 456)).

***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.

#### 6.72.2.10 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. 149) is NULL.

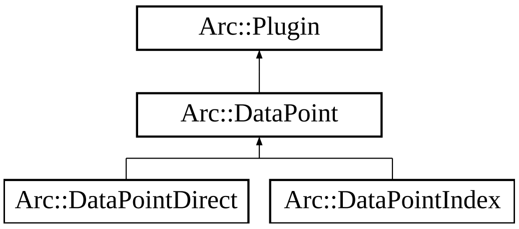
The documentation for this class was generated from the following file:

- DataMover.h

## 6.73 Arc::DataPoint Class Reference

This base class is an abstraction of **URL** (p. 456).

`#include <DataPoint.h>`Inheritance diagram for Arc::DataPoint::



### Public Member Functions

- **DataPoint** (const **URL** &url, const **UserConfig** &usercfg)
- virtual ∼**DataPoint** ()
- virtual const **URL** & **GetURL** () const
- virtual const **UserConfig** & **GetUserConfig** () 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 **DataStatus GetFailureReason** (void) const
- 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 long long int **BufSize** () const =0
- virtual int **BufNum** () const =0
- virtual bool **Cache** () const
- virtual bool **Local** () const =0
- virtual int **GetTries** () const
- virtual void **SetTries** (const int n)
- virtual bool **NextTry** (void)
- virtual bool **IsIndex** () const =0
- virtual bool **AcceptsMeta** ()=0
- virtual bool **ProvidesMeta** ()=0
- virtual void **SetMeta** (const **DataPoint** &p)
- virtual bool **CompareMeta** (const **DataPoint** &p) const
- virtual const **URL** & **CurrentLocation** () const =0
- virtual const std::string & **CurrentLocationMetadata** () const =0
- virtual bool **NextLocation** ()=0
- virtual bool **LocationValid** () const =0
- virtual bool **HaveLocations** () const =0
- virtual **DataStatus** **AddLocation** (const **URL** &url, const std::string &meta)=0
- virtual **DataStatus** **RemoveLocation** ()=0
- virtual **DataStatus** **RemoveLocations** (const **DataPoint** &p)=0

## Protected Attributes

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

## 6.73.1 Detailed Description

This base class is an abstraction of **URL** (p. 456). 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. 151) provides means to resolve an indexing service **URL** (p. 456) into multiple URLs and to loop through them.

## 6.73.2 Constructor & Destructor Documentation

### 6.73.2.1 Arc::DataPoint::DataPoint (const URL & *url*, const UserConfig & *usercfg*)

Constructor requires **URL** (p. 456) to be provided. References to url and usercfg arguments are stored internally and hence corresponding objects must stay available during whole lifetime of this instance.

### 6.73.3 Member Function Documentation

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

Add **URL** (p. 456) to list.

**Parameters:**

*url* Location **URL** (p. 456) to add.

*meta* Location meta information.

Implemented in **Arc::DataPointDirect** (p. 159), and **Arc::DataPointIndex** (p. 163).

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

**Query** (p. 360) the **DataPoint** (p. 151) 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. 163).

#### 6.73.3.3 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.

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

Returns meta information used to create current **URL** (p. 456). Usage differs between different indexing services.

Implemented in **Arc::DataPointDirect** (p. 159), and **Arc::DataPointIndex** (p. 163).

#### 6.73.3.5 virtual DataStatus Arc::DataPoint::GetFailureReason (void) const `[virtual]`

Returns reason of transfer failure, as reported by callbacks. This could be different from the failure returned by the methods themselves.

#### 6.73.3.6 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. 151) 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).

#### 6.73.3.7 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. 159), and **Arc::DataPointIndex** (p. 163).

#### 6.73.3.8 virtual bool Arc::DataPoint::NextTry (void) `[virtual]`

Decrease number of retries left. Returns false if no retries left.

#### 6.73.3.9 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. 159), and **Arc::DataPointIndex** (p. 163).

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

Index **Service** (p. 409) 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. 409) under same name.

Implemented in **Arc::DataPointDirect** (p. 159).

#### 6.73.3.11 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. 409).

Implemented in **Arc::DataPointDirect** (p. 160).

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

Index **Service** (p. 409) 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. 409) under same name.

Implemented in **Arc::DataPointDirect** (p. 160).

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

If endpoint can provide at least some meta information directly.

Implemented in **Arc::DataPointDirect** (p. 160), and **Arc::DataPointIndex** (p. 163).

#### 6.73.3.14 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. 160), and **Arc::DataPointIndex** (p. 164).

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

Allow/disallow **DataPoint** (p. 151) to produce scattered data during reading∗ operation.

**Parameters:**

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

Implemented in **Arc::DataPointDirect** (p. 160), and **Arc::DataPointIndex** (p. 164).

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

Check if file is registered in Indexing **Service** (p. 409). Proper value is obtainable only after Resolve.

Implemented in **Arc::DataPointDirect** (p. 160), and **Arc::DataPointIndex** (p. 164).

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

Resolves index service **URL** (p. 456) into list of ordinary URLs. Also obtains meta information about the file.

**Parameters:**

***source*** true if **DataPoint** (p. 151) object represents source of information.

Implemented in **Arc::DataPointDirect** (p. 161).

#### 6.73.3.18 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. 161), and **Arc::DataPointIndex** (p. 164).

#### 6.73.3.19 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.

#### 6.73.3.20 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. 161), and **Arc::DataPointIndex** (p. 164).

#### 6.73.3.21 virtual DataStatus Arc::DataPoint::StartReading (DataBuffer & *buffer*) `[pure virtual]`

Start reading data from **URL** (p. 456). 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. 164).

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

Start writing data to **URL** (p. 456). 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. 165).

#### 6.73.3.23 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. 165).

#### 6.73.3.24 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. 165).

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

Index **Service** (p. 409) unregistration. Remove information about file registered in Indexing **Service** (p. 409).

**Parameters:**

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

Implemented in **Arc::DataPointDirect** (p. 161).

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

Returns true if **URL** (p. 456) can accept scattered data for ∗writing∗ operation.

Implemented in **Arc::DataPointDirect** (p. 161), and **Arc::DataPointIndex** (p. 165).

### 6.73.4 Field Documentation

#### 6.73.4.1 std::list<std::string> Arc::DataPoint::valid_url_options `[protected]`

Subclasses should add their own specific options to this list

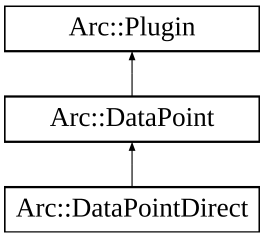
The documentation for this class was generated from the following file:

- DataPoint.h

## 6.74 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 long long int **BufSize** () const
- virtual int **BufNum** () 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** ()

### 6.74.1 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. 140) 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.

## 6.74.2 Member Function Documentation

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

Add **URL** (p. 456) to list.

**Parameters:**

***url*** Location **URL** (p. 456) to add.

***meta*** Location meta information.

Implements **Arc::DataPoint** (p. 153).

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

Returns meta information used to create current **URL** (p. 456). Usage differs between different indexing services.

Implements **Arc::DataPoint** (p. 153).

#### 6.74.2.3 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. 154).

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

Request passive transfers for FTP-like protocols.

**Parameters:**

***true*** to request.

Implements **Arc::DataPoint** (p. 154).

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

Index **Service** (p. 409) 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. 409) under same name.

Implements **Arc::DataPoint** (p. 154).

#### 6.74.2.6 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. 409).

Implements **Arc::DataPoint** (p. 154).

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

Index **Service** (p. 409) 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. 409) under same name.

Implements **Arc::DataPoint** (p. 154).

#### 6.74.2.8 virtual bool Arc::DataPointDirect::ProvidesMeta () `[virtual]`

If endpoint can provide at least some meta information directly.

Implements **Arc::DataPoint** (p. 155).

#### 6.74.2.9 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. 155).

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

Allow/disallow **DataPoint** (p. 151) to produce scattered data during reading∗ operation.

**Parameters:**

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

Implements **Arc::DataPoint** (p. 155).

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

Check if file is registered in Indexing **Service** (p. 409). Proper value is obtainable only after Resolve.

Implements **Arc::DataPoint** (p. 155).

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

Resolves index service **URL** (p. 456) into list of ordinary URLs. Also obtains meta information about the file.

**Parameters:**

***source*** true if **DataPoint** (p. 151) object represents source of information.

Implements **Arc::DataPoint** (p. 155).

#### 6.74.2.13 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. 155).

#### 6.74.2.14 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. 156).

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

Index **Service** (p. 409) unregistration. Remove information about file registered in Indexing **Service** (p. 409).

**Parameters:**

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

Implements **Arc::DataPoint** (p. 157).

#### 6.74.2.16 virtual bool Arc::DataPointDirect::WriteOutOfOrder () `[virtual]`

Returns true if **URL** (p. 456) can accept scattered data for ∗writing∗ operation.

Implements **Arc::DataPoint** (p. 157).

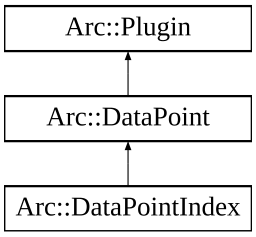
The documentation for this class was generated from the following file:

- DataPointDirect.h

## 6.75 Arc::DataPointIndex Class Reference

Complements **DataPoint** (p. 151) with attributes common for Indexing **Service** (p. 409) 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 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 long long int **BufSize** () const
- virtual int **BufNum** () 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)

### 6.75.1 Detailed Description

Complements **DataPoint** (p. 151) with attributes common for Indexing **Service** (p. 409) URLs. It should never be used directly. Instead inherit from it to provide a class for specific a Indexing **Service** (p. 409).

## 6.75.2 Member Function Documentation

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

Add **URL** (p. 456) to list.

**Parameters:**

***url*** Location **URL** (p. 456) to add.

***meta*** Location meta information.

Implements **Arc::DataPoint** (p. 153).

#### 6.75.2.2 virtual DataStatus Arc::DataPointIndex::Check () `[virtual]`

**Query** (p. 360) the **DataPoint** (p. 151) to check if object is accessible. If possible this method will also try to provide meta information about the object.

Implements **Arc::DataPoint** (p. 153).

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

Returns meta information used to create current **URL** (p. 456). Usage differs between different indexing services.

Implements **Arc::DataPoint** (p. 153).

#### 6.75.2.4 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. 154).

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

Request passive transfers for FTP-like protocols.

**Parameters:**

***true*** to request.

Implements **Arc::DataPoint** (p. 154).

#### 6.75.2.6 virtual bool Arc::DataPointIndex::ProvidesMeta () `[virtual]`

If endpoint can provide at least some meta information directly.

Implements **Arc::DataPoint** (p. 155).

#### 6.75.2.7 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. 155).

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

Allow/disallow **DataPoint** (p. 151) to produce scattered data during reading∗ operation.

**Parameters:**

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

Implements **Arc::DataPoint** (p. 155).

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

Check if file is registered in Indexing **Service** (p. 409). Proper value is obtainable only after Resolve.

Implements **Arc::DataPoint** (p. 155).

#### 6.75.2.10 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. 155).

#### 6.75.2.11 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. 156).

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

Start reading data from **URL** (p. 456). 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. 156).

#### 6.75.2.13 virtual DataStatus Arc::DataPointIndex::StartWriting (DataBuffer & *buffer*, DataCallback ∗ *space_cb* = `NULL`) `[virtual]`

Start writing data to **URL** (p. 456). 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. 156).

#### 6.75.2.14 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. 157).

#### 6.75.2.15 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. 157).

#### 6.75.2.16 virtual bool Arc::DataPointIndex::WriteOutOfOrder () `[virtual]`

Returns true if **URL** (p. 456) can accept scattered data for ∗writing∗ operation.

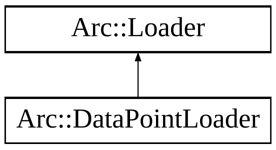
Implements **Arc::DataPoint** (p. 157).

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

- DataPointIndex.h

## 6.76 Arc::DataPointLoader Class Reference

Inheritance diagram for Arc::DataPointLoader::

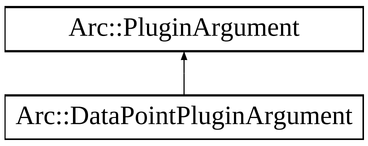


The documentation for this class was generated from the following file:

- DataPoint.h

## 6.77 Arc::DataPointPluginArgument Class Reference

Inheritance diagram for Arc::DataPointPluginArgument::



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

- DataPoint.h

## 6.78 Arc::DataSourceType Class Reference

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

- JobDescription.h

## 6.79 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)

### 6.79.1 Detailed Description

Keeps track of average and instantaneous transfer speed. Also detects data transfer inactivity and other transfer timeouts.

### 6.79.2 Constructor & Destructor Documentation

#### 6.79.2.1 Arc::DataSpeed::DataSpeed (time_t *base* = DATASPEED_AVERAGING_PERIOD)

Constructor

**Parameters:**

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

#### 6.79.2.2 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.

### 6.79.3 Member Function Documentation

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

Turn off speed control.

**Parameters:**

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

#### 6.79.3.2 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).

#### 6.79.3.3 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.

#### 6.79.3.4 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.

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

Set minmal avaerage speed.

**Parameters:**

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

#### 6.79.3.6 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***

#### 6.79.3.7 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.

#### 6.79.3.8 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).

#### 6.79.3.9 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.

#### 6.79.3.10 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

## 6.80 Arc::DataStagingType Class Reference

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

- JobDescription.h

## 6.81 Arc::DataStatus Class Reference

```
#include <DataStatus.h>
```

### Public Types

- enum **DataStatusType** {
  **Success** = 0, **ReadAcquireError** = 1 , **WriteAcquireError** = 2 , **ReadResolveError** = 3 ,
  **WriteResolveError** = 4 , **ReadStartError** = 5 , **WriteStartError** = 6 , **ReadError** = 7 ,
  **WriteError** = 8 , **TransferError** = 9 , **ReadStopError** = 10 , **WriteStopError** = 11 ,
  **PreRegisterError** = 12 , **PostRegisterError** = 13 , **UnregisterError** = 14 , **CacheError** = 15 ,
  **CredentialsExpiredError** = 16, **DeleteError** = 17 , **NoLocationError** = 18, **LocationAlreadyExistsError** = 19,
  **NotSupportedForDirectDataPointsError** = 20, **UnimplementedError** = 21, **IsReadingError** = 22, **IsWritingError** = 23,
  **CheckError** = 24 , **ListError** = 25 , **NotInitializedError** = 26, **SystemError** = 27,
  **StageError** = 28 , **UnknownError** = 29 }

### 6.81.1 Detailed Description

A class to be used for return types of all major data handling methods. It describes the outcome of the method.

### 6.81.2 Member Enumeration Documentation

#### 6.81.2.1 enum Arc::DataStatus::DataStatusType

**Enumerator:**

***Success*** Operation completed successfully.

***ReadAcquireError*** Source is bad **URL** (p. 456) or can't be used due to some reason.

***WriteAcquireError*** Destination is bad **URL** (p. 456) or can't be used due to some reason.

***ReadResolveError*** Resolving of index service **URL** (p. 456) for source failed.

***WriteResolveError*** Resolving of index service **URL** (p. 456) for destination failed.

***ReadStartError*** Can't read from source.

***WriteStartError*** Can't write to destination.

***ReadError*** Failed while reading from source.

***WriteError*** Failed while writing to destination.

***TransferError*** Failed while transfering data (mostly timeout).

***ReadStopError*** Failed while finishing reading from source.

***WriteStopError*** Failed while finishing writing to destination.

***PreRegisterError*** First stage of registration of index service **URL** (p. 456) failed.

***PostRegisterError*** Last stage of registration of index service **URL** (p. 456) failed.

***UnregisterError*** Unregistration of index service **URL** (p. 456) failed.

***CacheError*** Error in caching procedure.

***CredentialsExpiredError*** Error due to provided credentials are expired.

***DeleteError*** Error deleting location or **URL** (p. 456).

***NoLocationError*** No valid location available.

***LocationAlreadyExistsError*** No valid location available.

***NotSupportedForDirectDataPointsError*** Operation has no sense for this kind of **URL** (p. 456).

***UnimplementedError*** Feature is unimplemented.

***IsReadingError*** **DataPoint** (p. 151) is already reading.

***IsWritingError*** **DataPoint** (p. 151) is already writing.

***CheckError*** Access check failed.

***ListError*** File listing failed.

***NotInitializedError*** Object initialization failed.

***SystemError*** Error in OS.

***StageError*** Staging error.

***UnknownError*** Undefined.

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

- DataStatus.h

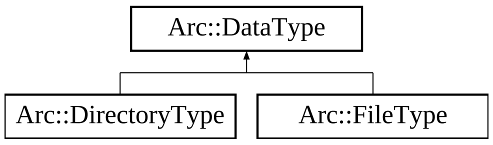
## 6.82 Arc::DataTargetType Class Reference

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

- JobDescription.h

## 6.83 Arc::DataType Class Reference

Inheritance diagram for Arc::DataType::

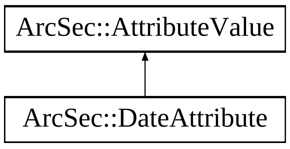


The documentation for this class was generated from the following file:

- JobDescription.h

## 6.84 ArcSec::DateAttribute Class Reference

Inheritance diagram for ArcSec::DateAttribute::



### Public Member Functions

- virtual std::string **encode** ()
- virtual std::string **getType** ()
- virtual std::string **getId** ()

### 6.84.1 Member Function Documentation

#### 6.84.1.1 virtual std::string ArcSec::DateAttribute::encode () `[virtual]`

encode the value in a string format

Implements **ArcSec::AttributeValue** (p. 78).

#### 6.84.1.2 virtual std::string ArcSec::DateAttribute::getId () `[inline, virtual]`

Get the AttributeId of the <Attribute>

Implements **ArcSec::AttributeValue** (p. 78).

#### 6.84.1.3 virtual std::string ArcSec::DateAttribute::getType () `[inline, virtual]`

Get the DataType of the <Attribute>

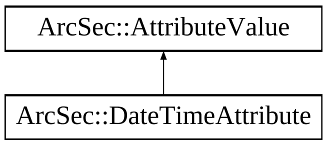
Implements **ArcSec::AttributeValue** (p. 78).

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

- DateTimeAttribute.h

## 6.85 ArcSec::DateTimeAttribute Class Reference

`#include <DateTimeAttribute.h>`Inheritance diagram for ArcSec::DateTimeAttribute::



### Public Member Functions

- virtual std::string **encode** ()
- virtual std::string **getType** ()
- virtual std::string **getId** ()

### 6.85.1 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

### 6.85.2 Member Function Documentation

#### 6.85.2.1 virtual std::string ArcSec::DateTimeAttribute::encode () `[virtual]`

encode the value in a string format

Implements **ArcSec::AttributeValue** (p. 78).

#### 6.85.2.2 virtual std::string ArcSec::DateTimeAttribute::getId () `[inline, virtual]`

Get the AttributeId of the <Attribute>

Implements **ArcSec::AttributeValue** (p. 78).

#### 6.85.2.3 virtual std::string ArcSec::DateTimeAttribute::getType () `[inline, virtual]`

Get the DataType of the <Attribute>

Implements **ArcSec::AttributeValue** (p. 78).

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

- DateTimeAttribute.h

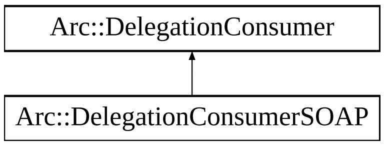
## 6.86 Arc::DBranch Class Reference

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

- DBranch.h

## 6.87 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)

### 6.87.1 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. 181) method for generating certificate request followed by call to **Acquire()** (p. 181) method for making complete credentials from certificate chain.

### 6.87.2 Constructor & Destructor Documentation

#### 6.87.2.1 Arc::DelegationConsumer::DelegationConsumer (void)

Creates object with new private key

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

Creates object with provided private key

### 6.87.3 Member Function Documentation

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

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

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

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

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

Stores content of this object into a string

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

Private key

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

Return identifier of this object - not implemented

#### 6.87.3.6 void Arc::DelegationConsumer::LogError (void) `[protected]`

Creates private key

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

Make X509 certificate request from internal private key

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

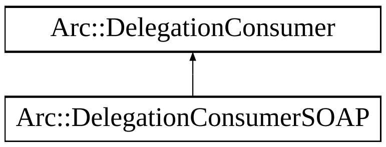
Restores content of object from string

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

- DelegationInterface.h

## 6.88 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)

### 6.88.1 Detailed Description

This class extends **DelegationConsumer** (p. 180) to support SOAP message exchange. Implements WS interface `http://www.nordugrid.org/schemas/delegation` described in delegation.wsdl.

### 6.88.2 Constructor & Destructor Documentation

#### 6.88.2.1 Arc::DelegationConsumerSOAP::DelegationConsumerSOAP (void)

Creates object with new private key

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

Creates object with specified private key

### 6.88.3 Member Function Documentation

#### 6.88.3.1 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.

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

Similar to UpdateCredentials but takes only DelegatedToken XML element

#### 6.88.3.3 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

#### 6.88.3.4 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

## 6.89 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_**

### 6.89.1 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. 182) with identifier included in SOAP message used to route execution to one of managed **DelegationConsumerSOAP** (p. 182) instances.

### 6.89.2 Member Function Documentation

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

See **DelegationConsumerSOAP::DelegateCredentialsInit** (p. 182)

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

See **DelegationConsumerSOAP::DelegatedToken** (p. 183)

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

See **DelegationConsumerSOAP::UpdateCredentials** (p. 183)

### 6.89.3 Field Documentation

#### 6.89.3.1 bool Arc::DelegationContainerSOAP::context_lock_ `[protected]`

If true delegation consumer is deleted when connection context is destroyed

#### 6.89.3.2 int Arc::DelegationContainerSOAP::max_duration_ `[protected]`

Lifetime of unused delegation consumer

#### 6.89.3.3 int Arc::DelegationContainerSOAP::max_size_ `[protected]`

Max. number of delegation consumers

#### 6.89.3.4 int Arc::DelegationContainerSOAP::max_usage_ `[protected]`

Max. times same delegation consumer may accept credentials

#### 6.89.3.5 bool Arc::DelegationContainerSOAP::restricted_ `[protected]`

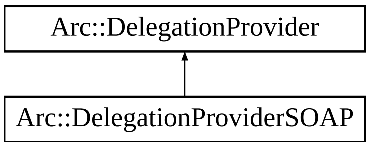
If true all delegation phases must be performed by same identity

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

- DelegationInterface.h

## 6.90 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())

### 6.90.1 Detailed Description

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

### 6.90.2 Constructor & Destructor Documentation

#### 6.90.2.1 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.

#### 6.90.2.2 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.

### 6.90.3 Member Function Documentation

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

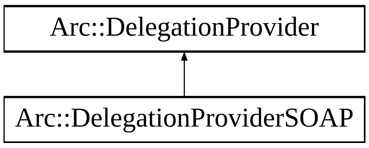
Perform delegation. Takes X509 certificate request and creates proxy credentials excluding private key. Result is then to be fed into **DelegationConsumer::Acquire** (p. 181)

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

- DelegationInterface.h

## 6.91 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)

### 6.91.1 Detailed Description

Extension of **DelegationProvider** (p. 186) with SOAP exchange interface. This class is also a temporary container for intermediate information used during delegation procedure.

### 6.91.2 Constructor & Destructor Documentation

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

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

#### 6.91.2.2 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.

## 6.91.3 Member Function Documentation

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

Extended version of **DelegateCredentialsInit(MCCInterface&,MessageContext∗)** (p. 188). Additionally takes attributes for request and response message to make fine control on message processing possible.

### 6.91.3.2 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.

### 6.91.3.3 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.

### 6.91.3.4 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.

### 6.91.3.5 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.

### 6.91.3.6 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. 182) instance.

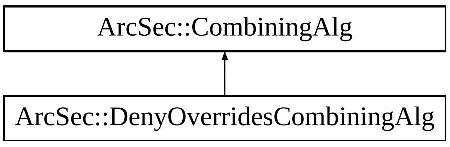
The documentation for this class was generated from the following file:

- DelegationInterface.h

## 6.92 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

### 6.92.1 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".

### 6.92.2 Member Function Documentation

#### 6.92.2.1 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. 111).

#### 6.92.2.2 virtual const std::string& ArcSec::DenyOverridesCombiningAlg::getalgId (void) const `[inline, virtual]`

Get the identifier

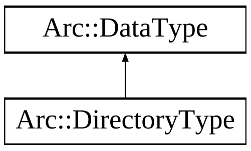
Implements **ArcSec::CombiningAlg** (p. 111).

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

- DenyOverridesAlg.h

## 6.93 Arc::DirectoryType Class Reference

Inheritance diagram for Arc::DirectoryType::



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

- JobDescription.h

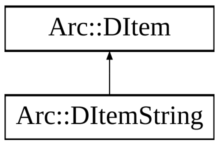
## 6.94 Arc::DiskSpaceRequirementType Class Reference

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

- JobDescription.h

## 6.95 Arc::DItem Class Reference

Inheritance diagram for Arc::DItem::

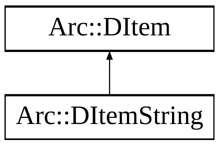


The documentation for this class was generated from the following file:

- DBranch.h

## 6.96 Arc::DItemString Class Reference

Inheritance diagram for Arc::DItemString::

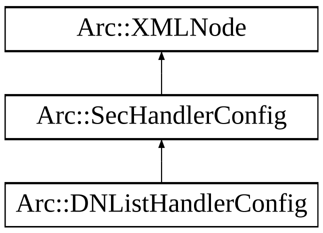


The documentation for this class was generated from the following file:

- DBranch.h

## 6.97 Arc::DNListHandlerConfig Class Reference

Inheritance diagram for Arc::DNListHandlerConfig::

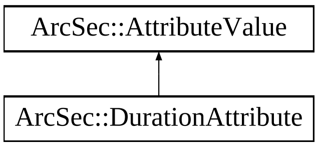


The documentation for this class was generated from the following file:

- ClientInterface.h

## 6.98 ArcSec::DurationAttribute Class Reference

`#include <DateTimeAttribute.h>`Inheritance diagram for ArcSec::DurationAttribute::



### Public Member Functions

- virtual std::string **encode** ()
- virtual std::string **getType** ()
- virtual std::string **getId** ()

### 6.98.1 Detailed Description

Formate: P??Y??M??DT??H??M??S

### 6.98.2 Member Function Documentation

#### 6.98.2.1 virtual std::string ArcSec::DurationAttribute::encode () `[virtual]`

encode the value in a string format

Implements **ArcSec::AttributeValue** (p. 78).

#### 6.98.2.2 virtual std::string ArcSec::DurationAttribute::getId () `[inline, virtual]`

Get the AttributeId of the <Attribute>

Implements **ArcSec::AttributeValue** (p. 78).

#### 6.98.2.3 virtual std::string ArcSec::DurationAttribute::getType () `[inline, virtual]`

Get the DataType of the <Attribute>

Implements **ArcSec::AttributeValue** (p. 78).

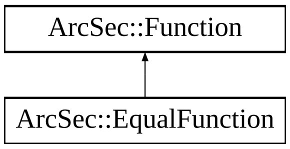
The documentation for this class was generated from the following file:

- DateTimeAttribute.h

## 6.99 ArcSec::EqualFunction Class Reference

Evaluate whether the two values are equal.

`#include <EqualFunction.h>`Inheritance diagram for ArcSec::EqualFunction::



### Public Member Functions

- virtual **AttributeValue** ∗ **evaluate** (**AttributeValue** ∗arg0, **AttributeValue** ∗arg1, bool check_-id=true)
- virtual std::list< **AttributeValue** ∗ > **evaluate** (std::list< **AttributeValue** ∗ > args, bool check_-id=true)

### Static Public Member Functions

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

### 6.99.1 Detailed Description

Evaluate whether the two values are equal.

### 6.99.2 Member Function Documentation

#### 6.99.2.1 virtual std::list<AttributeValue∗> ArcSec::EqualFunction::evaluate (std::list< AttributeValue ∗ > *args*, bool *check_id* = `true`) `[virtual]`

Evaluate a list of **AttributeValue** (p. 77) objects, and return a list of Attribute objects

Implements **ArcSec::Function** (p. 222).

#### 6.99.2.2 virtual AttributeValue∗ ArcSec::EqualFunction::evaluate (AttributeValue ∗ *arg0*, AttributeValue ∗ *arg1*, bool *check_id* = `true`) `[virtual]`

Evaluate two **AttributeValue** (p. 77) objects, and return one **AttributeValue** (p. 77) object

Implements **ArcSec::Function** (p. 222).

#### 6.99.2.3 static std::string ArcSec::EqualFunction::getFunctionName (std::string *datatype*) `[static]`

help function to get the FunctionName

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

- EqualFunction.h

## 6.100 ArcSec::EvalResult Struct Reference

Struct to record the xml node and effect, which will be used by **Evaluator** (p. 200) to get the information about which rule/policy(in xmlnode) is satisfied.

```
#include <Result.h>
```

### 6.100.1 Detailed Description

Struct to record the xml node and effect, which will be used by **Evaluator** (p. 200) 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

## 6.101 ArcSec::EvaluationCtx Class Reference

**EvaluationCtx** (p. 199), in charge of storing some context information for.

`#include <EvaluationCtx.h>`

### Public Member Functions

- **EvaluationCtx** (**Request** ∗request)

### 6.101.1 Detailed Description

**EvaluationCtx** (p. 199), in charge of storing some context information for.

### 6.101.2 Constructor & Destructor Documentation

#### 6.101.2.1 ArcSec::EvaluationCtx::EvaluationCtx (Request ∗ *request*) `[inline]`

Construct a new **EvaluationCtx** (p. 199) based on the given request

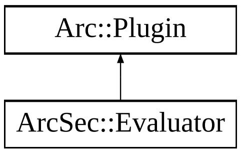
The documentation for this class was generated from the following file:

- EvaluationCtx.h

## 6.102 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

### 6.102.1 Detailed Description

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

### 6.102.2 Member Function Documentation

#### 6.102.2.1 virtual void ArcSec::Evaluator::addPolicy (Policy ∗ *policy*, const std::string & *id* = "") `[pure virtual]`

Add policy to the evaluator. **Policy** (p. 349) will be marked with id. The policy object is taken over by this instance and will be destroyed in destructor.

#### 6.102.2.2 virtual void ArcSec::Evaluator::addPolicy (const Source & *policy*, const std::string & *id* = "") `[pure virtual]`

Add policy from specified source to the evaluator. **Policy** (p. 349) will be marked with id.

#### 6.102.2.3 virtual Response∗ ArcSec::Evaluator::evaluate (EvaluationCtx ∗ *ctx*) `[protected, pure virtual]`

Evaluate the request by using the **EvaluationCtx** (p. 199) object (which includes the information about request). The ctx is destroyed inside this method (why?!?!?).

#### 6.102.2.4 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.

#### 6.102.2.5 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.

#### 6.102.2.6 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.

#### 6.102.2.7 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.

#### 6.102.2.8 virtual Response∗ ArcSec::Evaluator::evaluate (const Source & *request*) `[pure virtual]`

Evaluates the request by using a specified source

#### 6.102.2.9 virtual Response∗ ArcSec::Evaluator::evaluate (Request ∗ *request*) `[pure virtual]`

Evaluates the request by using a **Request** (p. 367) object. Evaluation is done till at least one of policies is satisfied.

#### 6.102.2.10 virtual AlgFactory∗ ArcSec::Evaluator::getAlgFactory () `[pure virtual]`

Get the **AlgFactory** (p. 63) object

Referenced by ArcSec::EvaluatorContext::operator AlgFactory ∗().

#### 6.102.2.11 virtual AttributeFactory∗ ArcSec::Evaluator::getAttrFactory () `[pure virtual]`

Get the **AttributeFactory** (p. 72) object

Referenced by ArcSec::EvaluatorContext::operator AttributeFactory ∗().

#### 6.102.2.12 virtual FnFactory∗ ArcSec::Evaluator::getFnFactory () `[pure virtual]`

Get the **FnFactory** (p. 221) object

Referenced by ArcSec::EvaluatorContext::operator FnFactory ∗().

#### 6.102.2.13 virtual const char∗ ArcSec::Evaluator::getName (void) const `[pure virtual]`

Get the name of this evaluator

#### 6.102.2.14 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.

#### 6.102.2.15 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

## 6.103 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** ∗ ()

### 6.103.1 Detailed Description

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

### 6.103.2 Member Function Documentation

#### 6.103.2.1 ArcSec::EvaluatorContext::operator AlgFactory ∗ () `[inline]`

Returns associated **AlgFactory** (p. 63) object

References ArcSec::Evaluator::getAlgFactory().

#### 6.103.2.2 ArcSec::EvaluatorContext::operator AttributeFactory ∗ () `[inline]`

Returns associated **AttributeFactory** (p. 72) object

References ArcSec::Evaluator::getAttrFactory().

#### 6.103.2.3 ArcSec::EvaluatorContext::operator FnFactory ∗ () `[inline]`

Returns associated **FnFactory** (p. 221) object

References ArcSec::Evaluator::getFnFactory().

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

- Evaluator.h

## 6.104 ArcSec::EvaluatorLoader Class Reference

**EvaluatorLoader** (p. 204) is implemented as a helper class for loading different **Evaluator** (p. 200) 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)

### 6.104.1 Detailed Description

**EvaluatorLoader** (p. 204) is implemented as a helper class for loading different **Evaluator** (p. 200) objects, like ArcEvaluator. The object loading is based on the configuration information about evaluator, including information for factory class, request, policy and evaluator itself

### 6.104.2 Member Function Documentation

#### 6.104.2.1 Evaluator∗ ArcSec::EvaluatorLoader::getEvaluator (const Request ∗ *request*)

Get evaluator object suitable for presented request

#### 6.104.2.2 Evaluator∗ ArcSec::EvaluatorLoader::getEvaluator (const Policy ∗ *policy*)

Get evaluator object suitable for presented policy

#### 6.104.2.3 Evaluator∗ ArcSec::EvaluatorLoader::getEvaluator (const std::string & *classname*)

Get evaluator object according to the class name

#### 6.104.2.4 Policy∗ ArcSec::EvaluatorLoader::getPolicy (const Source & *policysource*)

Get proper policy object according to the policy source

#### 6.104.2.5 Policy∗ ArcSec::EvaluatorLoader::getPolicy (const std::string & *classname*, const Source & *policysource*)

Get policy object according to the class name, based on the policy source

#### 6.104.2.6 Request∗ ArcSec::EvaluatorLoader::getRequest (const Source & *requestsource*)

Get request object according to the request source

#### 6.104.2.7 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

## 6.105 Arc::ExecutableType Class Reference

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

- JobDescription.h

## 6.106 Arc::ExecutionTarget Class Reference

**ExecutionTarget** (p. 207).

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

### Public Member Functions

- **ExecutionTarget** ()
- **ExecutionTarget** (const **ExecutionTarget** &target)
- **ExecutionTarget** (const long int addrptr)
- **ExecutionTarget** & **operator=** (const **ExecutionTarget** &target)
- **Submitter** ∗ **GetSubmitter** (const **UserConfig** &ucfg) const
- void **Update** (const **JobDescription** &jobdesc)
- void **Print** (bool longlist) const

### Data Fields

- int64_t **MaxMainMemory**
- int64_t **MaxVirtualMemory**
- int64_t **MaxDiskSpace**
- **Software OperatingSystem**
- std::list< **ApplicationEnvironment** > **ApplicationEnvironments**

### 6.106.1 Detailed Description

**ExecutionTarget** (p. 207). This class describe a target which accept computing jobs. All of the members contained in this class, with a few exceptions, are directly linked to attributes defined in the GLUE Specification v. 2.0 (GFD-R-P.147).

### 6.106.2 Constructor & Destructor Documentation

#### 6.106.2.1 Arc::ExecutionTarget::ExecutionTarget ()

Create an **ExecutionTarget** (p. 207). Default constructor to create an **ExecutionTarget** (p. 207). Takes no arguments.

#### 6.106.2.2 Arc::ExecutionTarget::ExecutionTarget (const ExecutionTarget & *target*)

Create an **ExecutionTarget** (p. 207). Copy constructor.

**Parameters:**

*target* **ExecutionTarget** (p. 207) to copy.

#### 6.106.2.3 Arc::ExecutionTarget::ExecutionTarget (const long int *addrptr*)

Create an **ExecutionTarget** (p. 207). Copy constructor? Needed from Python?

**Parameters:**

***addrptr***

### 6.106.3 Member Function Documentation

#### 6.106.3.1 Submitter∗ Arc::ExecutionTarget::GetSubmitter (const UserConfig & *ucfg*) const

Get **Submitter** (p. 436) to the computing resource represented by the **ExecutionTarget** (p. 207). Method which returns a specialized **Submitter** (p. 436) which can be used for submitting jobs to the computing resource represented by the **ExecutionTarget** (p. 207). In order to return the correct specialized **Submitter** (p. 436) the GridFlavour variable must be correctly set.

**Parameters:**

***ucfg*** **UserConfig** (p. 468) object with paths to user credentials etc.

#### 6.106.3.2 ExecutionTarget& Arc::ExecutionTarget::operator= (const ExecutionTarget & *target*)

Create an **ExecutionTarget** (p. 207). Assignment operator

**Parameters:**

***target*** is **ExecutionTarget** (p. 207) to copy.

#### 6.106.3.3 void Arc::ExecutionTarget::Print (bool *longlist*) const

Print the **ExecutionTarget** (p. 207) information to std::cout. Method to print the **ExecutionTarget** (p. 207) attributes to std::cout

**Parameters:**

***longlist*** is true for long list printing.

#### 6.106.3.4 void Arc::ExecutionTarget::Update (const JobDescription & *jobdesc*)

Update **ExecutionTarget** (p. 207) after succesful job submission. Method to update the **ExecutionTarget** (p. 207) after a job succesfully has been submitted to the computing resource it represents. E.g. if a job is sent to the computing resource and is expected to enter the queue, then the WaitingJobs attribute is incremented with 1.

**Parameters:**

***jobdesc*** contains all information about the job submitted.

### 6.106.4 Field Documentation

#### 6.106.4.1 std::list<ApplicationEnvironment> Arc::ExecutionTarget::ApplicationEnvironments

ApplicationEnvironments. The ApplicationEnvironments member is a list of ApplicationEnvironment's, defined in section 6.7 GLUE2.

#### 6.106.4.2 int64_t Arc::ExecutionTarget::MaxDiskSpace

MaxDiskSpace UInt64 0..1 GB. The maximum disk space that a job is allowed use in the working; if the limit is hit, then the LRMS MAY kill the job. A negative value specifies that this member is undefined.

#### 6.106.4.3 int64_t Arc::ExecutionTarget::MaxMainMemory

MaxMainMemory UInt64 0..1 MB. The maximum physical RAM that a job is allowed to use; if the limit is hit, then the LRMS MAY kill the job. A negative value specifies that this member is undefined.

#### 6.106.4.4 int64_t Arc::ExecutionTarget::MaxVirtualMemory

MaxVirtualMemory UInt64 0..1 MB. The maximum total memory size (RAM plus swap) that a job is allowed to use; if the limit is hit, then the LRMS MAY kill the job. A negative value specifies that this member is undefined.

#### 6.106.4.5 Software Arc::ExecutionTarget::OperatingSystem

OperatingSystem. The OperatingSystem member is not present in GLUE2 but contains the three GLUE2 attributes OSFamily, OSName and OSVersion.

- OSFamily OSFamily_t 1 $*$ The general family to which the Execution Environment operating $*$ system belongs.
- OSName OSName_t 0..1 $*$ The specific name of the operating sytem
- OSVersion String 0..1 $*$ The version of the operating system, as defined by the vendor.

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

- ExecutionTarget.h

## 6.107 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**

### 6.107.1 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.

### 6.107.2 Member Function Documentation

#### 6.107.2.1 Glib::TimeVal Arc::ExpirationReminder::getExpiryTime () const

Returns the expiry time. This method returns the expiry time of the reservation that this **ExpirationReminder** (p. 210) is associated with.

**Returns:**

The expiry time.

#### 6.107.2.2 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. 210) is associated with.

**Returns:**

The identification number.

#### 6.107.2.3 bool Arc::ExpirationReminder::operator< (const ExpirationReminder & *other*) const

Less than operator, compares "soonness". This is the less than operator for the **ExpirationReminder** (p. 210) 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.

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

- Counter.h

## 6.108 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)

### 6.108.1 Detailed Description

**FileCache** (p. 211) 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. 215) should be called, so that the cache file can be prepared and locked. When a transfer has finished successfully, **Link()** (p. 214) or **Copy()** (p. 213) 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. 215) 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. 456) 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. 456) 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. 456) 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. 215) creates this lock and **Stop()** (p. 215) releases it. All processes calling **Start()** (p. 215) must wait until they successfully obtain the lock before downloading can begin.

## 6.108.2 Constructor & Destructor Documentation

### 6.108.2.1 Arc::FileCache::FileCache (std::string *cache_path*, std::string *id*, uid_t *job_uid*, gid_t *job_gid*)

Create a new **FileCache** (p. 211) 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

### 6.108.2.2 Arc::FileCache::FileCache (std::vector< std::string > *caches*, std::string *id*, uid_t *job_uid*, gid_t *job_gid*)

Create a new **FileCache** (p. 211) 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

### 6.108.2.3 Arc::FileCache::FileCache (const FileCache & *cache*)

Copy constructor

### 6.108.2.4 Arc::FileCache::FileCache () `[inline]`

Default constructor. Invalid cache.

### 6.108.2.5 virtual Arc::FileCache::∼FileCache (void) `[virtual]`

Destructor

## 6.108.3 Member Function Documentation

### 6.108.3.1 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

### 6.108.3.2 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

### 6.108.3.3 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

### 6.108.3.4 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

### 6.108.3.5 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)

### 6.108.3.6 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

#### 6.108.3.7 std::string Arc::FileCache::File (std::string *url*)

Returns the full pathname of the file in the cache which corresponds to the given url.

#### 6.108.3.8 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

#### 6.108.3.9 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

#### 6.108.3.10 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. 214).

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

#### 6.108.3.11 Arc::FileCache::operator bool (void) `[inline]`

Returns true if object is useable.

#### 6.108.3.12 bool Arc::FileCache::operator== (const FileCache & *a*)

Return true if all attributes are equal

#### 6.108.3.13 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.

#### 6.108.3.14 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

#### 6.108.3.15 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

#### 6.108.3.16 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. 215) 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. 215)), or if it fails to delete the lock file.

**Parameters:**

***url*** the url of the file that was downloaded

#### 6.108.3.17 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. 215).

**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

## 6.109 FileCacheHash Class Reference

`#include <FileCacheHash.h>`

### Static Public Member Functions

- static std::string **getHash** (std::string url)
- static int **maxLength** ()

### 6.109.1 Detailed Description

**FileCacheHash** (p. 216) provides methods to make hashes from strings. Currently the md5 hash from the openssl library is used.

### 6.109.2 Member Function Documentation

#### 6.109.2.1 static std::string FileCacheHash::getHash (std::string *url*) `[static]`

Return a hash of the given URL, according to the current hash scheme.

#### 6.109.2.2 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

## 6.110 Arc::FileInfo Class Reference

**FileInfo** (p. 217) stores information about files (metadata).

```
#include <FileInfo.h>
```

### 6.110.1 Detailed Description

**FileInfo** (p. 217) stores information about files (metadata).

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

- FileInfo.h

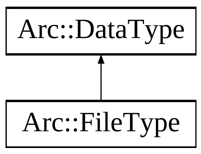
## 6.111 Arc::FileLock Class Reference

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

- FileLock.h

## 6.112 Arc::FileType Class Reference

Inheritance diagram for Arc::FileType::

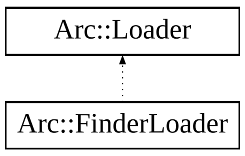


The documentation for this class was generated from the following file:

- JobDescription.h

## 6.113 Arc::FinderLoader Class Reference

Inheritance diagram for Arc::FinderLoader::



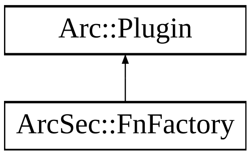
The documentation for this class was generated from the following file:

- FinderLoader.h

## 6.114 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

### 6.114.1 Detailed Description

Interface for function factory class. **FnFactory** (p. 221) is in charge of creating **Function** (p. 222) 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. 222) objects.

### 6.114.2 Member Function Documentation

#### 6.114.2.1 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. 222)

**Returns:**

The object of **Function** (p. 222)

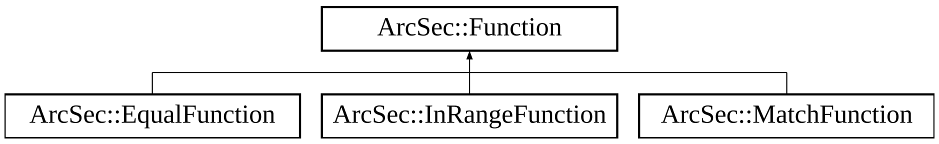
The documentation for this class was generated from the following file:

- FnFactory.h

## 6.115 ArcSec::Function Class Reference

Interface for function, which is in charge of evaluating two **AttributeValue** (p. 77).

`#include <Function.h>`Inheritance diagram for ArcSec::Function::



### Public Member Functions

- virtual **AttributeValue** ∗ **evaluate** (**AttributeValue** ∗arg0, **AttributeValue** ∗arg1, bool check_-id=true)=0
- virtual std::list< **AttributeValue** ∗ > **evaluate** (std::list< **AttributeValue** ∗ > args, bool check_-id=true)=0

### 6.115.1 Detailed Description

Interface for function, which is in charge of evaluating two **AttributeValue** (p. 77).

### 6.115.2 Member Function Documentation

#### 6.115.2.1 virtual std::list<AttributeValue∗> ArcSec::Function::evaluate (std::list< AttributeValue ∗ > *args*, bool *check_id* = `true`) `[pure virtual]`

Evaluate a list of **AttributeValue** (p. 77) objects, and return a list of Attribute objects

Implemented in **ArcSec::EqualFunction** (p. 196), **ArcSec::InRangeFunction** (p. 243), and **Arc-Sec::MatchFunction** (p. 276).

#### 6.115.2.2 virtual AttributeValue∗ ArcSec::Function::evaluate (AttributeValue ∗ *arg0*, AttributeValue ∗ *arg1*, bool *check_id* = `true`) `[pure virtual]`

Evaluate two **AttributeValue** (p. 77) objects, and return one **AttributeValue** (p. 77) object

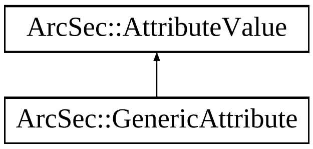
Implemented in **ArcSec::EqualFunction** (p. 196), **ArcSec::InRangeFunction** (p. 243), and **Arc-Sec::MatchFunction** (p. 276).

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

- Function.h

## 6.116 ArcSec::GenericAttribute Class Reference

Inheritance diagram for ArcSec::GenericAttribute::



### Public Member Functions

- virtual std::string **encode** ()
- virtual std::string **getType** ()
- virtual std::string **getId** ()

### 6.116.1 Member Function Documentation

#### 6.116.1.1 virtual std::string ArcSec::GenericAttribute::encode () `[inline, virtual]`

encode the value in a string format

Implements **ArcSec::AttributeValue** (p. 78).

#### 6.116.1.2 virtual std::string ArcSec::GenericAttribute::getId () `[inline, virtual]`

Get the AttributeId of the <Attribute>

Implements **ArcSec::AttributeValue** (p. 78).

#### 6.116.1.3 virtual std::string ArcSec::GenericAttribute::getType () `[inline, virtual]`

Get the DataType of the <Attribute>

Implements **ArcSec::AttributeValue** (p. 78).

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

- GenericAttribute.h

## 6.117 Arc::GlobusResult Class Reference

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

- GlobusErrorUtils.h

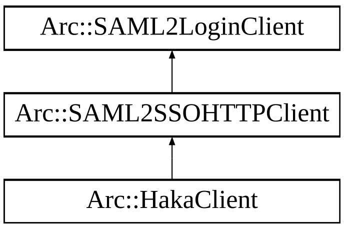
## 6.118 Arc::GSSCredential Class Reference

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

- GSSCredential.h

## 6.119 Arc::HakaClient Class Reference

Inheritance diagram for Arc::HakaClient::



### Protected Member Functions

- **MCC_Status processIdPLogin** (const std::string username, const std::string password)
- **MCC_Status processConsent** ()
- **MCC_Status processIdP2Confusa** ()

### 6.119.1 Member Function Documentation

#### 6.119.1.1 MCC_Status Arc::HakaClient::processConsent () `[protected, virtual]`

If the IdP has a consent module and the user has not saved her consent, this method will ask the user for consent to transmission of her data to Confusa

Implements **Arc::SAML2SSOHTTPClient** (p. 393).

#### 6.119.1.2 MCC_Status Arc::HakaClient::processIdP2Confusa () `[protected, virtual]`

Redirects the user back from identity provider to the Confusa SP

Implements **Arc::SAML2SSOHTTPClient** (p. 394).

#### 6.119.1.3 MCC_Status Arc::HakaClient::processIdPLogin (const std::string *username*, const std::string *password*) `[protected, virtual]`

Actual identity provider parsers for next three methods implemented in subdirectory idp/

Parse identity provider login page and submit username and password in the previsioned way

Implements **Arc::SAML2SSOHTTPClient** (p. 394).

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

- HakaClient.h

## 6.120 Arc::HTTPClientInfo Struct Reference

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

- ClientInterface.h

## 6.121 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)

### 6.121.1 Detailed Description

Stores XML document in filesystem split into parts.

### 6.121.2 Constructor & Destructor Documentation

#### 6.121.2.1 Arc::InfoCache::InfoCache (const Config & *cfg*, const std::string & *service_id*)

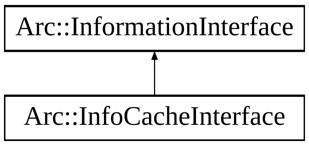
Creates object according to configuration (see InfoCacheConfig.xsd). XML configuration is passed in cfg. Argument service_id is used to distiguish between various documents stored under same path - corresponding files will be stored in subdirectory with service_id name.

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

- InfoCache.h

## 6.122 Arc::InfoCacheInterface Class Reference

Inheritance diagram for Arc::InfoCacheInterface::



### Protected Member Functions

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

### 6.122.1 Member Function Documentation

#### 6.122.1.1 virtual void Arc::InfoCacheInterface::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. 237).

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

- InfoCache.h

## 6.123 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

### 6.123.1 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.

### 6.123.2 Constructor & Destructor Documentation

#### 6.123.2.1 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.

### 6.123.3 Member Function Documentation

#### 6.123.3.1 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.

#### 6.123.3.2 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

## 6.124 Arc::InfoRegister Class Reference

Registration to ISIS interface.

`#include <InfoRegister.h>`

### 6.124.1 Detailed Description

Registration to ISIS interface. This class represents service registering to Information Indexing **Service** (p. 409). It does not perform registration itself. It only collects configuration information. Configuration is as described in InfoRegisterConfig.xsd for element InfoRegistration.

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

- InfoRegister.h

## 6.125 Arc::InfoRegisterContainer Class Reference

```
#include <InfoRegister.h>
```

### Public Member Functions

- **InfoRegistrar** ∗ **addRegistrar** (**XMLNode** doc)
- void **addService** (**InfoRegister** ∗reg, const std::list< std::string > &ids, **XMLNode** cfg=**XMLNode**())
- void **removeService** (**InfoRegister** ∗reg)

### 6.125.1 Detailed Description

Singleton class for scanning configuration and storing refernces to registration elements.

### 6.125.2 Member Function Documentation

#### 6.125.2.1 InfoRegistrar∗ Arc::InfoRegisterContainer::addRegistrar (XMLNode *doc*)

Adds ISISes to list of handled services. Supplied configuration document is scanned for **InfoRegistrar** (p. 234) elements and those are turned into **InfoRegistrar** (p. 234) classes for handling connection to ISIS service each.

#### 6.125.2.2 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.

#### 6.125.2.3 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

## 6.126 Arc::InfoRegisters Class Reference

Handling multiple registrations to ISISes.

`#include <InfoRegister.h>`

### Public Member Functions

- **InfoRegisters** (**XMLNode** &cfg, **Service** ∗service_)

### 6.126.1 Detailed Description

Handling multiple registrations to ISISes.

### 6.126.2 Constructor & Destructor Documentation

#### 6.126.2.1 Arc::InfoRegisters::InfoRegisters (XMLNode & *cfg*, Service ∗ *service_*)

Constructor creates **InfoRegister** (p. 231) objects according to configuration. Inside cfg elements InfoRegistration are found and for each corresponding **InfoRegister** (p. 231) 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

## 6.127 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** ∗)

### 6.127.1 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. 234).

### 6.127.2 Member Function Documentation

#### 6.127.2.1 bool Arc::InfoRegistrar::addService (InfoRegister ∗, XMLNode &)

Adds new service to list of handled services. **Service** (p. 409) is described by it's **InfoRegister** (p. 231) object which must be valid as long as this object is functional.

#### 6.127.2.2 void Arc::InfoRegistrar::registration (void)

Performs registartion in a loop. Never exits unless there is a critical error or requested by destructor.

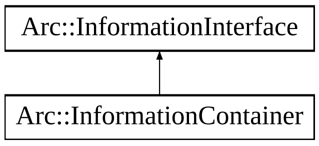
The documentation for this class was generated from the following file:

- InfoRegister.h

## 6.128 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_**

### 6.128.1 Detailed Description

Information System document container and processor. This class inherits form **InformationInterface** (p. 237) and offers container for storing informational XML document.

### 6.128.2 Constructor & Destructor Documentation

#### 6.128.2.1 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.

### 6.128.3 Member Function Documentation

#### 6.128.3.1 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()

#### 6.128.3.2 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.

#### 6.128.3.3 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. 237).

### 6.128.4 Field Documentation

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

Either link or container of XML document

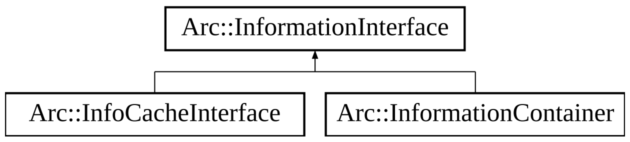
The documentation for this class was generated from the following file:

- InformationInterface.h

## 6.129 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_**

### 6.129.1 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.

### 6.129.2 Constructor & Destructor Documentation

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

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

### 6.129.3 Member Function Documentation

#### 6.129.3.1 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::InfoCacheInterface** (p. 229), and **Arc::InformationContainer** (p. 236).

### 6.129.4 Field Documentation

#### 6.129.4.1 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

## 6.130 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)

### 6.130.1 Detailed Description

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

### 6.130.2 Constructor & Destructor Documentation

#### 6.130.2.1 Arc::InformationRequest::InformationRequest (void)

Dummy constructor

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

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

#### 6.130.2.3 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.

#### 6.130.2.4 Arc::InformationRequest::InformationRequest (XMLNode *query*)

Request for attributes specified by XPath query.

### 6.130.3 Member Function Documentation

#### 6.130.3.1 SOAPEnvelope∗ Arc::InformationRequest::SOAP (void)

Returns generated SOAP message

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

- InformationInterface.h

## 6.131 Arc::InformationResponse Class Reference

Informational response from InfoSystem.

```
#include <InformationInterface.h>
```

### Public Member Functions

- **InformationResponse** (SOAPEnvelope &soap)
- std::list< **XMLNode** > **Result** (void)

### 6.131.1 Detailed Description

Informational response from InfoSystem. This is a convenience wrapper analyzing WS-ResourceProperties response from InfoSystem interface of service.

### 6.131.2 Constructor & Destructor Documentation

#### 6.131.2.1 Arc::InformationResponse::InformationResponse (SOAPEnvelope & *soap*)

Constructor parses WS-ResourceProperties ressponse. Provided SOAPEnvelope object must be valid as long as this object is in use.

### 6.131.3 Member Function Documentation

#### 6.131.3.1 std::list<XMLNode> Arc::InformationResponse::Result (void)

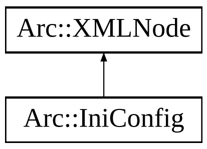
Returns set of attributes which were in SOAP message passed to constructor.

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

- InformationInterface.h

## 6.132 Arc::IniConfig Class Reference

Inheritance diagram for Arc::IniConfig::



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

- IniConfig.h

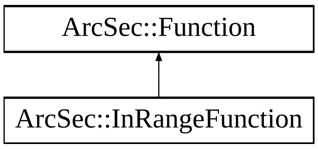
## 6.133 Arc::initializeCredentialsType Class Reference

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

- UserConfig.h

## 6.134 ArcSec::InRangeFunction Class Reference

Inheritance diagram for ArcSec::InRangeFunction::



### Public Member Functions

- virtual **AttributeValue** ∗ **evaluate** (**AttributeValue** ∗arg0, **AttributeValue** ∗arg1, bool check_-id=true)
- virtual std::list< **AttributeValue** ∗ > **evaluate** (std::list< **AttributeValue** ∗ > args, bool check_-id=true)

### 6.134.1 Member Function Documentation

#### 6.134.1.1 virtual std::list<AttributeValue∗> ArcSec::InRangeFunction::evaluate (std::list< AttributeValue ∗ > *args*, bool *check_id* = `true`) `[virtual]`

Evaluate a list of **AttributeValue** (p. 77) objects, and return a list of Attribute objects

Implements **ArcSec::Function** (p. 222).

#### 6.134.1.2 virtual AttributeValue∗ ArcSec::InRangeFunction::evaluate (AttributeValue ∗ *arg0*, AttributeValue ∗ *arg1*, bool *check_id* = `true`) `[virtual]`

Evaluate two **AttributeValue** (p. 77) objects, and return one **AttributeValue** (p. 77) object

Implements **ArcSec::Function** (p. 222).

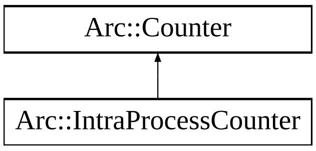
The documentation for this class was generated from the following file:

- InRangeFunction.h

## 6.135 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**)

### 6.135.1 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. 119) class for further information about counters and examples of usage.

### 6.135.2 Constructor & Destructor Documentation

#### 6.135.2.1 Arc::IntraProcessCounter::IntraProcessCounter (int *limit*, int *excess*)

Creates an **IntraProcessCounter** (p. 244) 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.

#### 6.135.2.2 virtual Arc::IntraProcessCounter::∼IntraProcessCounter () `[virtual]`

Destructor. This is the destructor of the **IntraProcessCounter** (p. 244) class. Does not need to do anything.

### 6.135.3 Member Function Documentation

#### 6.135.3.1 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. 126) that corresponds to the reservation.

**Parameters:**

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

Implements **Arc::Counter** (p. 121).

#### 6.135.3.2 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. 121).

#### 6.135.3.3 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. 122).

#### 6.135.3.4 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. 126) 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.

Implements **Arc::Counter** (p. 122).

#### 6.135.3.5 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. 123).

#### 6.135.3.6 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. 123).

#### 6.135.3.7 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. 124).

#### 6.135.3.8 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. 126) that can be queried about the status of the reservation as well as for cancellations and extensions.

Implements **Arc::Counter** (p. 124).

### 6.135.3.9 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. 124).

### 6.135.3.10 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. 125).

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

- IntraProcessCounter.h

## 6.136 Arc::ISIS_description Struct Reference

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

- InfoRegister.h

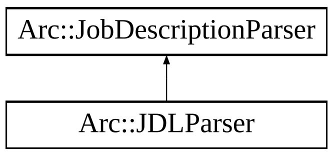
## 6.137 Arc::IString Class Reference

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

- IString.h

## 6.138 Arc::JDLParser Class Reference

Inheritance diagram for Arc::JDLParser::



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

- JDLParser.h

## 6.139 Arc::Job Class Reference

**Job** (p. 251).

`#include <Job.h>`

### Public Member Functions

- **Job** ()
- void **Print** (bool longlist) const

### 6.139.1 Detailed Description

**Job** (p. 251). This class describe a Grid job. Most of the members contained in this class are directly linked to the ComputingActivity defined in the GLUE Specification v. 2.0 (GFD-R-P.147).

### 6.139.2 Constructor & Destructor Documentation

#### 6.139.2.1 Arc::Job::Job ()

Create a **Job** (p. 251) object. Default constructor. Takes no arguments.

### 6.139.3 Member Function Documentation

#### 6.139.3.1 void Arc::Job::Print (bool *longlist*) const

Print the **Job** (p. 251) information to std::cout. Method to print the **Job** (p. 251) attributes to std::cout

**Parameters:**

***longlist*** is boolean for long listing (more details).

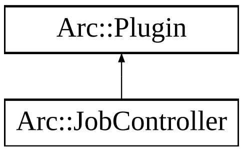
The documentation for this class was generated from the following file:

- Job.h

## 6.140 Arc::JobController Class Reference

Base class for the JobControllers.

`#include <JobController.h>`Inheritance diagram for Arc::JobController::



### Public Member Functions

- void **FillJobStore** (const std::list< **URL** > &jobids)
- bool **PrintJobStatus** (const std::list< std::string > &status, const bool longlist)
- bool **Migrate** (**TargetGenerator** &targetGen, **Broker** ∗broker, const **UserConfig** &usercfg, const bool forcemigration, std::list< **URL** > &migratedJobIDs)

### 6.140.1 Detailed Description

Base class for the JobControllers. The **JobController** (p. 252) is the base class for middleware specialized derived classes. The **JobController** (p. 252) base class is also the implementer of all public functionality that should be offered by the middleware specific specializations. In other words all virtual functions of the **JobController** (p. 252) are private. The initialization of a (specialized) **JobController** (p. 252) object takes two steps. First the **JobController** (p. 252) specialization for the required grid flavour must be loaded by the **JobControllerLoader** (p. 254), which sees to that the **JobController** (p. 252) receives information about its Grid flavour and the local joblist file containing information about all active jobs (flavour independent). The next step is the filling of the **JobController** (p. 252) job pool (JobStore) which is the pool of jobs that the **JobController** (p. 252) can manage. Must be specialiced for each supported middleware flavour.

### 6.140.2 Member Function Documentation

#### 6.140.2.1 void Arc::JobController::FillJobStore (const std::list< URL > & *jobids*)

Fill jobstore. Method to fill the jobstore with jobs that should be managed.

**Parameters:**

*jobids* List of jobids to be loaded to the jobstore. If empty all jobs of the specialized grid flavour present in the joblist file (given through the usercfg to the constructor) will be loaded to the jobstore.

#### 6.140.2.2 bool Arc::JobController::Migrate (TargetGenerator & *targetGen*, Broker ∗ *broker*, const UserConfig & *usercfg*, const bool *forcemigration*, std::list< URL > & *migratedJobIDs*)

Migrate job from cluster A to Cluster B. Method to migrate the jobs contained in the jobstore.

**Parameters:**

***targetGen*** **TargetGenerator** (p. 440) with targets to migrate the job to.

***broker*** **Broker** (p. 87) to be used when selecting target.

***forcemigration*** boolean which specifies whether a migrated job should persist if the new cluster does not succeed sending a kill/terminate request for the job.

#### 6.140.2.3 bool Arc::JobController::PrintJobStatus (const std::list< std::string > & *status*, const bool *longlist*)

Print job status to stdout. The job status is printed to stdout when calling this method. More specifically the **Job::Print** (p. 251) method is called on each of the **Job** (p. 251) objects stored in this object, and the boolean argument *longlist* is passed directly to the method indicating whether verbose job status should be printed. The *status* argument is a list of strings each representing a job state (**JobState** (p. 261)) which is used to indicate that only jobs with a job state in the list should be considered. If the list *status* is empty all jobs will be considered.

This method is not supposed to be overloaded by extending classes.

**Parameters:**

***status*** a list of strings representing states to be considered.

***longlist*** a boolean indicating whether verbose job information should be printed.

**Returns:**

This method always returns true.

**See also:**

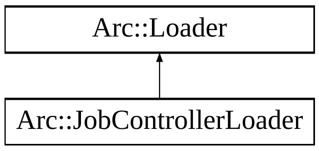
GetJobInformation
**Job::Print** (p. 251)
**JobState** (p. 261)

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

- JobController.h

## 6.141 Arc::JobControllerLoader Class Reference

`#include <JobController.h>`Inheritance diagram for Arc::JobControllerLoader::



### Public Member Functions

- **JobControllerLoader** ()
- ∼**JobControllerLoader** ()
- **JobController** ∗ **load** (const std::string &name, const **UserConfig** &usercfg)
- const std::list< **JobController** ∗ > & **GetJobControllers** () const

### 6.141.1 Detailed Description

Class responsible for loading **JobController** (p. 252) plugins The **JobController** (p. 252) objects returned by a **JobControllerLoader** (p. 254) must not be used after the **JobControllerLoader** (p. 254) goes out of scope.

### 6.141.2 Constructor & Destructor Documentation

#### 6.141.2.1 Arc::JobControllerLoader::JobControllerLoader ()

Constructor Creates a new **JobControllerLoader** (p. 254).

#### 6.141.2.2 Arc::JobControllerLoader::∼JobControllerLoader ()

Destructor Calling the destructor destroys all JobControllers loaded by the **JobControllerLoader** (p. 254) instance.

### 6.141.3 Member Function Documentation

#### 6.141.3.1 const std::list<JobController∗>& Arc::JobControllerLoader::GetJobControllers () const `[inline]`

Retrieve the list of loaded JobControllers.

**Returns:**

A reference to the list of JobControllers.

Referenced by Arc::JobSupervisor::GetJobControllers().

#### 6.141.3.2 JobController∗ Arc::JobControllerLoader::load (const std::string & *name*, const UserConfig & *usercfg*)

Load a new **JobController** (p. 252)

**Parameters:**

*name* The name of the **JobController** (p. 252) to load.

*usercfg* The **UserConfig** (p. 468) object for the new **JobController** (p. 252).

**Returns:**

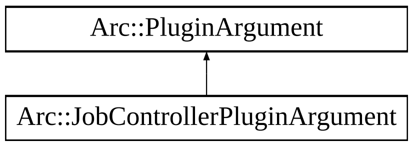
A pointer to the new **JobController** (p. 252) (NULL on error).

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

- JobController.h

## 6.142 Arc::JobControllerPluginArgument Class Reference

Inheritance diagram for Arc::JobControllerPluginArgument::



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

- JobController.h

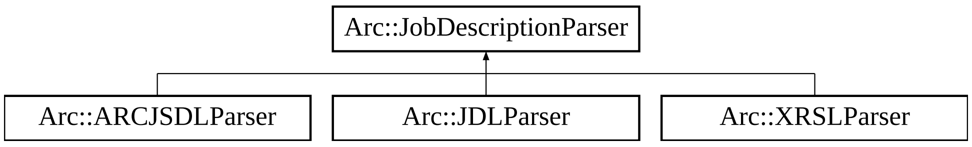
## 6.143 Arc::JobDescription Class Reference

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

- JobDescription.h

## 6.144 Arc::JobDescriptionParser Class Reference

Inheritance diagram for Arc::JobDescriptionParser::



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

- JobDescriptionParser.h

## 6.145 Arc::JobIdentificationType Class Reference

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

- JobDescription.h

## 6.146 Arc::JobMetaType Class Reference

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

- JobDescription.h

## 6.147 Arc::JobState Class Reference

`#include <JobState.h>`

### 6.147.1 Detailed Description

ARC general state model. The class comprise the general state model of the ARC-lib, and are herein used to compare job states from the different middlewares supported by the plugin structure of the ARC-lib. Which is why every ACC plugin should contain a class derived from this class. The derived class should consist of a constructor and a mapping function (a JobStateMap) which maps a std::string to a **JobState** (p. 261):StateType. An example of a constructor in a plugin could be: JobStatePlugin::JobStatePluging(const std::string& state) : JobState(state, &pluginStateMap) { } where &pluginStateMap is a reference to the JobStateMap defined by the derived class.

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

- JobState.h

## 6.148 Arc::JobSupervisor Class Reference

% **JobSupervisor** (p. 262) class

`#include <JobSupervisor.h>`

### Public Member Functions

- **JobSupervisor** (const **UserConfig** &usercfg, const std::list< std::string > &jobs)
- const std::list< **JobController** ∗ > & **GetJobControllers** ()

### 6.148.1 Detailed Description

% **JobSupervisor** (p. 262) class The **JobSupervisor** (p. 262) class is tool for loading **JobController** (p. 252) plugins for managing Grid jobs.

### 6.148.2 Constructor & Destructor Documentation

#### 6.148.2.1 Arc::JobSupervisor::JobSupervisor (const UserConfig & *usercfg*, const std::list< std::string > & *jobs*)

Create a **JobSupervisor** (p. 262) object. Default constructor to create a **JobSupervisor** (p. 262). Automatically loads **JobController** (p. 252) plugins based upon the input jobids.

**Parameters:**

*usercfg* Reference to **UserConfig** (p. 468) object with information about user credentials and joblistfile.

*jobs* List of jobs(jobid or job name) to be managed.

### 6.148.3 Member Function Documentation

#### 6.148.3.1 const std::list<JobController∗>& Arc::JobSupervisor::GetJobControllers () `[inline]`

Get list of JobControllers. Method to get the list of JobControllers loaded by constructor.

References Arc::JobControllerLoader::GetJobControllers().

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

- JobSupervisor.h

## 6.149 Arc::LoadableModuleDesciption Class Reference

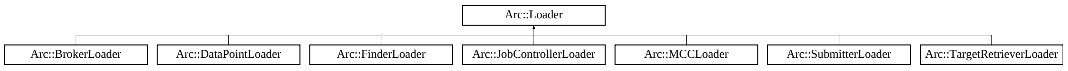
The documentation for this class was generated from the following file:

- ModuleManager.h

## 6.150 Arc::Loader Class Reference

Plugins loader.

`#include <Loader.h>`Inheritance diagram for Arc::Loader::



### Public Member Functions

- **Loader** (const **Config** &cfg)
- ∼**Loader** ()

### Protected Attributes

- **PluginsFactory** ∗ **factory_**

### 6.150.1 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.

### 6.150.2 Constructor & Destructor Documentation

#### 6.150.2.1 Arc::Loader::Loader (const Config & *cfg*)

Constructor that takes whole XML configuration and performs common configuration part

#### 6.150.2.2 Arc::Loader::∼Loader ()

Destructor destroys all components created by constructor

### 6.150.3 Field Documentation

#### 6.150.3.1 PluginsFactory∗ Arc::Loader::factory_ `[protected]`

Link to Factory responsible for loading and creation of **Plugin** (p. 343) and derived objects

Referenced by Arc::ChainContext::operator PluginsFactory ∗().

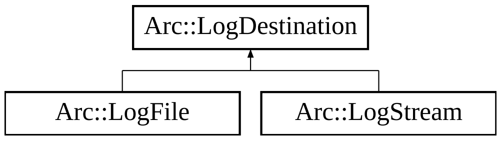
The documentation for this class was generated from the following file:

- Loader.h

## 6.151 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)

### 6.151.1 Detailed Description

A base class for log destinations. This class defines an interface for LogDestinations. **LogDestination** (p. 265) objects will typically contain synchronization mechanisms and should therefore never be copied.

### 6.151.2 Constructor & Destructor Documentation

#### 6.151.2.1 Arc::LogDestination::LogDestination () `[protected]`

Default constructor. This destination will use the default locale.

#### 6.151.2.2 Arc::LogDestination::LogDestination (const std::string & *locale*) `[protected]`

Constructor with specific locale. This destination will use the specified locale.

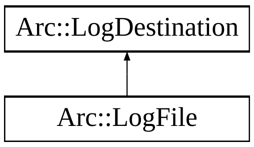
The documentation for this class was generated from the following file:

- Logger.h

## 6.152 Arc::LogFile Class Reference

A class for logging to files.

`#include <Logger.h>`Inheritance diagram for Arc::LogFile::



### Public Member Functions

- **LogFile** (const std::string &path)
- **LogFile** (const std::string &path, const std::string &locale)
- void **setMaxSize** (int newsize)
- void **setBackups** (int newbackup)
- **operator bool** (void)
- bool **operator!** (void)
- virtual void **log** (const **LogMessage** &message)

### 6.152.1 Detailed Description

A class for logging to files. This class is used for logging to files. It provides synchronization in order to prevent different LogMessages to appear mixed with each other in the stream. It is possible to limit size of created file. Whenever specified size is exceeded fiel is deleted and new one is created. Old files may be moved into backup files instead of being deleted. Those files have names same as initial file with additional number suffix - similar to those found in /var/log of many Unix-like systems.

### 6.152.2 Constructor & Destructor Documentation

#### 6.152.2.1 Arc::LogFile::LogFile (const std::string & *path*)

Creates a **LogFile** (p. 266) connected to a file. Creates a **LogFile** (p. 266) connected to the file located at specified path. In order not to break synchronization, it is important not to connect more than one **LogFile** (p. 266) object to a certain file. If file does not exist it will be created.

**Parameters:**

***path*** The path to file to which to write LogMessages.

#### 6.152.2.2 Arc::LogFile::LogFile (const std::string & *path*, const std::string & *locale*)

Creates a **LogFile** (p. 266) connected to a file. Creates a **LogFile** (p. 266) connected to the file located at specified path. The output will be localised to the specified locale.

## 6.152.3 Member Function Documentation

### 6.152.3.1 virtual void Arc::LogFile::log (const LogMessage & *message*) `[virtual]`

Writes a **LogMessage** (p. 272) to the file. This method writes a **LogMessage** (p. 272) to the file that is connected to this **LogFile** (p. 266) object. If after writitng size of file exceeds one set by **setMaxSize()** (p. 267) file is moved to backup and new one is created.

**Parameters:**

***message*** The **LogMessage** (p. 272) to write.

Implements **Arc::LogDestination** (p. 265).

### 6.152.3.2 void Arc::LogFile::setBackups (int *newbackup*)

Set number of backups to store. Set number of backups to store. When file size exceeds one specified with **setMaxSize()** (p. 267) file is closed and moved to one named path.1. If path.1 exists it is moved to path.2 and so on. Number of path.# files is one set in newbackup.

**Parameters:**

***newbackup*** Number of backup files.

### 6.152.3.3 void Arc::LogFile::setMaxSize (int *newsize*)

Set maximal allowed size of file. Set maximal allowed size of file. This value is not obeyed exactly. Spesified size may be exceeded by amount of one **LogMessage** (p. 272). To disable limit specify -1.

**Parameters:**

***newsize*** Max size of log file.

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

- Logger.h

## 6.153 Arc::Logger Class Reference

A logger class.

```
#include <Logger.h>
```

### Public Member Functions

- **Logger** (**Logger** &parent, const std::string &subdomain)
- **Logger** (**Logger** &parent, const std::string &subdomain, **LogLevel** threshold)
- ∼**Logger** ()
- void **addDestination** (**LogDestination** &destination)
- void **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** ()

### 6.153.1 Detailed Description

A logger class. This class defines a **Logger** (p. 268) to which LogMessages can be sent.

Every **Logger** (p. 268) (except for the rootLogger) has a parent **Logger** (p. 268). The domain of a **Logger** (p. 268) (a string that indicates the origin of LogMessages) is composed by adding a subdomain to the domain of its parent **Logger** (p. 268).

A **Logger** (p. 268) also has a threshold. Every **LogMessage** (p. 272) that have a level that is greater than or equal to the threshold is forwarded to any **LogDestination** (p. 265) connected to this **Logger** (p. 268) as well as to the parent **Logger** (p. 268).

Typical usage of the **Logger** (p. 268) class is to declare a global **Logger** (p. 268) object for each library/-module/component to be used by all classes and methods there.

### 6.153.2 Constructor & Destructor Documentation

#### 6.153.2.1 Arc::Logger::Logger (Logger & *parent*, const std::string & *subdomain*)

Creates a logger. Creates a logger. The threshold is inherited from its parent **Logger** (p. 268).

**Parameters:**

***parent*** The parent **Logger** (p. 268) of the new **Logger** (p. 268).

***subdomain*** The subdomain of the new logger.

#### 6.153.2.2 Arc::Logger::Logger (Logger & *parent*, const std::string & *subdomain*, LogLevel *threshold*)

Creates a logger. Creates a logger.

**Parameters:**

***parent*** The parent **Logger** (p. 268) of the new **Logger** (p. 268).

***subdomain*** The subdomain of the new logger.

***threshold*** The threshold of the new logger.

#### 6.153.2.3 Arc::Logger::∼Logger ()

Destroys a logger. Destructor

### 6.153.3 Member Function Documentation

#### 6.153.3.1 void Arc::Logger::addDestination (LogDestination & *destination*)

Adds a **LogDestination** (p. 265). Adds a **LogDestination** (p. 265) to which to forward LogMessages sent to this logger (if they pass the threshold). Since LogDestinatoins should not be copied, the new **LogDestination** (p. 265) is passed by reference and a pointer to it is kept for later use. It is therefore important that the **LogDestination** (p. 265) passed to this **Logger** (p. 268) exists at least as long as the **Logger** (p. 268) iteslf.

#### 6.153.3.2 static Logger& Arc::Logger::getRootLogger () `[static]`

The root **Logger** (p. 268). This is the root **Logger** (p. 268). It is an ancestor of any other **Logger** (p. 268) and allways exists.

#### 6.153.3.3 LogLevel Arc::Logger::getThreshold () const

Returns the threshold. Returns the threshold.

**Returns:**

The threshold of this **Logger** (p. 268).

#### 6.153.3.4 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. 272) and sends it to the other **msg()** (p. 270) method.

**Parameters:**

***level*** The level of the message.

***str*** The message text.

References msg().

#### 6.153.3.5 void Arc::Logger::msg (LogMessage *message*)

Sends a **LogMessage** (p. 272). Sends a **LogMessage** (p. 272).

**Parameters:**

***The*** **LogMessage** (p. 272) to send.

Referenced by msg(), and Arc::stringto().

#### 6.153.3.6 void Arc::Logger::setThreshold (LogLevel *threshold*)

Sets the threshold. This method sets the threshold of the **Logger** (p. 268). Any message sent to this **Logger** (p. 268) 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

## 6.154 Arc::LoggerFormat Struct Reference

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

- Logger.h

## 6.155 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)

### 6.155.1 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.

### 6.155.2 Constructor & Destructor Documentation

#### 6.155.2.1 Arc::LogMessage::LogMessage (LogLevel *level*, const IString & *message*)

Creates a **LogMessage** (p. 272) with the specified level and message text. This constructor creates a **LogMessage** (p. 272) with the specified level and message text. The time is set automatically, the domain is set by the **Logger** (p. 268) to which the **LogMessage** (p. 272) 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. 272).

***message*** The message text.

#### 6.155.2.2 Arc::LogMessage::LogMessage (LogLevel *level*, const IString & *message*, const std::string & *identifier*)

Creates a **LogMessage** (p. 272) with the specified attributes. This constructor creates a **LogMessage** (p. 272) with the specified level, message text and identifier. The time is set automatically and the domain is set by the **Logger** (p. 268) to which the **LogMessage** (p. 272) is sent.

**Parameters:**

***level*** The level of the **LogMessage** (p. 272).

***message*** The message text.

***ident*** The identifier of the **LogMessage** (p. 272).

## 6.155.3 Member Function Documentation

### 6.155.3.1 LogLevel Arc::LogMessage::getLevel () const

Returns the level of the **LogMessage** (p. 272). Returns the level of the **LogMessage** (p. 272).

**Returns:**

The level of the **LogMessage** (p. 272).

### 6.155.3.2 void Arc::LogMessage::setIdentifier (std::string *identifier*) `[protected]`

Sets the identifier of the **LogMessage** (p. 272). The purpose of this method is to allow subclasses (in case there are any) to set the identifier of a **LogMessage** (p. 272).

**Parameters:**

***The*** identifier.

## 6.155.4 Friends And Related Function Documentation

### 6.155.4.1 friend class Logger `[friend]`

The **Logger** (p. 268) class is a friend. The **Logger** (p. 268) class must have some privileges (e.g. ability to call the setDomain() method), therefore it is a friend.

### 6.155.4.2 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.

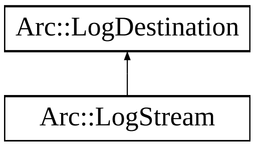
The documentation for this class was generated from the following file:

- Logger.h

## 6.156 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)

### 6.156.1 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. 274) object as long as the **Logger** (p. 268) to which it has been registered.

### 6.156.2 Constructor & Destructor Documentation

#### 6.156.2.1 Arc::LogStream::LogStream (std::ostream & *destination*)

Creates a **LogStream** (p. 274) connected to an ostream. Creates a **LogStream** (p. 274) connected to the specified ostream. In order not to break synchronization, it is important not to connect more than one **LogStream** (p. 274) object to a certain stream.

**Parameters:**

***destination*** The ostream to which to erite LogMessages.

#### 6.156.2.2 Arc::LogStream::LogStream (std::ostream & *destination*, const std::string & *locale*)

Creates a **LogStream** (p. 274) connected to an ostream. Creates a **LogStream** (p. 274) connected to the specified ostream. The output will be localised to the specified locale.

### 6.156.3 Member Function Documentation

#### 6.156.3.1 virtual void Arc::LogStream::log (const LogMessage & *message*) `[virtual]`

Writes a **LogMessage** (p. 272) to the stream. This method writes a **LogMessage** (p. 272) to the ostream that is connected to this **LogStream** (p. 274) object. It is synchronized so that not more than one **LogMessage** (p. 272) can be written at a time.

**Parameters:**

***message*** The **LogMessage** (p. 272) to write.

Implements **Arc::LogDestination** (p. 265).

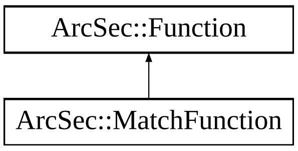
The documentation for this class was generated from the following file:

- Logger.h

## 6.157 ArcSec::MatchFunction Class Reference

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

`#include <MatchFunction.h>`Inheritance diagram for ArcSec::MatchFunction::



### Public Member Functions

- virtual **AttributeValue** ∗ **evaluate** (**AttributeValue** ∗arg0, **AttributeValue** ∗arg1, bool check_-id=true)
- virtual std::list< **AttributeValue** ∗ > **evaluate** (std::list< **AttributeValue** ∗ > args, bool check_-id=true)

### Static Public Member Functions

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

### 6.157.1 Detailed Description

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

### 6.157.2 Member Function Documentation

#### 6.157.2.1 virtual std::list<AttributeValue∗> ArcSec::MatchFunction::evaluate (std::list< AttributeValue ∗ > *args*, bool *check_id* = `true`) `[virtual]`

Evaluate a list of **AttributeValue** (p. 77) objects, and return a list of Attribute objects

Implements **ArcSec::Function** (p. 222).

#### 6.157.2.2 virtual AttributeValue∗ ArcSec::MatchFunction::evaluate (AttributeValue ∗ *arg0*, AttributeValue ∗ *arg1*, bool *check_id* = `true`) `[virtual]`

Evaluate two **AttributeValue** (p. 77) objects, and return one **AttributeValue** (p. 77) object

Implements **ArcSec::Function** (p. 222).

#### 6.157.2.3 static std::string ArcSec::MatchFunction::getFunctionName (std::string *datatype*) `[static]`

help function to get the FunctionName

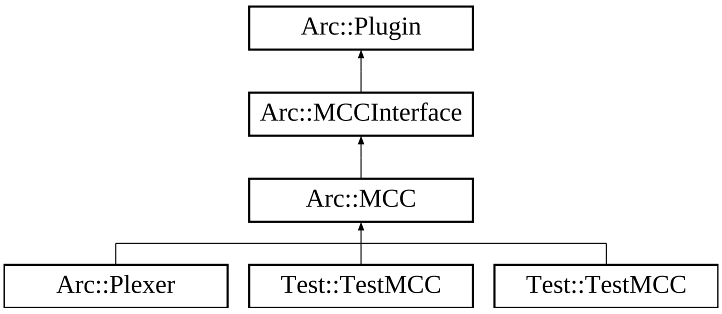
The documentation for this class was generated from the following file:

- MatchFunction.h

## 6.158 Arc::MCC Class Reference

**Message** (p. 290) Chain Component - base class for every **MCC** (p. 278) 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**

### 6.158.1 Detailed Description

**Message** (p. 290) Chain Component - base class for every **MCC** (p. 278) plugin. This is partialy virtual class which defines interface and common functionality for every **MCC** (p. 278) plugin needed for managing of component in a chain.

## 6.158.2 Constructor & Destructor Documentation

### 6.158.2.1 Arc::MCC::MCC (Config ∗) `[inline]`

Example contructor - **MCC** (p. 278) takes at least it's configuration subtree

## 6.158.3 Member Function Documentation

### 6.158.3.1 virtual void Arc::MCC::AddSecHandler (Config ∗ *cfg*, ArcSec::SecHandler ∗ *sechandler*, const std::string & *label* = `""`) `[virtual]`

Add security components/handlers to this **MCC** (p. 278). 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. 278) 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. 278) 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.

### 6.158.3.2 virtual void Arc::MCC::Next (MCCInterface ∗ *next*, const std::string & *label* = `""`) `[virtual]`

Add reference to next **MCC** (p. 278) in chain. This method is called by **Loader** (p. 264) for every potentially labeled link to next component which implements **MCCInterface** (p. 284). If next is NULL corresponding link is removed.

Reimplemented in **Arc::Plexer** (p. 341).

### 6.158.3.3 virtual MCC_Status Arc::MCC::process (Message &, Message &) `[inline, virtual]`

Dummy **Message** (p. 290) processing method. Just a placeholder.

Implements **Arc::MCCInterface** (p. 284).

Reimplemented in **Arc::Plexer** (p. 341).

### 6.158.3.4 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. 278).

### 6.158.3.5 virtual void Arc::MCC::Unlink () `[virtual]`

Removing all links. Useful for destroying chains.

### 6.158.4 Field Documentation

#### 6.158.4.1 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. 341).

#### 6.158.4.2 std::map<std::string, MCCInterface ∗> Arc::MCC::next_ `[protected]`

Set of labeled "next" components. Each implemented **MCC** (p. 278) must call **process()** (p. 279) method of corresponding **MCCInterface** (p. 284) from this set in own **process()** (p. 279) method.

#### 6.158.4.3 std::map<std::string, std::list<ArcSec::SecHandler ∗> > Arc::MCC::sechandlers_ `[protected]`

Set of labeled authentication and authorization handlers. **MCC** (p. 278) 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

## 6.159 Arc::MCC_Status Class Reference

A class for communication of **MCC** (p. 278) 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

### 6.159.1 Detailed Description

A class for communication of **MCC** (p. 278) processing results. This class is used to communicate result status between MCCs. It contains a status kind, a string specifying the origin (**MCC** (p. 278)) of the status object and an explanation.

### 6.159.2 Constructor & Destructor Documentation

#### 6.159.2.1 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. 281) object.

**Parameters:**

***kind*** The StatusKind (default: STATUS_UNDEFINED)

***origin*** The origin **MCC** (p. 278) (default: "???")

***explanation*** An explanation (default: "No explanation.")

### 6.159.3 Member Function Documentation

#### 6.159.3.1 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.

#### 6.159.3.2 StatusKind Arc::MCC_Status::getKind () const

Returns the status kind. Returns the status kind of this object.

**Returns:**

The status kind of this object.

#### 6.159.3.3 const std::string& Arc::MCC_Status::getOrigin () const

Returns the origin. This method returns a string specifying the origin **MCC** (p. 278) of this object.

**Returns:**

A string specifying the origin **MCC** (p. 278) of this object.

#### 6.159.3.4 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!().

#### 6.159.3.5 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().

#### 6.159.3.6 Arc::MCC_Status::operator std::string () const

Conversion to string. This operator converts a **MCC_Status** (p. 281) object to a string.

#### 6.159.3.7 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

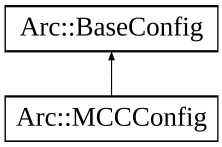
References isOk().

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

- MCC_Status.h

## 6.160 Arc::MCCConfig Class Reference

Inheritance diagram for Arc::MCCConfig::



### Public Member Functions

- virtual **XMLNode MakeConfig** (**XMLNode** cfg) const

### 6.160.1 Member Function Documentation

#### 6.160.1.1 virtual XMLNode Arc::MCCConfig::MakeConfig (XMLNode *cfg*) const `[virtual]`

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

Reimplemented from **Arc::BaseConfig** (p. 85).

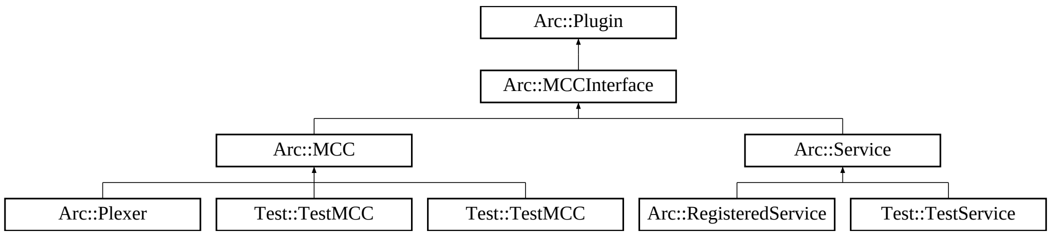
The documentation for this class was generated from the following file:

- MCC.h

## 6.161 Arc::MCCInterface Class Reference

Interface for communication between **MCC** (p. 278), **Service** (p. 409) and **Plexer** (p. 340) objects.

`#include <MCC.h>`Inheritance diagram for Arc::MCCInterface::



### Public Member Functions

- virtual **MCC_Status process** (**Message** &request, **Message** &response)=0

### 6.161.1 Detailed Description

Interface for communication between **MCC** (p. 278), **Service** (p. 409) and **Plexer** (p. 340) objects. The Interface consists of the method **process()** (p. 284) which is called by the previous **MCC** (p. 278) in the chain. For memory management policies please read the description of the **Message** (p. 290) class.

### 6.161.2 Member Function Documentation

#### 6.161.2.1 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. 278) in chain when a request needs to be processed. This method must call similar method of next **MCC** (p. 278) in chain unless any failure happens. Result returned by call to next **MCC** (p. 278) should be processed and passed back to previous **MCC** (p. 278). In case of failure this method is expected to generate valid error response and return it back to previous **MCC** (p. 278) without calling the next one.

**Parameters:**

***request*** The request that needs to be processed.

***response*** A **Message** (p. 290) object that will contain the response of the request when the method returns.

**Returns:**

An object representing the status of the call.

Implemented in **Test::TestService** (p. 449), **Arc::MCC** (p. 279), and **Arc::Plexer** (p. 341).

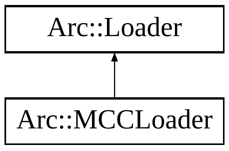
The documentation for this class was generated from the following file:

- MCC.h

## 6.162 Arc::MCCLoader Class Reference

Creator of **Message** (p. 290) Component Chains (**MCC** (p. 278)).

`#include <MCCLoader.h>`Inheritance diagram for Arc::MCCLoader::



### Public Member Functions

- **MCCLoader** (**Config** &cfg)
- ∼**MCCLoader** ()
- **MCC** ∗ **operator[ ]** (const std::string &id)

### 6.162.1 Detailed Description

Creator of **Message** (p. 290) Component Chains (**MCC** (p. 278)). 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. 278), **Service** (p. 409) and **Plexer** (p. 340). **MCC** (p. 278) and **Service** (p. 409) are loaded from dynamic libraries. For **Plexer** (p. 340) 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. 290) 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.

### 6.162.2 Constructor & Destructor Documentation

#### 6.162.2.1 Arc::MCCLoader::MCCLoader (Config & *cfg*)

Constructor that takes whole XML configuration and creates component chains

#### 6.162.2.2 Arc::MCCLoader::∼MCCLoader ()

Destructor destroys all components created by constructor

### 6.162.3 Member Function Documentation

#### 6.162.3.1 MCC∗ Arc::MCCLoader::operator[ ] (const std::string & *id*)

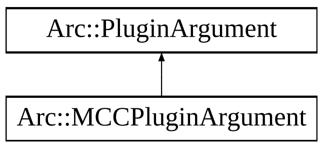
Access entry MCCs in chains. Those are components exposed for external access using 'entry' attribute

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

- MCCLoader.h

## 6.163 Arc::MCCPluginArgument Class Reference

Inheritance diagram for Arc::MCCPluginArgument::



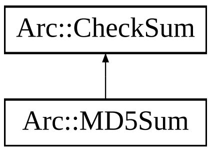
The documentation for this class was generated from the following file:

- MCC.h

## 6.164 Arc::MD5Sum Class Reference

Implementation of MD5 checksum.

`#include <CheckSum.h>`Inheritance diagram for Arc::MD5Sum::



### 6.164.1 Detailed Description

Implementation of MD5 checksum.

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

- CheckSum.h

## 6.165 Arc::MemoryAllocationException Class Reference

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

- ByteArray.h

## 6.166 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)

### 6.166.1 Detailed Description

Object being passed through chain of MCCs. An instance of this class refers to objects with main content (**MessagePayload** (p. 300)), authentication/authorization information (**MessageAuth** (p. 296)) and common purpose attributes (**MessageAttributes** (p. 293)). **Message** (p. 290) class does not manage pointers to objects and their content. It only serves for grouping those objects. **Message** (p. 290) objects are supposed to be processed by MCCs and Services implementing **MCCInterface** (p. 284) method process(). All objects constituting content of **Message** (p. 290) 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. 290). b) Objects whose management is completely acquired by objects assigned to 'response' **Message** (p. 290).

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. 290) object).

4. It is allowed to change content of pointers of 'request' **Message** (p. 290). Calling process() method must not rely on that object to stay intact.

5. Called process() method should either fill 'response' **Message** (p. 290) with pointers to valid objects or to keep them intact. This makes it possible for calling process() to preload 'response' with valid error message.

## 6.166.2 Constructor & Destructor Documentation

#### 6.166.2.1 Arc::Message::Message (void) `[inline]`

true if auth_ctx_ was created internally Dummy constructor

#### 6.166.2.2 Arc::Message::Message (Message & *msg*) `[inline]`

Copy constructor. Ensures shallow copy.

#### 6.166.2.3 Arc::Message::Message (long *msg_ptr_addr*)

Copy constructor. Used by language bindigs

#### 6.166.2.4 Arc::Message::∼Message (void) `[inline]`

Destructor does not affect refered objects except those created internally

## 6.166.3 Member Function Documentation

#### 6.166.3.1 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=().

#### 6.166.3.2 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=().

#### 6.166.3.3 void Arc::Message::AuthContext (MessageAuthContext ∗ *auth_ctx*) `[inline]`

Assigns auth∗ context object

#### 6.166.3.4 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=().

#### 6.166.3.5 void Arc::Message::Context (MessageContext ∗ *ctx*) `[inline]`

Assigns message context object

#### 6.166.3.6 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. 278) in a chain is connectionless like one implementing UDP protocol.

Referenced by operator=().

#### 6.166.3.7 Message& Arc::Message::operator= (Message & *msg*) `[inline]`

Assignment. Ensures shallow copy.

References Attributes(), Auth(), AuthContext(), and Context().

#### 6.166.3.8 MessagePayload∗ Arc::Message::Payload (MessagePayload ∗ *payload*) `[inline]`

Replaces payload with new one. Returns the old one.

#### 6.166.3.9 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

## 6.167 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_**

### 6.167.1 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. 290) Chain Component (**MCC** (p. 278)) 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. 278) 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. 278). 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. 278) and used by the plexer for routing the message to the appropriate service.

### 6.167.2 Constructor & Destructor Documentation

#### 6.167.2.1 Arc::MessageAttributes::MessageAttributes ()

The default constructor. This is the default constructor of the **MessageAttributes** (p. 293) class. It constructs an empty object that initially contains no attributes.

### 6.167.3 Member Function Documentation

#### 6.167.3.1 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.

#### 6.167.3.2 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.

#### 6.167.3.3 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.

#### 6.167.3.4 AttributeIterator Arc::MessageAttributes::getAll (const std::string & *key*) const

Access the value(s) of an attribute. This method returns an **AttributeIterator** (p. 73) 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. 73) for access of the values of the attribute.

#### 6.167.3.5 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.

#### 6.167.3.6 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.

#### 6.167.3.7 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.

### 6.167.4 Field Documentation

#### 6.167.4.1 AttrMap Arc::MessageAttributes::attributes_ `[protected]`

Internal storage of attributes. An AttrMap (multimap) in which all attributes (key-value pairs) are stored.

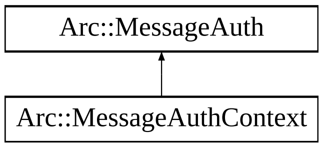
The documentation for this class was generated from the following file:

- MessageAttributes.h

## 6.168 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

### 6.168.1 Detailed Description

Contains authencity information, authorization tokens and decisions. This class only supports string keys and **SecAttr** (p. 400) values.

### 6.168.2 Member Function Documentation

#### 6.168.2.1 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. 296) tries to merge generated information to already existing like everything would be generated inside same **Export()** (p. 296) method. If does not represent valid node then new XML tree is created.

#### 6.168.2.2 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. 296) with attributes filtered. In new instance all attributes with keys listed in are removed. If is not empty only corresponding attributes are transfered to new instance. Created instance does not own refered attributes. Hence parent instance must not be deleted as long as this one is in use.

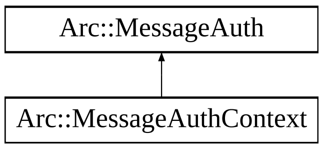
The documentation for this class was generated from the following file:

- MessageAuth.h

## 6.169 Arc::MessageAuthContext Class Reference

Handler for content of message auth∗ context.

`#include <Message.h>`Inheritance diagram for Arc::MessageAuthContext::



### 6.169.1 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. 290) object usually by first **MCC** (p. 278) in a chain and is kept as long as connection persists.

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

- Message.h

## 6.170 Arc::MessageContext Class Reference

Handler for content of message context.

```
#include <Message.h>
```

### Public Member Functions

- void **Add** (const std::string &name, **MessageContextElement** ∗element)

### 6.170.1 Detailed Description

Handler for content of message context. This class is a container for objects derived from **MessageContextElement** (p. 299). It gets associated with **Message** (p. 290) object usually by first **MCC** (p. 278) in a chain and is kept as long as connection persists.

### 6.170.2 Member Function Documentation

#### 6.170.2.1 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.

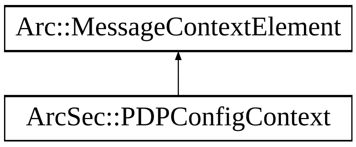
The documentation for this class was generated from the following file:

- Message.h

## 6.171 Arc::MessageContextElement Class Reference

Top class for elements contained in message context.

`#include <Message.h>`Inheritance diagram for Arc::MessageContextElement::



### 6.171.1 Detailed Description

Top class for elements contained in message context. Objects of classes inherited with this one may be stored in **MessageContext** (p. 298) container.

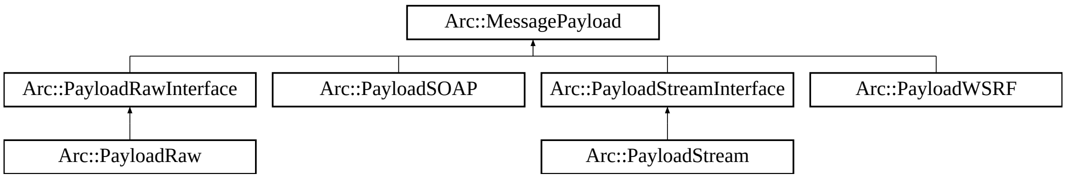
The documentation for this class was generated from the following file:

- Message.h

## 6.172 Arc::MessagePayload Class Reference

Base class for content of message passed through chain.

`#include <Message.h>`Inheritance diagram for Arc::MessagePayload::



### 6.172.1 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.

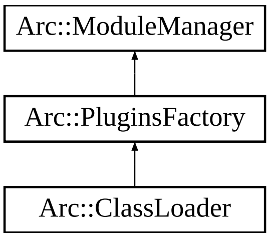
The documentation for this class was generated from the following file:

- Message.h

## 6.173 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)
- void **unload** (Glib::Module ∗module)
- void **unload** (const std::string &name)
- std::string **findLocation** (const std::string &name)
- bool **makePersistent** (Glib::Module ∗module)
- bool **makePersistent** (const std::string &name)
- void **setCfg** (**Config** ∗cfg)

### 6.173.1 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.

### 6.173.2 Constructor & Destructor Documentation

#### 6.173.2.1 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.

### 6.173.3 Member Function Documentation

#### 6.173.3.1 std::string Arc::ModuleManager::findLocation (const std::string & *name*)

Finds shared library corresponding to module 'name' and returns path to it

#### 6.173.3.2 Glib::Module∗ Arc::ModuleManager::load (const std::string & *name*, bool *probe* = `false`)

Finds module 'name' in cache or loads corresponding shared library

#### 6.173.3.3 bool Arc::ModuleManager::makePersistent (const std::string & *name*)

Make sure this module is never unloaded. Even if **unload**() (p. 302) is called.

#### 6.173.3.4 bool Arc::ModuleManager::makePersistent (Glib::Module ∗ *module*)

Make sure this module is never unloaded. Even if **unload**() (p. 302) is called.

#### 6.173.3.5 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.

#### 6.173.3.6 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** (p. 100) to adopt the singleton pattern

#### 6.173.3.7 void Arc::ModuleManager::unload (const std::string & *name*)

Unload module by its name

#### 6.173.3.8 void Arc::ModuleManager::unload (Glib::Module ∗ *module*)

Unload module by its identifier

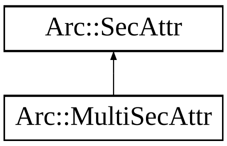
The documentation for this class was generated from the following file:

- ModuleManager.h

## 6.174 Arc::MultiSecAttr Class Reference

Container of multiple **SecAttr** (p. 400) attributes.

`#include <SecAttr.h>`Inheritance diagram for Arc::MultiSecAttr::



### Public Member Functions

- virtual **operator bool** () const
- virtual bool **Export** (**SecAttrFormat** format, **XMLNode** &val) const

### 6.174.1 Detailed Description

Container of multiple **SecAttr** (p. 400) 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.

### 6.174.2 Member Function Documentation

#### 6.174.2.1 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. 400).

#### 6.174.2.2 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.

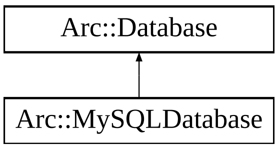
Reimplemented from **Arc::SecAttr** (p. 401).

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

- SecAttr.h

## 6.175 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** ()

### 6.175.1 Detailed Description

Implement the database accessing interface in **DBInterface.h** (p. **??**) by using mysql client library for accessing mysql database

### 6.175.2 Member Function Documentation

#### 6.175.2.1 virtual void Arc::MySQLDatabase::close () `[virtual]`

Close the connection with database server

Implements **Arc::Database** (p. 139).

#### 6.175.2.2 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. 139).

#### 6.175.2.3 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. 139).

#### 6.175.2.4 virtual bool Arc::MySQLDatabase::isconnected () const `[inline, virtual]`

Get the connection status

Implements **Arc::Database** (p. 139).

#### 6.175.2.5 virtual bool Arc::MySQLDatabase::shutdown () `[virtual]`

Ask database server to shutdown

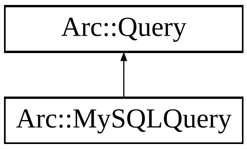
Implements **Arc::Database** (p. 139).

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

- MysqlWrapper.h

## 6.176 Arc::MySQLQuery Class Reference

Inheritance diagram for Arc::MySQLQuery::



### Public Member Functions

- virtual int **get_num_colums** ()
- virtual int **get_num_rows** ()
- virtual bool **execute** (const std::string &sqlstr)
- virtual QueryRowResult **get_row** (int row_number) const
- virtual QueryRowResult **get_row** () const
- virtual std::string **get_row_field** (int row_number, std::string &field_name)
- virtual bool **get_array** (std::string &sqlstr, QueryArrayResult &result, std::vector< std::string > &arguments)

### 6.176.1 Member Function Documentation

#### 6.176.1.1 virtual bool Arc::MySQLQuery::execute (const std::string & *sqlstr*) `[virtual]`

Execute the query

**Parameters:**

***sqlstr*** The sql sentence used to query

Implements **Arc::Query** (p. 360).

#### 6.176.1.2 virtual bool Arc::MySQLQuery::get_array (std::string & *sqlstr*, QueryArrayResult & *result*, std::vector< std::string > & *arguments*) `[virtual]`

**Query** (p. 360) the database by using some parameters into sql sentence e.g. "select table.value from table where table.name = ?"

**Parameters:**

***sqlstr*** The sql sentence with some parameters marked with "?".

***result*** The result in an array which includes all of the value in query result.

***arguments*** The argument list which should exactely correspond with the parametes in sql sentence.

Implements **Arc::Query** (p. 361).

#### 6.176.1.3 virtual int Arc::MySQLQuery::get_num_colums () `[virtual]`

Get the colum number in the query result

Implements **Arc::Query** (p. 361).

#### 6.176.1.4 virtual int Arc::MySQLQuery::get_num_rows () `[virtual]`

Get the row number in the query result

Implements **Arc::Query** (p. 361).

#### 6.176.1.5 virtual QueryRowResult Arc::MySQLQuery::get_row () const `[virtual]`

Get the value of one row in the query result, the row number will be automatically increased each time the method is called

Implements **Arc::Query** (p. 361).

#### 6.176.1.6 virtual QueryRowResult Arc::MySQLQuery::get_row (int *row_number*) const `[virtual]`

Get the value of one row in the query result

**Parameters:**

***row_number*** The number of the row

**Returns:**

A vector includes all the values in the row

Implements **Arc::Query** (p. 361).

#### 6.176.1.7 virtual std::string Arc::MySQLQuery::get_row_field (int *row_number*, std::string & *field_name*) `[virtual]`

Get the value of one specific field in one specific row

**Parameters:**

***row_number*** The row number inside the query result

***field_name*** The field name for the value which will be return

**Returns:**

The value of the specified filed in the specified row

Implements **Arc::Query** (p. 362).

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

- MysqlWrapper.h

## 6.177 Arc::NS Class Reference

### Public Member Functions

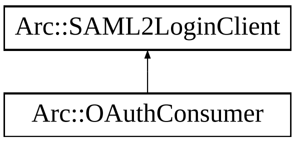
- **NS** (void)
- **NS** (const char ∗prefix, const char ∗uri)
- **NS** (const char ∗nslist[ ][2])

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

- XMLNode.h

## 6.178 Arc::OAuthConsumer Class Reference

`#include <OAuthConsumer.h>`Inheritance diagram for Arc::OAuthConsumer::



### Public Member Functions

- **OAuthConsumer** (const **MCCConfig** cfg, const **URL** url, std::list< std::string > idp_stack)
- **MCC_Status parseDN** (std::string ∗dn)
- **MCC_Status approveCSR** (const std::string approve_page)
- **MCC_Status pushCSR** (const std::string b64_pub_key, const std::string pub_key_hash, std::string ∗approve_page)
- **MCC_Status storeCert** (const std::string cert_path, const std::string auth_token, const std::string b64_dn)

### Protected Member Functions

- **MCC_Status processLogin** (const std::string username="", const std::string password="")

### 6.178.1 Detailed Description

The OAuth functionality depends on the availability of the liboauth C-bindings library

### 6.178.2 Constructor & Destructor Documentation

#### 6.178.2.1 Arc::OAuthConsumer::OAuthConsumer (const MCCConfig *cfg*, const URL *url*, std::list< std::string > *idp_stack*)

Construct an OAuth consumer with url as service provider. idp_name is currently ignored, since the idp to which the SAML2 redirect will take place is presently a hardcoded value on the SAML2 SP side. This is expected to change in the future.

### 6.178.3 Member Function Documentation

#### 6.178.3.1 MCC_Status Arc::OAuthConsumer::approveCSR (const std::string *approve_page*) `[virtual]`

Unsupported placeholder function until Confusa supports OAuth.

Implements **Arc::SAML2LoginClient** (p. 392).

#### 6.178.3.2 MCC_Status Arc::OAuthConsumer::parseDN (std::string ∗ *dn*) `[virtual]`

Unsupported placeholder function until Confusa supports OAuth.

Implements **Arc::SAML2LoginClient** (p. 392).

#### 6.178.3.3 MCC_Status Arc::OAuthConsumer::processLogin (const std::string *username* = `""`, const std::string *password* = `""`) `[protected, virtual]`

Main function performing all the OAuth login steps. Username and password will be ignored.

Implements **Arc::SAML2LoginClient** (p. 392).

#### 6.178.3.4 MCC_Status Arc::OAuthConsumer::pushCSR (const std::string *b64_pub_key*, const std::string *pub_key_hash*, std::string ∗ *approve_page*) `[virtual]`

Unsupported placeholder function until Confusa supports OAuth.

Implements **Arc::SAML2LoginClient** (p. 392).

#### 6.178.3.5 MCC_Status Arc::OAuthConsumer::storeCert (const std::string *cert_path*, const std::string *auth_token*, const std::string *b64_dn*) `[virtual]`

Unsupported placeholder function until Confusa supports OAuth.

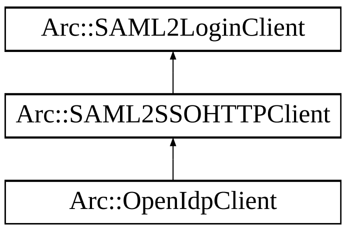
Implements **Arc::SAML2LoginClient** (p. 392).

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

- OAuthConsumer.h

## 6.179 Arc::OpenIdpClient Class Reference

Inheritance diagram for Arc::OpenIdpClient::



### Protected Member Functions

- **MCC_Status processIdPLogin** (const std::string username, const std::string password)
- **MCC_Status processConsent** ()
- **MCC_Status processIdP2Confusa** ()

### 6.179.1 Member Function Documentation

#### 6.179.1.1 MCC_Status Arc::OpenIdpClient::processConsent () `[protected, virtual]`

If the IdP has a consent module and the user has not saved her consent, this method will ask the user for consent to transmission of her data to Confusa

Implements **Arc::SAML2SSOHTTPClient** (p. 393).

#### 6.179.1.2 MCC_Status Arc::OpenIdpClient::processIdP2Confusa () `[protected, virtual]`

Redirects the user back from identity provider to the Confusa SP

Implements **Arc::SAML2SSOHTTPClient** (p. 394).

#### 6.179.1.3 MCC_Status Arc::OpenIdpClient::processIdPLogin (const std::string *username*, const std::string *password*) `[protected, virtual]`

Actual identity provider parsers for next three methods implemented in subdirectory idp/

Parse identity provider login page and submit username and password in the previsioned way

Implements **Arc::SAML2SSOHTTPClient** (p. 394).

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

- OpenIdpClient.h

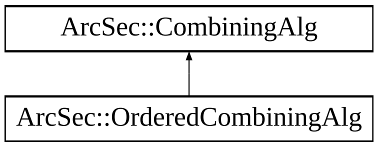
## 6.180 Arc::OptionParser Class Reference

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

- OptionParser.h

## 6.181 ArcSec::OrderedCombiningAlg Class Reference

Inheritance diagram for ArcSec::OrderedCombiningAlg::



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

- OrderedAlg.h

## 6.182 passwd Struct Reference

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

- win32.h

## 6.183 Arc::PathIterator Class Reference

Class to iterate through elements of path.

`#include <URL.h>`

### Public Member Functions

- **PathIterator** (const std::string &path, bool end=false)
- **PathIterator** & **operator++** ()
- **PathIterator** & **operator--** ()
- **operator bool** () const
- std::string **operator∗** () const
- std::string **Rest** () const

### 6.183.1 Detailed Description

Class to iterate through elements of path.

### 6.183.2 Constructor & Destructor Documentation

#### 6.183.2.1 Arc::PathIterator::PathIterator (const std::string & *path*, bool *end* = `false`)

Constructor accepts path and stores it internally. If end is set to false iterator is pointing at first element in path. Otherwise selected element is one before last.

### 6.183.3 Member Function Documentation

#### 6.183.3.1 Arc::PathIterator::operator bool () const

Return false when iterator moved outside path elements

#### 6.183.3.2 std::string Arc::PathIterator::operator∗ () const

Returns part of initial path from first till and including current

#### 6.183.3.3 PathIterator& Arc::PathIterator::operator++ ()

Advances iterator to point at next path element

#### 6.183.3.4 PathIterator& Arc::PathIterator::operator-- ()

Moves iterator to element before current

#### 6.183.3.5 std::string Arc::PathIterator::Rest () const

Returns part of initial path from one after current till end

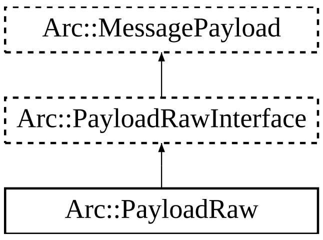
The documentation for this class was generated from the following file:

- **URL.h**

## 6.184 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[]** (Size_t pos) const
- virtual char ∗ **Content** (Size_t pos=-1)
- virtual Size_t **Size** (void) const
- virtual char ∗ **Insert** (Size_t pos=0, Size_t size=0)
- virtual char ∗ **Insert** (const char ∗s, Size_t pos=0, Size_t size=-1)
- virtual char ∗ **Buffer** (unsigned int num=0)
- virtual Size_t **BufferSize** (unsigned int num=0) const
- virtual Size_t **BufferPos** (unsigned int num=0) const
- virtual bool **Truncate** (Size_t size)

### 6.184.1 Detailed Description

Raw byte multi-buffer. This is implementation of **PayloadRawInterface** (p. 321). Buffers are memory blocks logically placed one after another.

### 6.184.2 Constructor & Destructor Documentation

#### 6.184.2.1 Arc::PayloadRaw::PayloadRaw (void) `[inline]`

List of handled buffers. Constructor. Created object contains no buffers.

#### 6.184.2.2 virtual Arc::PayloadRaw::∼PayloadRaw (void) `[virtual]`

Destructor. Frees allocated buffers.

### 6.184.3 Member Function Documentation

#### 6.184.3.1 virtual char∗ Arc::PayloadRaw::Buffer (unsigned int *num* = 0) `[virtual]`

Returns pointer to num'th buffer

Implements **Arc::PayloadRawInterface** (p. 321).

#### 6.184.3.2 virtual Size_t Arc::PayloadRaw::BufferPos (unsigned int *num* = 0) const [virtual]

Returns position of num'th buffer

Implements **Arc::PayloadRawInterface** (p. 321).

#### 6.184.3.3 virtual Size_t Arc::PayloadRaw::BufferSize (unsigned int *num* = 0) const [virtual]

Returns length of num'th buffer

Implements **Arc::PayloadRawInterface** (p. 322).

#### 6.184.3.4 virtual char* Arc::PayloadRaw::Content (Size_t *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. 322).

#### 6.184.3.5 virtual char* Arc::PayloadRaw::Insert (const char * *s*, Size_t *pos* = 0, Size_t *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. 322).

#### 6.184.3.6 virtual char* Arc::PayloadRaw::Insert (Size_t *pos* = 0, Size_t *size* = 0) [virtual]

Create new buffer at global position 'pos' of size 'size'.

Implements **Arc::PayloadRawInterface** (p. 322).

#### 6.184.3.7 virtual char Arc::PayloadRaw::operator[] (Size_t *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. 322).

#### 6.184.3.8 virtual Size_t Arc::PayloadRaw::Size (void) const [virtual]

Returns logical size of whole structure.

Implements **Arc::PayloadRawInterface** (p. 322).

#### 6.184.3.9 virtual bool Arc::PayloadRaw::Truncate (Size_t *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. 322).

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

- PayloadRaw.h

## 6.185 Arc::PayloadRawBuf Struct Reference

### Data Fields

- int **size**
- int **length**
- bool **allocated**

### 6.185.1 Field Documentation

#### 6.185.1.1 bool Arc::PayloadRawBuf::allocated

size of used memory - size of buffer

#### 6.185.1.2 int Arc::PayloadRawBuf::length

size of allocated memory

#### 6.185.1.3 int Arc::PayloadRawBuf::size

pointer to buffer in memory

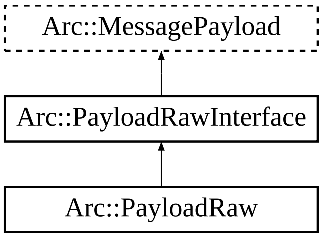
The documentation for this struct was generated from the following file:

- PayloadRaw.h

## 6.186 Arc::PayloadRawInterface Class Reference

Random Access Payload for **Message** (p. 290) objects.

`#include <PayloadRaw.h>`Inheritance diagram for Arc::PayloadRawInterface::



### Public Member Functions

- virtual char **operator[ ]** (Size_t pos) const =0
- virtual char ∗ **Content** (Size_t pos=-1)=0
- virtual Size_t **Size** (void) const =0
- virtual char ∗ **Insert** (Size_t pos=0, Size_t size=0)=0
- virtual char ∗ **Insert** (const char ∗s, Size_t pos=0, Size_t size=-1)=0
- virtual char ∗ **Buffer** (unsigned int num)=0
- virtual Size_t **BufferSize** (unsigned int num) const =0
- virtual Size_t **BufferPos** (unsigned int num) const =0
- virtual bool **Truncate** (Size_t size)=0

### 6.186.1 Detailed Description

Random Access Payload for **Message** (p. 290) objects. This class is a virtual interface for managing **Message** (p. 290) 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.

### 6.186.2 Member Function Documentation

#### 6.186.2.1 virtual char∗ Arc::PayloadRawInterface::Buffer (unsigned int *num*) `[pure virtual]`

Returns pointer to num'th buffer

Implemented in **Arc::PayloadRaw** (p. 317).

#### 6.186.2.2 virtual Size_t Arc::PayloadRawInterface::BufferPos (unsigned int *num*) const `[pure virtual]`

Returns position of num'th buffer

Implemented in **Arc::PayloadRaw** (p. 318).

#### 6.186.2.3 virtual Size_t Arc::PayloadRawInterface::BufferSize (unsigned int *num*) const `[pure virtual]`

Returns length of num'th buffer

Implemented in **Arc::PayloadRaw** (p. 318).

#### 6.186.2.4 virtual char∗ Arc::PayloadRawInterface::Content (Size_t *pos* = `-1`) `[pure virtual]`

Get pointer to buffer content at global position 'pos'. By default to beginning of main buffer whatever that means.

Implemented in **Arc::PayloadRaw** (p. 318).

#### 6.186.2.5 virtual char∗ Arc::PayloadRawInterface::Insert (const char ∗ *s*, Size_t *pos* = `0`, Size_t *size* = `-1`) `[pure virtual]`

Create new buffer at global position 'pos' of size 'size'. Created buffer is filled with content of memory at 's'. If 'size' is negative content at 's' is expected to be null-terminated.

Implemented in **Arc::PayloadRaw** (p. 318).

#### 6.186.2.6 virtual char∗ Arc::PayloadRawInterface::Insert (Size_t *pos* = `0`, Size_t *size* = `0`) `[pure virtual]`

Create new buffer at global position 'pos' of size 'size'.

Implemented in **Arc::PayloadRaw** (p. 318).

#### 6.186.2.7 virtual char Arc::PayloadRawInterface::operator[ ] (Size_t *pos*) const `[pure virtual]`

Returns content of byte at specified position. Specified position 'pos' is treated as global one and goes through all buffers placed one after another.

Implemented in **Arc::PayloadRaw** (p. 318).

#### 6.186.2.8 virtual Size_t Arc::PayloadRawInterface::Size (void) const `[pure virtual]`

Returns logical size of whole structure.

Implemented in **Arc::PayloadRaw** (p. 318).

#### 6.186.2.9 virtual bool Arc::PayloadRawInterface::Truncate (Size_t *size*) `[pure virtual]`

Change size of stored information. If size exceeds end of allocated buffer, buffers are not re-allocated, only logical size is extended. Buffers with location behind new size are deallocated.

Implemented in **Arc::PayloadRaw** (p. 318).

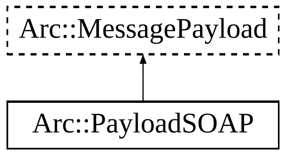
The documentation for this class was generated from the following file:

- PayloadRaw.h

## 6.187 Arc::PayloadSOAP Class Reference

Payload of **Message** (p. 290) 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)

### 6.187.1 Detailed Description

Payload of **Message** (p. 290) with SOAP content. This class combines **MessagePayload** (p. 300) with SOAPEnvelope to make it possible to pass SOAP messages through **MCC** (p. 278) chain.

### 6.187.2 Constructor & Destructor Documentation

#### 6.187.2.1 Arc::PayloadSOAP::PayloadSOAP (const NS & *ns*, bool *fault* = `false`)

Constructor - creates new **Message** (p. 290) payload

#### 6.187.2.2 Arc::PayloadSOAP::PayloadSOAP (const SOAPEnvelope & *soap*)

Constructor - creates **Message** (p. 290) payload from SOAP document. Provided SOAP document is copied to new object.

#### 6.187.2.3 Arc::PayloadSOAP::PayloadSOAP (const MessagePayload & *source*)

Constructor - creates SOAP message from payload. **PayloadRawInterface** (p. 321) and derived classes are supported.

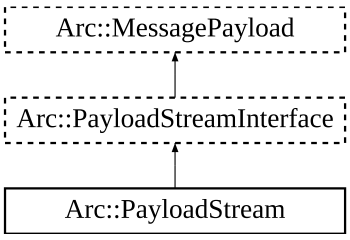
The documentation for this class was generated from the following file:

- PayloadSOAP.h

## 6.188 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, Size_t 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 Size_t **Pos** (void) const
- virtual Size_t **Size** (void) const
- virtual Size_t **Limit** (void) const

### Protected Attributes

- int **handle_**
- bool **seekable_**

### 6.188.1 Detailed Description

POSIX handle as Payload. This is an implemetation of **PayloadStreamInterface** (p. 328) for generic POSIX handle.

### 6.188.2 Constructor & Destructor Documentation

#### 6.188.2.1 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.

#### 6.188.2.2 virtual Arc::PayloadStream::∼PayloadStream (void) `[inline, virtual]`

Destructor.

### 6.188.3 Member Function Documentation

#### 6.188.3.1 virtual std::string Arc::PayloadStream::Get (void) `[inline, virtual]`

Read as many as possible (sane amount) of bytes.

Implements **Arc::PayloadStreamInterface** (p. 328).

References Get().

Referenced by Get().

#### 6.188.3.2 virtual bool Arc::PayloadStream::Get (std::string & *buf*) `[virtual]`

Read as many as possible (sane amount) of bytes into buf.

Implements **Arc::PayloadStreamInterface** (p. 328).

#### 6.188.3.3 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. 329).

#### 6.188.3.4 virtual Size_t Arc::PayloadStream::Limit (void) const `[inline, virtual]`

Returns position at which stream reading will stop if supported. That may be not same as **Size()** (p. 326) if instance is meant to provide access to only part of underlying obejct.

Implements **Arc::PayloadStreamInterface** (p. 329).

#### 6.188.3.5 virtual Arc::PayloadStream::operator bool (void) `[inline, virtual]`

Returns true if stream is valid.

Implements **Arc::PayloadStreamInterface** (p. 329).

References handle_.

#### 6.188.3.6 virtual bool Arc::PayloadStream::operator! (void) `[inline, virtual]`

Returns true if stream is invalid.

Implements **Arc::PayloadStreamInterface** (p. 329).

References handle_.

#### 6.188.3.7 virtual Size_t Arc::PayloadStream::Pos (void) const `[inline, virtual]`

Returns current position in stream if supported.

Implements **Arc::PayloadStreamInterface** (p. 329).

#### 6.188.3.8 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. 329).

References Put().

Referenced by Put().

#### 6.188.3.9 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. 329).

References Put().

Referenced by Put().

#### 6.188.3.10 virtual bool Arc::PayloadStream::Put (const char ∗ *buf*, Size_t *size*) `[virtual]`

Push 'size' bytes from 'buf' into stream. Returns true on success.

Implements **Arc::PayloadStreamInterface** (p. 329).

#### 6.188.3.11 virtual Size_t Arc::PayloadStream::Size (void) const `[inline, virtual]`

Returns size of underlying object if supported.

Implements **Arc::PayloadStreamInterface** (p. 330).

#### 6.188.3.12 virtual void Arc::PayloadStream::Timeout (int *to*) `[inline, virtual]`

Set current timeout for **Get()** (p. 325) and **Put()** (p. 326) operations.

Implements **Arc::PayloadStreamInterface** (p. 330).

#### 6.188.3.13 virtual int Arc::PayloadStream::Timeout (void) const `[inline, virtual]`

**Query** (p. 360) current timeout for **Get()** (p. 325) and **Put()** (p. 326) operations.

Implements **Arc::PayloadStreamInterface** (p. 330).

### 6.188.4 Field Documentation

#### 6.188.4.1 int Arc::PayloadStream::handle_ `[protected]`

Timeout for read/write operations

Referenced by operator bool(), and operator!().

#### 6.188.4.2 bool Arc::PayloadStream::seekable_ `[protected]`

Handle for operations

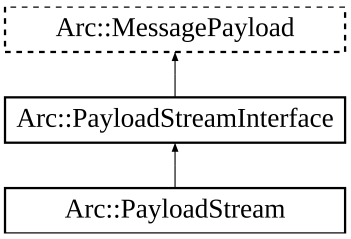
The documentation for this class was generated from the following file:

- PayloadStream.h

## 6.189 Arc::PayloadStreamInterface Class Reference

Stream-like Payload for **Message** (p. 290) object.

`#include <PayloadStream.h>`Inheritance diagram for Arc::PayloadStreamInterface::



### Public Member Functions

- virtual bool **Get** (char ∗buf, int &size)=0
- virtual bool **Get** (std::string &buf)=0
- virtual std::string **Get** (void)=0
- virtual bool **Put** (const char ∗buf, Size_t size)=0
- virtual bool **Put** (const std::string &buf)=0
- virtual bool **Put** (const char ∗buf)=0
- virtual **operator bool** (void)=0
- virtual bool **operator!** (void)=0
- virtual int **Timeout** (void) const =0
- virtual void **Timeout** (int to)=0
- virtual Size_t **Pos** (void) const =0
- virtual Size_t **Size** (void) const =0
- virtual Size_t **Limit** (void) const =0

### 6.189.1 Detailed Description

Stream-like Payload for **Message** (p. 290) object. This class is a virtual interface for managing stream-like source and destination. It's supposed to be passed through **MCC** (p. 278) chain as payload of **Message** (p. 290). It must be treated by MCCs and Services as dynamic payload. This class is purely virtual.

### 6.189.2 Member Function Documentation

#### 6.189.2.1 virtual std::string Arc::PayloadStreamInterface::Get (void) `[pure virtual]`

Read as many as possible (sane amount) of bytes.

Implemented in **Arc::PayloadStream** (p. 325).

#### 6.189.2.2 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. 325).

#### 6.189.2.3 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. 325).

#### 6.189.2.4 virtual Size_t Arc::PayloadStreamInterface::Limit (void) const `[pure virtual]`

Returns position at which stream reading will stop if supported. That may be not same as **Size()** (p. 330) if instance is meant to provide access to only part of underlying obejct.

Implemented in **Arc::PayloadStream** (p. 325).

#### 6.189.2.5 virtual Arc::PayloadStreamInterface::operator bool (void) `[pure virtual]`

Returns true if stream is valid.

Implemented in **Arc::PayloadStream** (p. 325).

#### 6.189.2.6 virtual bool Arc::PayloadStreamInterface::operator! (void) `[pure virtual]`

Returns true if stream is invalid.

Implemented in **Arc::PayloadStream** (p. 325).

#### 6.189.2.7 virtual Size_t Arc::PayloadStreamInterface::Pos (void) const `[pure virtual]`

Returns current position in stream if supported.

Implemented in **Arc::PayloadStream** (p. 326).

#### 6.189.2.8 virtual bool Arc::PayloadStreamInterface::Put (const char ∗ *buf*) `[pure virtual]`

Push null terminated information from 'buf' into stream. Returns true on success.

Implemented in **Arc::PayloadStream** (p. 326).

#### 6.189.2.9 virtual bool Arc::PayloadStreamInterface::Put (const std::string & *buf*) `[pure virtual]`

Push information from 'buf' into stream. Returns true on success.

Implemented in **Arc::PayloadStream** (p. 326).

#### 6.189.2.10 virtual bool Arc::PayloadStreamInterface::Put (const char ∗ *buf*, Size_t *size*) `[pure virtual]`

Push 'size' bytes from 'buf' into stream. Returns true on success.

Implemented in **Arc::PayloadStream** (p. 326).

#### 6.189.2.11 virtual Size_t Arc::PayloadStreamInterface::Size (void) const `[pure virtual]`

Returns size of underlying object if supported.

Implemented in **Arc::PayloadStream** (p. 326).

#### 6.189.2.12 virtual void Arc::PayloadStreamInterface::Timeout (int *to*) `[pure virtual]`

Set current timeout for **Get()** (p. 328) and **Put()** (p. 329) operations.

Implemented in **Arc::PayloadStream** (p. 326).

#### 6.189.2.13 virtual int Arc::PayloadStreamInterface::Timeout (void) const `[pure virtual]`

**Query** (p. 360) current timeout for **Get()** (p. 328) and **Put()** (p. 329) operations.

Implemented in **Arc::PayloadStream** (p. 326).

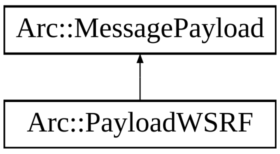
The documentation for this class was generated from the following file:

- PayloadStream.h

## 6.190 Arc::PayloadWSRF Class Reference

This class combines **MessagePayload** (p. 300) with **WSRF** (p. 503).

`#include <PayloadWSRF.h>`Inheritance diagram for Arc::PayloadWSRF::



### Public Member Functions

- **PayloadWSRF** (const SOAPEnvelope &soap)
- **PayloadWSRF** (**WSRF** &wsrp)
- **PayloadWSRF** (const **MessagePayload** &source)

### 6.190.1 Detailed Description

This class combines **MessagePayload** (p. 300) with **WSRF** (p. 503). It's intention is to make it possible to pass **WSRF** (p. 503) messages through **MCC** (p. 278) chain as one more Payload type.

### 6.190.2 Constructor & Destructor Documentation

#### 6.190.2.1 Arc::PayloadWSRF::PayloadWSRF (const SOAPEnvelope & *soap*)

Constructor - creates **Message** (p. 290) payload from SOAP message. Returns invalid **WSRF** (p. 503) if SOAP does not represent WS-ResourceProperties

#### 6.190.2.2 Arc::PayloadWSRF::PayloadWSRF (WSRF & *wsrp*)

Constructor - creates **Message** (p. 290) payload with acquired **WSRF** (p. 503) message. **WSRF** (p. 503) message will be destroyed by destructor of this object.

#### 6.190.2.3 Arc::PayloadWSRF::PayloadWSRF (const MessagePayload & *source*)

Constructor - creates **WSRF** (p. 503) message from payload. All classes derived from SOAPEnvelope are supported.

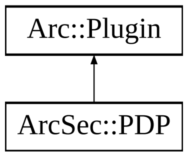
The documentation for this class was generated from the following file:

- PayloadWSRF.h

## 6.191 ArcSec::PDP Class Reference

Base class for **Policy** (p. 349) Decision Point plugins.

`#include <PDP.h>`Inheritance diagram for ArcSec::PDP::



### 6.191.1 Detailed Description

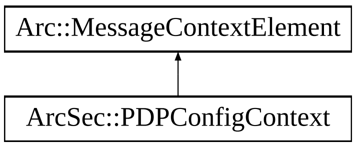
Base class for **Policy** (p. 349) Decision Point plugins. This virtual class defines method isPermitted() which processes security related information/attributes in Message and makes security decision - permit (true) or deny (false). Configuration of **PDP** (p. 332) is consumed during creation of instance through XML subtree fed to constructor.

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

- PDP.h

## 6.192 ArcSec::PDPConfigContext Class Reference

Inheritance diagram for ArcSec::PDPConfigContext::

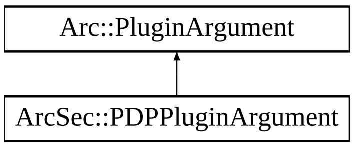


The documentation for this class was generated from the following file:

- PDP.h

## 6.193 ArcSec::PDPPluginArgument Class Reference

Inheritance diagram for ArcSec::PDPPluginArgument::



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

- PDP.h

## 6.194 Arc::Period Class Reference

### Public Member Functions

- **Period** ()
- **Period** (const time_t &)
- **Period** (const std::string &, PeriodBase base=PeriodSeconds)
- **Period** & **operator=** (const time_t &)
- **Period** & **operator=** (const **Period** &)
- void **SetPeriod** (const time_t &)
- time_t **GetPeriod** () const
- const sigc::slot< const char * > * **istr** () const
- **operator std::string** () const
- bool **operator<** (const **Period** &) const
- bool **operator>** (const **Period** &) const
- bool **operator<=** (const **Period** &) const
- bool **operator>=** (const **Period** &) const
- bool **operator==** (const **Period** &) const
- bool **operator!=** (const **Period** &) const

### 6.194.1 Constructor & Destructor Documentation

#### 6.194.1.1 Arc::Period::Period ()

Default constructor. The period is set to 0 length.

#### 6.194.1.2 Arc::Period::Period (const time_t &)

Constructor that takes a time_t variable and stores it.

#### 6.194.1.3 Arc::Period::Period (const std::string &, PeriodBase *base* = `PeriodSeconds`)

Constructor that tries to convert a string.

### 6.194.2 Member Function Documentation

#### 6.194.2.1 time_t Arc::Period::GetPeriod () const

gets the period

#### 6.194.2.2 const sigc::slot<const char*>* Arc::Period::istr () const

For use with **IString** (p. 249)

#### 6.194.2.3 Arc::Period::operator std::string () const

Returns a string representation of the period.

#### 6.194.2.4 bool Arc::Period::operator!= (const Period &) const

Comparing two **Period** (p. 335) objects.

#### 6.194.2.5 bool Arc::Period::operator< (const Period &) const

Comparing two **Period** (p. 335) objects.

#### 6.194.2.6 bool Arc::Period::operator<= (const Period &) const

Comparing two **Period** (p. 335) objects.

#### 6.194.2.7 Period& Arc::Period::operator= (const Period &)

Assignment operator from a **Period** (p. 335).

#### 6.194.2.8 Period& Arc::Period::operator= (const time_t &)

Assignment operator from a time_t.

#### 6.194.2.9 bool Arc::Period::operator== (const Period &) const

Comparing two **Period** (p. 335) objects.

#### 6.194.2.10 bool Arc::Period::operator> (const Period &) const

Comparing two **Period** (p. 335) objects.

#### 6.194.2.11 bool Arc::Period::operator>= (const Period &) const

Comparing two **Period** (p. 335) objects.

#### 6.194.2.12 void Arc::Period::SetPeriod (const time_t &)

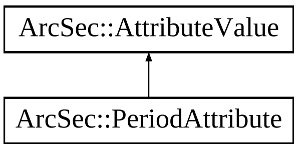
sets the period

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

- DateTime.h

## 6.195 ArcSec::PeriodAttribute Class Reference

`#include <DateTimeAttribute.h>`Inheritance diagram for ArcSec::PeriodAttribute::



### Public Member Functions

- virtual std::string **encode** ()
- virtual std::string **getType** ()
- virtual std::string **getId** ()

### 6.195.1 Detailed Description

Formate: datetime"/"duration datetime"/"datetime duration"/"datetime

### 6.195.2 Member Function Documentation

#### 6.195.2.1 virtual std::string ArcSec::PeriodAttribute::encode () `[virtual]`

encode the value in a string format

Implements **ArcSec::AttributeValue** (p. 78).

#### 6.195.2.2 virtual std::string ArcSec::PeriodAttribute::getId () `[inline, virtual]`

Get the AttributeId of the <Attribute>

Implements **ArcSec::AttributeValue** (p. 78).

#### 6.195.2.3 virtual std::string ArcSec::PeriodAttribute::getType () `[inline, virtual]`

Get the DataType of the <Attribute>

Implements **ArcSec::AttributeValue** (p. 78).

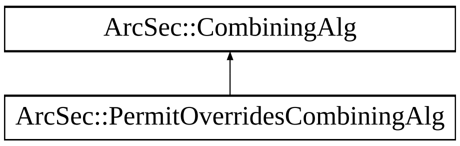
The documentation for this class was generated from the following file:

- DateTimeAttribute.h

## 6.196 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

### 6.196.1 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".

### 6.196.2 Member Function Documentation

#### 6.196.2.1 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. 111).

#### 6.196.2.2 virtual const std::string& ArcSec::PermitOverridesCombiningAlg::getalgId (void) const `[inline, virtual]`

Get the identifier

Implements **ArcSec::CombiningAlg** (p. 111).

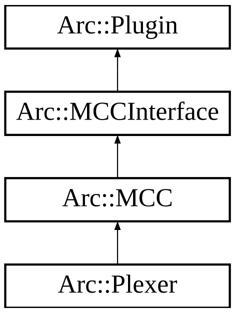
The documentation for this class was generated from the following file:

- PermitOverridesAlg.h

## 6.197 Arc::Plexer Class Reference

The **Plexer** (p. 340) 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**

### 6.197.1 Detailed Description

The **Plexer** (p. 340) class, used for routing messages to services. This is the **Plexer** (p. 340) class. Its purpose is to route incoming messages to appropriate Services and **MCC** (p. 278) chains.

### 6.197.2 Constructor & Destructor Documentation

#### 6.197.2.1 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.

#### 6.197.2.2 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.

## 6.197.3 Member Function Documentation

### 6.197.3.1 virtual void Arc::Plexer::Next (MCCInterface $*$ *next*, const std::string & *label*) `[virtual]`

Add reference to next **MCC** (p. 278) in chain. This method is called by **Loader** (p. 264) for every potentially labeled link to next component which implements **MCCInterface** (p. 284). If next is set NULL corresponding link is removed.

Reimplemented from **Arc::MCC** (p. 279).

### 6.197.3.2 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. 279).

## 6.197.4 Field Documentation

### 6.197.4.1 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. 280).

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

- Plexer.h

## 6.198 Arc::PlexerEntry Class Reference

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

```
#include <Plexer.h>
```

### 6.198.1 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.

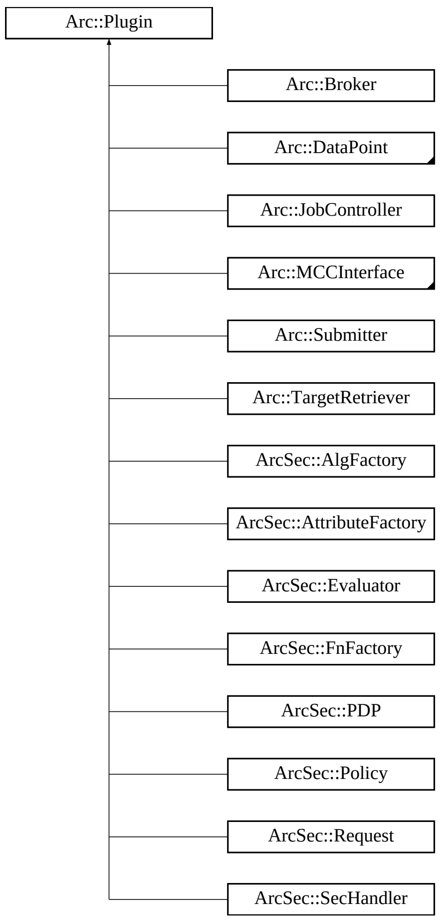
The documentation for this class was generated from the following file:

- Plexer.h

## 6.199 Arc::Plugin Class Reference

Base class for loadable ARC components.

`#include <Plugin.h>`Inheritance diagram for Arc::Plugin::



### 6.199.1 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.

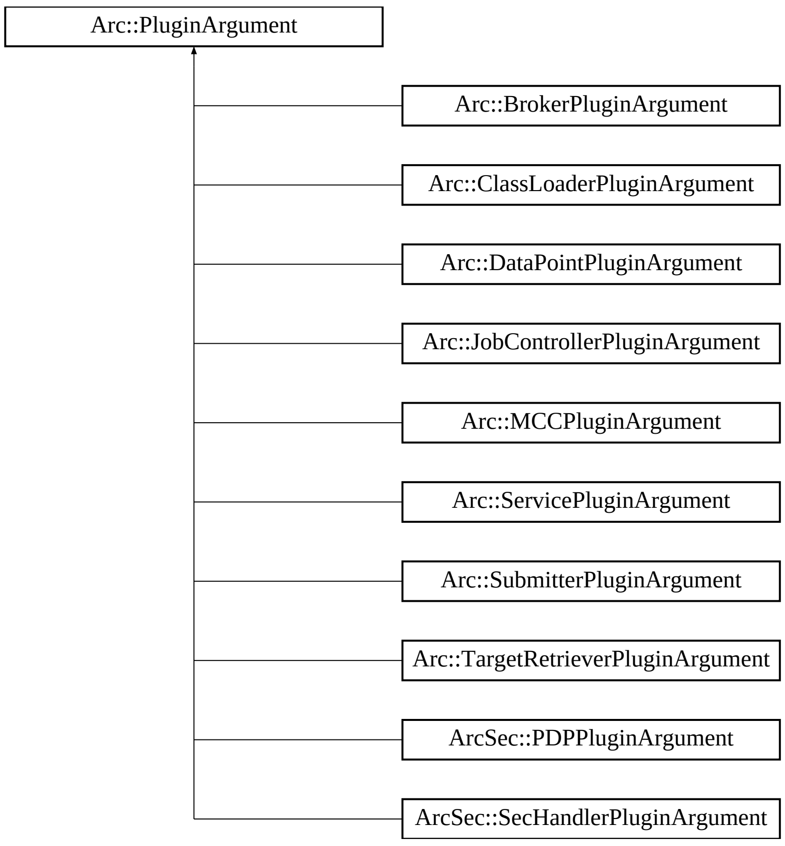
The documentation for this class was generated from the following file:

- Plugin.h

## 6.200 Arc::PluginArgument Class Reference

Base class for passing arguments to loadable ARC components.

`#include <Plugin.h>`Inheritance diagram for Arc::PluginArgument::


Arc::PluginArgument
Arc::BrokerPluginArgument
Arc::ClassLoaderPluginArgument
Arc::DataPointPluginArgument
Arc::JobControllerPluginArgument
Arc::MCCPluginArgument
Arc::ServicePluginArgument
Arc::SubmitterPluginArgument
Arc::TargetRetrieverPluginArgument
ArcSec::PDPPluginArgument
ArcSec::SecHandlerPluginArgument

### Public Member Functions

- **PluginsFactory** ∗ **get_factory** (void)
- Glib::Module ∗ **get_module** (void)

### 6.200.1 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.

### 6.200.2 Member Function Documentation

#### 6.200.2.1 PluginsFactory∗ Arc::PluginArgument::get_factory (void)

Returns pointer to factory which instantiated plugin. Because factory usually destroys/unloads plugins in its destructor it should be safe to keep this pointer inside plugin for later use. But one must always check.

#### 6.200.2.2 Glib::Module∗ Arc::PluginArgument::get_module (void)

Returns pointer to loadable module/library which contains plugin. Corresponding factory keeps list of modules till itself is destroyed. So it should be safe to keep that pointer. But care must be taken if module contains persistent plugins. Such modules stay in memory after factory is detroyed. So it is advisable to use obtained pointer only in constructor function of plugin.

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

- Plugin.h

## 6.201 Arc::PluginDescriptor Struct Reference

Description of ARC lodable component.

```
#include <Plugin.h>
```

### 6.201.1 Detailed Description

Description of ARC lodable component.

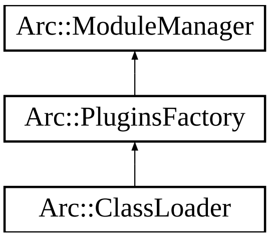
The documentation for this struct was generated from the following file:

- Plugin.h

## 6.202 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 search=true)
- bool **load** (const std::string &name)

### 6.202.1 Detailed Description

Generic ARC plugins loader. The instance of this class provides functionality of loading pluggable ARC components stored in shared libraries. For more information please check HED documentation. This class is thread-safe - its methods are proceted from simultatneous use form multiple threads. Current thread protection implementation is suboptimal and will be revised in future.

### 6.202.2 Constructor & Destructor Documentation

#### 6.202.2.1 Arc::PluginsFactory::PluginsFactory (const Config & *cfg*)

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

### 6.202.3 Member Function Documentation

#### 6.202.3.1 Plugin∗ Arc::PluginsFactory::get_instance (const std::string & *kind*, PluginArgument ∗ *arg*, bool *search* = `true`)

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. 347) class. If serach is set to false then no attempt is made to find plugins in shared libraries. Only plugins already loaded with previous calls to **get_instance()** (p. 347) and **load()** (p. 348) are checked. Returns created instance.

#### 6.202.3.2 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. 347) class. Returns true if library was loaded.

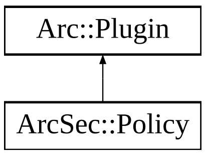
The documentation for this class was generated from the following file:

- Plugin.h

## 6.203 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

### 6.203.1 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.

### 6.203.2 Constructor & Destructor Documentation

#### 6.203.2.1 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.

#### 6.203.2.2 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. 203) which includes the factory objects for combining algorithm and function

### 6.203.3 Member Function Documentation

#### 6.203.3.1 virtual void ArcSec::Policy::addPolicy (Policy ∗ *pl*) `[inline, virtual]`

Add a policy element to into "this" object

#### 6.203.3.2 virtual Result ArcSec::Policy::eval (EvaluationCtx ∗) `[pure virtual]`

Evaluate policy For the <Rule> of **Arc** (p. 23), only get the "Effect" from rules; For the <Policy> of **Arc** (p. 23), 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>

#### 6.203.3.3 virtual std::string ArcSec::Policy::getEffect () const `[pure virtual]`

Get the "Effect" attribute

#### 6.203.3.4 virtual const char∗ ArcSec::Policy::getEvalName () const `[pure virtual]`

Get the name of **Evaluator** (p. 200) which can evaluate this policy

#### 6.203.3.5 virtual EvalResult& ArcSec::Policy::getEvalResult () `[pure virtual]`

Get eveluation result

#### 6.203.3.6 virtual const char∗ ArcSec::Policy::getName () const `[pure virtual]`

Get the name of this policy

#### 6.203.3.7 virtual void ArcSec::Policy::make_policy () `[inline, virtual]`

Parse XMLNode, and construct the low-level Rule object

#### 6.203.3.8 virtual void ArcSec::Policy::setEvalResult (EvalResult & *res*) `[pure virtual]`

Set eveluation result

#### 6.203.3.9 virtual void ArcSec::Policy::setEvaluatorContext (EvaluatorContext ∗) `[inline, virtual]`

Set **Evaluator** (p. 200) Context for the usage in creating low-level policy object

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

- Policy.h

## 6.204 ArcSec::PolicyStore::PolicyElement Class Reference

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

- PolicyStore.h

## 6.205 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)

### 6.205.1 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.).

### 6.205.2 Member Function Documentation

#### 6.205.2.1 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. 203) which includes the ∗∗Factory

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

- PolicyParser.h

## 6.206 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)

### 6.206.1 Detailed Description

Storage place for policy objects.

### 6.206.2 Constructor & Destructor Documentation

#### 6.206.2.1 ArcSec::PolicyStore::PolicyStore (const std::string & *alg*, const std::string & *policyclassname*, EvaluatorContext ∗ *ctx*)

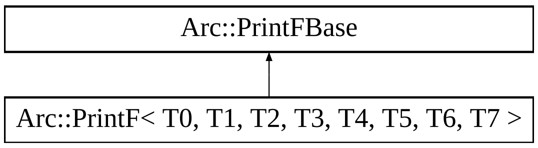
Creates policy store with specified combing algorithm (alg - not used yet), policy name (policyclassname) and context (ctx)

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

- PolicyStore.h

## 6.207 Arc::PrintF< T0, T1, T2, T3, T4, T5, T6, T7 > Class Template Reference

Inheritance diagram for Arc::PrintF< T0, T1, T2, T3, T4, T5, T6, T7 >::



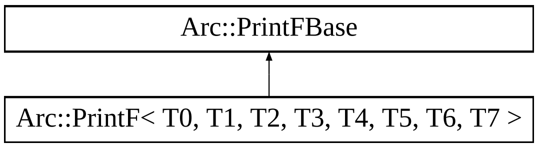
**template<class T0 = int, class T1 = int, class T2 = int, class T3 = int, class T4 = int, class T5 = int, class T6 = int, class T7 = int> class Arc::PrintF< T0, T1, T2, T3, T4, T5, T6, T7 >**

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

- IString.h

## 6.208 Arc::PrintFBase Class Reference

Inheritance diagram for Arc::PrintFBase::

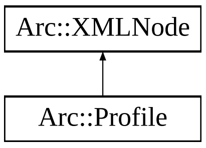


The documentation for this class was generated from the following file:

- IString.h

## 6.209 Arc::Profile Class Reference

Inheritance diagram for Arc::Profile::



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

- Profile.h

## 6.210 ArcCredential::PROXYCERTINFO_st Struct Reference

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

- Proxycertinfo.h

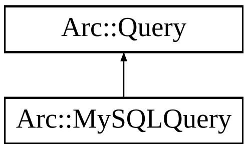
## 6.211 ArcCredential::PROXYPOLICY_st Struct Reference

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

- Proxycertinfo.h

## 6.212 Arc::Query Class Reference

Inheritance diagram for Arc::Query::



### Public Member Functions

- **Query** ()
- **Query** (**Database** ∗db)
- virtual ∼**Query** ()
- virtual int **get_num_colums** ()=0
- virtual int **get_num_rows** ()=0
- virtual bool **execute** (const std::string &sqlstr)=0
- virtual QueryRowResult **get_row** (int row_number) const =0
- virtual QueryRowResult **get_row** () const =0
- virtual std::string **get_row_field** (int row_number, std::string &field_name)=0
- virtual bool **get_array** (std::string &sqlstr, QueryArrayResult &result, std::vector< std::string > &arguments)=0

### 6.212.1 Constructor & Destructor Documentation

#### 6.212.1.1 Arc::Query::Query () `[inline]`

Default constructor

#### 6.212.1.2 Arc::Query::Query (Database ∗ *db*) `[inline]`

Constructor

**Parameters:**

***db*** The database object which will be used by **Query** (p. 360) class to get the database connection

#### 6.212.1.3 virtual Arc::Query::∼Query () `[inline, virtual]`

Deconstructor

### 6.212.2 Member Function Documentation

#### 6.212.2.1 virtual bool Arc::Query::execute (const std::string & *sqlstr*) `[pure virtual]`

Execute the query

**Parameters:**

*sqlstr* The sql sentence used to query

Implemented in **Arc::MySQLQuery** (p. 306).

#### 6.212.2.2 virtual bool Arc::Query::get_array (std::string & *sqlstr*, QueryArrayResult & *result*, std::vector< std::string > & *arguments*) `[pure virtual]`

**Query** (p. 360) the database by using some parameters into sql sentence e.g. "select table.value from table where table.name = ?"

**Parameters:**

*sqlstr* The sql sentence with some parameters marked with "?".

*result* The result in an array which includes all of the value in query result.

*arguments* The argument list which should exactely correspond with the parametes in sql sentence.

Implemented in **Arc::MySQLQuery** (p. 306).

#### 6.212.2.3 virtual int Arc::Query::get_num_colums () `[pure virtual]`

Get the colum number in the query result

Implemented in **Arc::MySQLQuery** (p. 306).

#### 6.212.2.4 virtual int Arc::Query::get_num_rows () `[pure virtual]`

Get the row number in the query result

Implemented in **Arc::MySQLQuery** (p. 307).

#### 6.212.2.5 virtual QueryRowResult Arc::Query::get_row () const `[pure virtual]`

Get the value of one row in the query result, the row number will be automatically increased each time the method is called

Implemented in **Arc::MySQLQuery** (p. 307).

#### 6.212.2.6 virtual QueryRowResult Arc::Query::get_row (int *row_number*) const `[pure virtual]`

Get the value of one row in the query result

**Parameters:**

*row_number* The number of the row

**Returns:**

A vector includes all the values in the row

Implemented in **Arc::MySQLQuery** (p. 307).

#### 6.212.2.7 virtual std::string Arc::Query::get_row_field (int *row_number*, std::string & *field_name*) `[pure virtual]`

Get the value of one specific field in one specific row

**Parameters:**

***row_number*** The row number inside the query result

***field_name*** The field name for the value which will be return

**Returns:**

The value of the specified filed in the specified row

Implemented in **Arc::MySQLQuery** (p. 307).

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

- DBInterface.h

## 6.213 Arc::Range< T > Class Template Reference

**template<class T> class Arc::Range< T >**

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

- JobDescription.h

## 6.214 Arc::Register_Info_Type Struct Reference

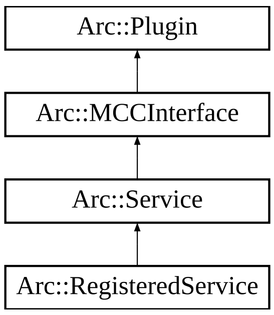
The documentation for this struct was generated from the following file:

- InfoRegister.h

## 6.215 Arc::RegisteredService Class Reference

**RegisteredService** (p. 365) - extension of **Service** (p. 409) performing self-registration.

`#include <RegisteredService.h>`Inheritance diagram for Arc::RegisteredService::



### Public Member Functions

- **RegisteredService** (**Config** ∗)

### 6.215.1 Detailed Description

**RegisteredService** (p. 365) - extension of **Service** (p. 409) performing self-registration.

### 6.215.2 Constructor & Destructor Documentation

#### 6.215.2.1 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

## 6.216 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

### 6.216.1 Detailed Description

A regular expression class. This class is a wrapper around the functions provided in regex.h.

### 6.216.2 Member Function Documentation

#### 6.216.2.1 bool Arc::RegularExpression::match (const std::string & *str*, std::list< std::string > & *unmatched*, std::list< std::string > & *matched*) const

Returns true if this regex matches the string provided. Unmatched parts of the string are stored in 'unmatched'. Matched parts of the string are stored in 'matched'.

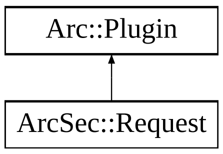
The documentation for this class was generated from the following file:

- ArcRegex.h

## 6.217 ArcSec::Request Class Reference

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

`#include <Request.h>`Inheritance diagram for ArcSec::Request::



### Public Member Functions

- virtual ReqItemList **getRequestItems** () const
- virtual void **setRequestItems** (ReqItemList sl)
- virtual void **addRequestItem** (**Attrs** &sub, **Attrs** &res, **Attrs** &act, **Attrs** &ctx)
- 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** &)

### 6.217.1 Detailed Description

Base class/Interface for request, includes a container for RequestItems and some operations. A **Request** (p. 367) object can has a few <subjects, actions, objects> tuples, i.e. **RequestItem** (p. 370) The **Request** (p. 367) 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. 367) name="arc.request" /> <......> </pdp:PDPConfig> </Service>

There can be different types of subclass which inherit **Request** (p. 367), such like XACMLRequest, ArcRequest, GACLRequest

### 6.217.2 Constructor & Destructor Documentation

#### 6.217.2.1 ArcSec::Request::Request () `[inline]`

Default constructor

#### 6.217.2.2 ArcSec::Request::Request (const Source &) `[inline]`

Constructor: Parse request information from a xml stucture in memory

## 6.217.3 Member Function Documentation

### 6.217.3.1 virtual void ArcSec::Request::addRequestItem (Attrs & *sub*, Attrs & *res*, Attrs & *act*, Attrs & *ctx*) `[inline, virtual]`

Add request tuple from non-XMLNode

### 6.217.3.2 virtual const char∗ ArcSec::Request::getEvalName () const `[pure virtual]`

Get the name of corresponding evaulator

### 6.217.3.3 virtual const char∗ ArcSec::Request::getName () const `[pure virtual]`

Get the name of this request

### 6.217.3.4 virtual ReqItemList ArcSec::Request::getRequestItems () const `[inline, virtual]`

Get all the **RequestItem** (p. 370) inside **RequestItem** (p. 370) container

### 6.217.3.5 virtual void ArcSec::Request::make_request () `[pure virtual]`

Create the objects included in **Request** (p. 367) according to the node attached to the **Request** (p. 367) object

### 6.217.3.6 virtual void ArcSec::Request::setAttributeFactory (AttributeFactory ∗ *attributefactory*) `[pure virtual]`

Set the attribute factory for the usage of **Request** (p. 367)

### 6.217.3.7 virtual void ArcSec::Request::setRequestItems (ReqItemList *sl*) `[inline, virtual]`

Set the content of the container

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

- Request.h

## 6.218 ArcSec::RequestAttribute Class Reference

Wrapper which includes **AttributeValue** (p. 77) 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** &)

### 6.218.1 Detailed Description

Wrapper which includes **AttributeValue** (p. 77) object which is generated according to date type of one spefic node in Request.xml.

### 6.218.2 Constructor & Destructor Documentation

#### 6.218.2.1 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>

### 6.218.3 Member Function Documentation

#### 6.218.3.1 RequestAttribute& ArcSec::RequestAttribute::duplicate (RequestAttribute &)

Duplicate the parameter into "this"

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

- RequestAttribute.h

## 6.219 ArcSec::RequestItem Class Reference

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

```
#include <RequestItem.h>
```

### Public Member Functions

- **RequestItem** (**Arc::XMLNode** &, **AttributeFactory** ∗)

### 6.219.1 Detailed Description

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

### 6.219.2 Constructor & Destructor Documentation

#### 6.219.2.1 ArcSec::RequestItem::RequestItem (Arc::XMLNode &, AttributeFactory ∗) `[inline]`

Constructor

**Parameters:**

***node*** The XMLNode structure of the request item

***attributefactory*** The **AttributeFactory** (p. 72) which will be used to generate **RequestAttribute** (p. 369)

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

- RequestItem.h

## 6.220 ArcSec::RequestTuple Class Reference

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

- EvaluationCtx.h

## 6.221 Arc::ResourceSlotType Class Reference

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

- JobDescription.h

## 6.222 Arc::ResourcesType Class Reference

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

- JobDescription.h

## 6.223 Arc::ResourceTargetType Class Reference

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

- JobDescription.h

## 6.224 ArcSec::Response Class Reference

Container for the evaluation results.

```
#include <Response.h>
```

### 6.224.1 Detailed Description

Container for the evaluation results.

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

- Response.h

## 6.225 ArcSec::ResponseItem Class Reference

Evaluation result concerning one **RequestTuple** (p. 371).

```
#include <Response.h>
```

### 6.225.1 Detailed Description

Evaluation result concerning one **RequestTuple** (p. 371). Include the **RequestTuple** (p. 371), related XMLNode, the set of policy objects which give positive evaluation result, and the related XMLNode

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

- Response.h

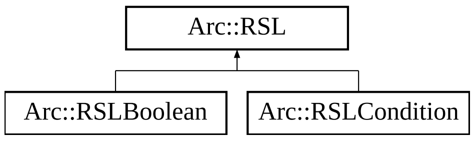
## 6.226 ArcSec::ResponseList Class Reference

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

- Response.h

## 6.227 Arc::RSL Class Reference

Inheritance diagram for Arc::RSL::

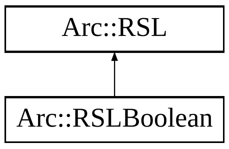


The documentation for this class was generated from the following file:

- RSLParser.h

## 6.228 Arc::RSLBoolean Class Reference

Inheritance diagram for Arc::RSLBoolean::

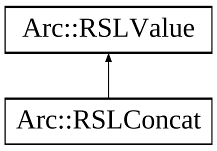


The documentation for this class was generated from the following file:

- RSLParser.h

## 6.229 Arc::RSLConcat Class Reference

Inheritance diagram for Arc::RSLConcat::

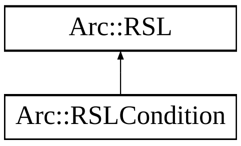


The documentation for this class was generated from the following file:

- RSLParser.h

## 6.230 Arc::RSLCondition Class Reference

Inheritance diagram for Arc::RSLCondition::

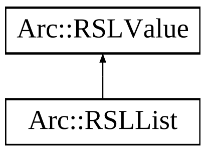


The documentation for this class was generated from the following file:

- RSLParser.h

## 6.231 Arc::RSLList Class Reference

Inheritance diagram for Arc::RSLList::

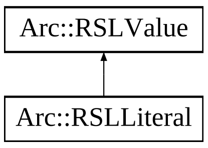


The documentation for this class was generated from the following file:

- RSLParser.h

## 6.232 Arc::RSLLiteral Class Reference

Inheritance diagram for Arc::RSLLiteral::



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

- RSLParser.h

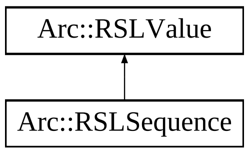
## 6.233 Arc::RSLParser Class Reference

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

- RSLParser.h

## 6.234 Arc::RSLSequence Class Reference

Inheritance diagram for Arc::RSLSequence::

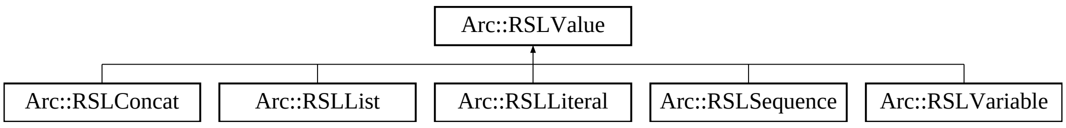


The documentation for this class was generated from the following file:

- RSLParser.h

## 6.235 Arc::RSLValue Class Reference

Inheritance diagram for Arc::RSLValue::

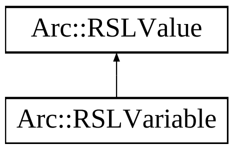


The documentation for this class was generated from the following file:

- RSLParser.h

## 6.236 Arc::RSLVariable Class Reference

Inheritance diagram for Arc::RSLVariable::



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

- RSLParser.h

## 6.237 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)
- void **Abandon** (void)

### 6.237.1 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.

### 6.237.2 Constructor & Destructor Documentation

#### 6.237.2.1 Arc::Run::Run (const std::string & *cmdline*)

Constructor preapres object to run cmdline

#### 6.237.2.2 Arc::Run::Run (const std::list< std::string > & *argv*)

Constructor preapres object to run executable and arguments specified in argv

#### 6.237.2.3 Arc::Run::∼Run (void)

Destructor kills running executable and releases associated resources

### 6.237.3 Member Function Documentation

#### 6.237.3.1 void Arc::Run::Abandon (void)

Detach this object from running process. After calling this method instance is not associated with external process anymore. As result destructor will not kill process.

#### 6.237.3.2 void Arc::Run::AssignStderr (std::string & *str*)

Associate stderr handle of executable with string. This method must be called before **Start()** (p. 390). str object must be valid as long as this object exists.

#### 6.237.3.3 void Arc::Run::AssignStdin (std::string & *str*)

Associate stdin handle of executable with string. This method must be called before **Start()** (p. 390). str object must be valid as long as this object exists.

#### 6.237.3.4 void Arc::Run::AssignStdout (std::string & *str*)

Associate stdout handle of executable with string. This method must be called before **Start()** (p. 390). str object must be valid as long as this object exists.

#### 6.237.3.5 void Arc::Run::AssignWorkingDirectory (std::string & *wd*) `[inline]`

Assign working direcotry of the running process

#### 6.237.3.6 void Arc::Run::CloseStderr (void)

Closes pipe associated with stderr handle

#### 6.237.3.7 void Arc::Run::CloseStdin (void)

Closes pipe associated with stdin handle

#### 6.237.3.8 void Arc::Run::CloseStdout (void)

Closes pipe associated with stdout handle

#### 6.237.3.9 void Arc::Run::KeepStderr (bool *keep* = `true`)

Keep stderr same as parent's if keep = true

#### 6.237.3.10 void Arc::Run::KeepStdin (bool *keep* = `true`)

Keep stdin same as parent's if keep = true

#### 6.237.3.11 void Arc::Run::KeepStdout (bool *keep* = `true`)

Keep stdout same as parent's if keep = true

#### 6.237.3.12 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

#### 6.237.3.13 Arc::Run::operator bool (void) `[inline]`

Returns true if object is valid

#### 6.237.3.14 bool Arc::Run::operator! (void) `[inline]`

Returns true if object is invalid

#### 6.237.3.15 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.

#### 6.237.3.16 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.

#### 6.237.3.17 int Arc::Run::Result (void) `[inline]`

Returns exit code of execution.

#### 6.237.3.18 bool Arc::Run::Running (void)

Return true if execution is going on.

#### 6.237.3.19 bool Arc::Run::Start (void)

Starts running executable. This method may be called only once.

#### 6.237.3.20 bool Arc::Run::Wait (void)

Wait till execution finished

#### 6.237.3.21 bool Arc::Run::Wait (int *timeout*)

Wait till execution finished or till timeout seconds expires. Returns true if execution is complete.

#### 6.237.3.22 int Arc::Run::WriteStdin (int *timeout*, const char ∗ *buf*, int *size*)

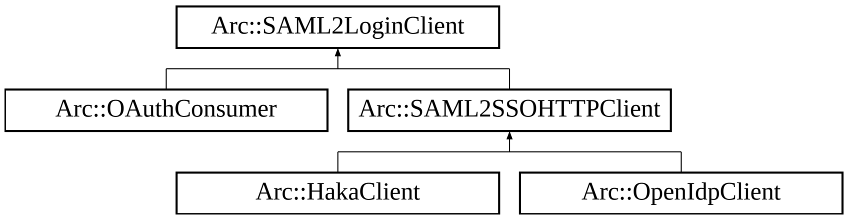
Write to stdin handle of running executable. This method may be used while stdin is directed to string. But result is unpredictable.

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

- Run.h

## 6.238 Arc::SAML2LoginClient Class Reference

Inheritance diagram for Arc::SAML2LoginClient::



### Public Member Functions

- **SAML2LoginClient** (const **MCCConfig** cfg, const **URL** url, std::list< std::string > idp_stack)
- virtual **MCC_Status processLogin** (const std::string username="", const std::string password="")=0
- **MCC_Status findSimpleSAMLInstallation** ()

### 6.238.1 Constructor & Destructor Documentation

#### 6.238.1.1 Arc::SAML2LoginClient::SAML2LoginClient (const MCCConfig *cfg*, const URL *url*, std::list< std::string > *idp_stack*)

list with the idp for nested wayf For example, Confusa can use betawayf.wayf.dk as an identity provider, which is itself only a wayf and shares the metadata with concrete service providers or even further nested wayfs. Since due to mutual authentication with metadata, we are obliged to follow the SSO redirects from WAYF to WAYF, the WAYFs are stored in a list.

### 6.238.2 Member Function Documentation

#### 6.238.2.1 MCC_Status Arc::SAML2LoginClient::findSimpleSAMLInstallation ()

find the location of the simplesamlphp installation on the SP side Will be stored in (∗sso_-pages)[SimpleSAML]

#### 6.238.2.2 virtual MCC_Status Arc::SAML2LoginClient::processLogin (const std::string *username* = "", const std::string *password* = "") `[pure virtual]`

Base interface for all login procedures

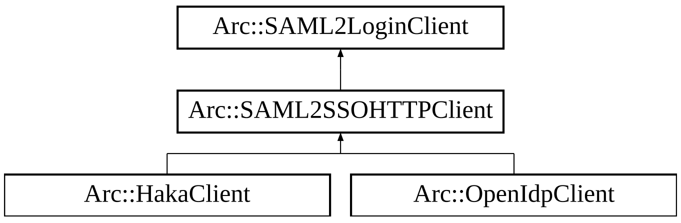
Implemented in **Arc::OAuthConsumer** (p. 310), and **Arc::SAML2SSOHTTPClient** (p. 394).

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

- SAML2LoginClient.h

## 6.239 Arc::SAML2SSOHTTPClient Class Reference

Inheritance diagram for Arc::SAML2SSOHTTPClient::



### Public Member Functions

- **MCC_Status processLogin** (const std::string username, const std::string password)
- **MCC_Status parseDN** (std::string ∗dn)
- **MCC_Status approveCSR** (const std::string approve_page)
- **MCC_Status pushCSR** (const std::string b64_pub_key, const std::string pub_key_hash, std::string ∗approve_page)
- **MCC_Status storeCert** (const std::string cert_path, const std::string auth_token, const std::string b64_dn)

### Protected Member Functions

- virtual **MCC_Status processIdPLogin** (const std::string username, const std::string password)=0
- virtual **MCC_Status processConsent** ()=0
- virtual **MCC_Status processIdP2Confusa** ()=0

### 6.239.1 Member Function Documentation

#### 6.239.1.1 MCC_Status Arc::SAML2SSOHTTPClient::approveCSR (const std::string *approve_page*) `[virtual]`

Simulate click on the approve cert signing request link

Implements **Arc::SAML2LoginClient** (p. 392).

#### 6.239.1.2 MCC_Status Arc::SAML2SSOHTTPClient::parseDN (std::string ∗ *dn*) `[virtual]`

Parse the used DN from the Confusa about_you page

Implements **Arc::SAML2LoginClient** (p. 392).

#### 6.239.1.3 virtual MCC_Status Arc::SAML2SSOHTTPClient::processConsent () `[protected, pure virtual]`

If the IdP has a consent module and the user has not saved her consent, this method will ask the user for consent to transmission of her data to Confusa

Implemented in **Arc::HakaClient** (p. 226), and **Arc::OpenIdpClient** (p. 311).

#### 6.239.1.4 virtual MCC_Status Arc::SAML2SSOHTTPClient::processIdP2Confusa () `[protected, pure virtual]`

Redirects the user back from identity provider to the Confusa SP

Implemented in **Arc::HakaClient** (p. 226), and **Arc::OpenIdpClient** (p. 311).

#### 6.239.1.5 virtual MCC_Status Arc::SAML2SSOHTTPClient::processIdPLogin (const std::string *username*, const std::string *password*) `[protected, pure virtual]`

Actual identity provider parsers for next three methods implemented in subdirectory idp/

Parse identity provider login page and submit username and password in the previsioned way

Implemented in **Arc::HakaClient** (p. 226), and **Arc::OpenIdpClient** (p. 311).

#### 6.239.1.6 MCC_Status Arc::SAML2SSOHTTPClient::processLogin (const std::string *username*, const std::string *password*) `[virtual]`

Models complete SAML2 WebSSO authN flow with start -> WAYF -> Login -> (consent) -> start

Implements **Arc::SAML2LoginClient** (p. 392).

#### 6.239.1.7 MCC_Status Arc::SAML2SSOHTTPClient::pushCSR (const std::string *b64_pub_key*, const std::string *pub_key_hash*, std::string ∗ *approve_page*) `[virtual]`

Send the cert signing request to Confusa for signing

Implements **Arc::SAML2LoginClient** (p. 392).

#### 6.239.1.8 MCC_Status Arc::SAML2SSOHTTPClient::storeCert (const std::string *cert_path*, const std::string *auth_token*, const std::string *b64_dn*) `[virtual]`

Download the signed certificate from Confusa and store it locally

Implements **Arc::SAML2LoginClient** (p. 392).

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

- SAML2LoginClient.h

## 6.240 Arc::SAMLToken Class Reference

Class for manipulating SAML Token **Profile** (p. 357).

```
#include <SAMLToken.h>
```

### Public Types

- enum **SAMLVersion**

### Public Member Functions

- **SAMLToken** (SOAPEnvelope &soap)
- **SAMLToken** (SOAPEnvelope &soap, const std::string &certfile, const std::string &keyfile, **SAMLVersion** saml_version=SAML2, **XMLNode** saml_assertion=**XMLNode**())
- ∼**SAMLToken** (void)
- **operator bool** (void)
- bool **Authenticate** (const std::string &cafile, const std::string &capath)
- bool **Authenticate** (void)

### 6.240.1 Detailed Description

Class for manipulating SAML Token **Profile** (p. 357). This class is for generating/consuming SAML Token profile. See WS-Security SAML Token **Profile** (p. 357) 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** (p. 357) 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** (p. 357) 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).

## 6.240.2 Member Enumeration Documentation

#### 6.240.2.1 enum Arc::SAMLToken::SAMLVersion

Since the specfication SAMLVersion is for distinguishing two types of saml version. It is used as the parameter of constructor.

## 6.240.3 Constructor & Destructor Documentation

#### 6.240.3.1 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. 395) object will be used for authentication.

**Parameters:**

*soap* The SOAP message which contains the **SAMLToken** (p. 395) in the soap header

#### 6.240.3.2 Arc::SAMLToken::SAMLToken (SOAPEnvelope & *soap*, const std::string & *certfile*, const std::string & *keyfile*, SAMLVersion *saml_version* = `SAML2`, XMLNode *saml_assertion* = XMLNode `()`)

Constructor. Add SAML Token information into the SOAP header. Generated token contains elements SAML token and signature, and is meant to be used for authentication on the consuming side. This constructor is for a specific SAML Token profile usage, in which the attesting entity signs the SAML assertion for itself (self-sign). This usage implicitly requires that the relying party trust the attesting entity. More general (requires issuing authority) usage will be provided by other constructor. And the under-developing SAML service will be used as the issuing authority.

**Parameters:**

*soap* The SOAP message to which the SAML Token will be inserted.

*certfile* The certificate file.

*keyfile* The key file which will be used to create signature.

*samlversion* The SAML version, only SAML2 is supported currently.

*samlassertion* The SAML assertion got from 3rd party, and used for protecting the SOAP message; If not present, then self-signed assertion will be generated.

#### 6.240.3.3 Arc::SAMLToken::∼SAMLToken (void)

Deconstructor. Nothing to be done except finalizing the xmlsec library.

### 6.240.4 Member Function Documentation

#### 6.240.4.1 bool Arc::SAMLToken::Authenticate (void)

Check signature by using the cert information in soap message

#### 6.240.4.2 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. 396) 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

#### 6.240.4.3 Arc::SAMLToken::operator bool (void)

Returns true of constructor succeeded

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

- SAMLToken.h

## 6.241 Arc::ScalableTime< T > Class Template Reference

**template<class T> class Arc::ScalableTime< T >**

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

- JobDescription.h

## 6.242 Arc::ScalableTime< int > Class Template Reference

**template<> class Arc::ScalableTime< int >**

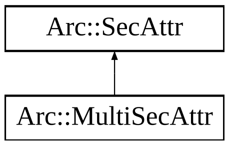
The documentation for this class was generated from the following file:

- JobDescription.h

## 6.243 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**

### 6.243.1 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.

### 6.243.2 Member Function Documentation

#### 6.243.2.1 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. 303).

#### 6.243.2.2 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.

#### 6.243.2.3 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.

#### 6.243.2.4 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. 303).

#### 6.243.2.5 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.

#### 6.243.2.6 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.

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

- SecAttr.h

## 6.244 Arc::SecAttrFormat Class Reference

Export/import format.

```
#include <SecAttr.h>
```

### 6.244.1 Detailed Description

Export/import format. Format is identified by textual identity string. Class description includes basic formats only. That list may be extended.

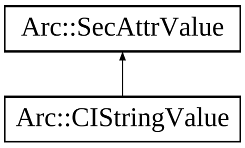
The documentation for this class was generated from the following file:

- SecAttr.h

## 6.245 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** ()

### 6.245.1 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.

### 6.245.2 Member Function Documentation

#### 6.245.2.1 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. 99).

#### 6.245.2.2 bool Arc::SecAttrValue::operator!= (SecAttrValue & *b*)

This is a convenience function to allow the usage of "not equal" conditions and need not be overridden.

#### 6.245.2.3 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.

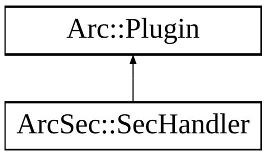
The documentation for this class was generated from the following file:

- SecAttrValue.h

## 6.246 ArcSec::SecHandler Class Reference

Base class for simple security handling plugins.

`#include <SecHandler.h>`Inheritance diagram for ArcSec::SecHandler::



### 6.246.1 Detailed Description

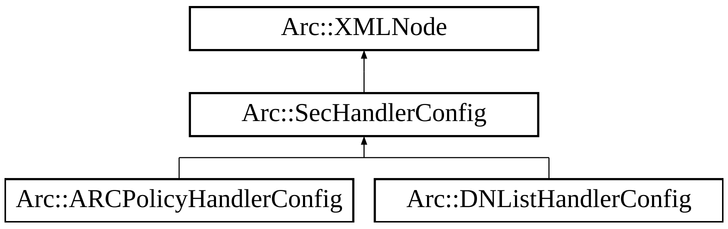
Base class for simple security handling plugins. This virtual class defines method Handle() which processes security related information/attributes in Message and optionally makes security decision. Instances of such classes are normally arranged in chains abd are called on incoming and outgoing messages 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. 404) 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

## 6.247 Arc::SecHandlerConfig Class Reference

Inheritance diagram for Arc::SecHandlerConfig::

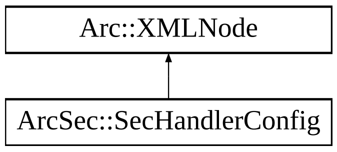


The documentation for this class was generated from the following file:

- ClientInterface.h

## 6.248 ArcSec::SecHandlerConfig Class Reference

`#include <SecHandler.h>`Inheritance diagram for ArcSec::SecHandlerConfig::



### 6.248.1 Detailed Description

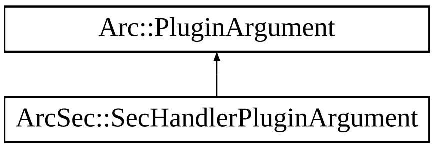
Helper class to create **Security** (p. 408) Handler configuration

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

- SecHandler.h

## 6.249 ArcSec::SecHandlerPluginArgument Class Reference

Inheritance diagram for ArcSec::SecHandlerPluginArgument::



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

- SecHandler.h

## 6.250 ArcSec::Security Class Reference

Common stuff used by security related slasses.

```
#include <Security.h>
```

### 6.250.1 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.

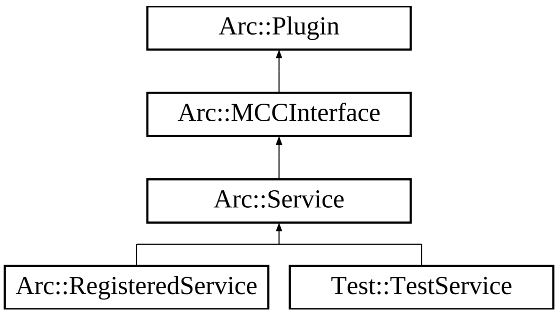
The documentation for this class was generated from the following file:

- Security.h

## 6.251 Arc::Service Class Reference

**Service** (p. 409) - last component in a **Message** (p. 290) 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**

### 6.251.1 Detailed Description

**Service** (p. 409) - last component in a **Message** (p. 290) Chain. This class which defines interface and common functionality for every **Service** (p. 409) plugin. Interface is made of method **process()** (p. 284) which is called by **Plexer** (p. 340) or **MCC** (p. 278) class. There is one **Service** (p. 409) object created for every service description processed by **Loader** (p. 264) class objects. Classes derived from **Service** (p. 409) class must implement **process()** (p. 284) method of **MCCInterface** (p. 284). It is up to developer how internal state of service is stored and communicated to other services and external utilites. **Service** (p. 409) 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. 278) it must accept and generate messages with **PayloadSOAP** (p. 323) payload. Method **process()** (p. 284) of class derived from **Service** (p. 409) 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 .

### 6.251.2 Constructor & Destructor Documentation

#### 6.251.2.1 Arc::Service::Service (Config ∗)

Example contructor - Server takes at least it's configuration subtree

### 6.251.3 Member Function Documentation

#### 6.251.3.1 virtual void Arc::Service::AddSecHandler (Config ∗ *cfg*, ArcSec::SecHandler ∗ *sechandler*, const std::string & *label* = "") `[virtual]`

Add security components/handlers to this **MCC** (p. 278). For more information please see description of **MCC::AddSecHandler** (p. 279)

#### 6.251.3.2 virtual std::string Arc::Service::getID () `[inline, virtual]`

**Service** (p. 409) may implement own service identitifer gathering method. This method return identifier of service which is used for registering it Information Services.

#### 6.251.3.3 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. 279)

#### 6.251.3.4 virtual bool Arc::Service::RegistrationCollector (XMLNode & *doc*) `[virtual]`

**Service** (p. 409) 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.

### 6.251.4 Field Documentation

#### 6.251.4.1 Logger Arc::Service::logger `[static, protected]`

**Logger** (p. 268) object used to print messages generated by this class.

#### 6.251.4.2 std::map<std::string,std::list<ArcSec::SecHandler∗> > Arc::Service::sechandlers_ `[protected]`

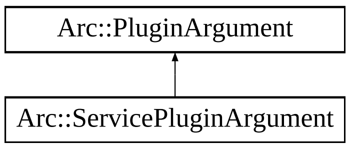
Set of labeled authentication and authorization handlers. **MCC** (p. 278) calls sequence of handlers at specific point depending on associated identifier. in most aces those are "in" and "out" for incoming and outgoing messages correspondingly.

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

- Service.h

## 6.252 Arc::ServicePluginArgument Class Reference

Inheritance diagram for Arc::ServicePluginArgument::



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

- Service.h

## 6.253 Arc::SimpleCondition Class Reference

Helper function to create simple thread.

`#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)

### 6.253.1 Detailed Description

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. The created thread will be joinable. Returns true on success. This function is currently disable becaueit is not clear if joinability is a needed feature Simple triggered condition. Provides condition and semaphor objects in one element.

### 6.253.2 Member Function Documentation

#### 6.253.2.1 void Arc::SimpleCondition::broadcast (void) `[inline]`

Signal about condition to all waiting threads

#### 6.253.2.2 void Arc::SimpleCondition::lock (void) `[inline]`

Acquire semaphor

#### 6.253.2.3 void Arc::SimpleCondition::reset (void) `[inline]`

Reset object to initial state

#### 6.253.2.4 void Arc::SimpleCondition::signal (void) `[inline]`

Signal about condition

#### 6.253.2.5 void Arc::SimpleCondition::signal_nonblock (void) `[inline]`

Signal about condition without using semaphor

#### 6.253.2.6 void Arc::SimpleCondition::unlock (void) `[inline]`

Release semaphor

#### 6.253.2.7 bool Arc::SimpleCondition::wait (int *t*) `[inline]`

Wait for condition no longer than t milliseconds

#### 6.253.2.8 void Arc::SimpleCondition::wait (void) `[inline]`

Wait for condition

#### 6.253.2.9 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

## 6.254 Arc::SOAPMessage Class Reference

**Message** (p. 290) 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)

### 6.254.1 Detailed Description

**Message** (p. 290) restricted to SOAP payload. This is a special **Message** (p. 290) 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. 290) but can carry only SOAP content.

### 6.254.2 Constructor & Destructor Documentation

#### 6.254.2.1 Arc::SOAPMessage::SOAPMessage (void) `[inline]`

Dummy constructor

#### 6.254.2.2 Arc::SOAPMessage::SOAPMessage (long *msg_ptr_addr*)

Copy constructor. Used by language bindigs

#### 6.254.2.3 Arc::SOAPMessage::SOAPMessage (Message & *msg*)

Copy constructor. Ensures shallow copy.

#### 6.254.2.4 Arc::SOAPMessage::∼SOAPMessage (void)

Destructor does not affect refered objects

### 6.254.3 Member Function Documentation

#### 6.254.3.1 MessageAttributes∗ Arc::SOAPMessage::Attributes (void) `[inline]`

Returns a pointer to the current attributes object or NULL if no attributes object has been assigned.

#### 6.254.3.2 void Arc::SOAPMessage::Payload (SOAPEnvelope ∗ *new_payload*)

Replace payload with a COPY of new one

#### 6.254.3.3 SOAPEnvelope∗ Arc::SOAPMessage::Payload (void)

Returns pointer to current payload or NULL if no payload assigned.

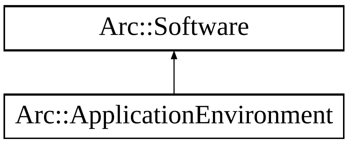
The documentation for this class was generated from the following file:

- SOAPMessage.h

## 6.255 Arc::Software Class Reference

Used to represent software (names and version) and comparison.

`#include <Software.h>`Inheritance diagram for Arc::Software::



### Public Types

- enum **ComparisonOperatorEnum** {
  **NOTEQUAL** = 0, **EQUAL** = 1, **GREATERTHAN** = 2, **LESSTHAN** = 3,
  **GREATERTHANOREQUAL** = 4, **LESSTHANOREQUAL** = 5 }
- typedef bool(Software::∗ **ComparisonOperator** )(const **Software** &) const

### Public Member Functions

- **Software** ()
- **Software** (const std::string &name_version)
- **Software** (const std::string &name, const std::string &version)
- **Software** (const std::string &family, const std::string &name, const std::string &version)
- bool **empty** () const
- bool **operator==** (const **Software** &sw) const
- bool **operator!=** (const **Software** &sw) const
- bool **operator>** (const **Software** &sw) const
- bool **operator<** (const **Software** &sw) const
- bool **operator>=** (const **Software** &sw) const
- bool **operator<=** (const **Software** &sw) const
- std::string **operator()** () const
- **operator std::string** (void) const
- const std::string & **getFamily** () const
- const std::string & **getName** () const
- const std::string & **getVersion** () const

### Static Public Member Functions

- static **ComparisonOperator convert** (const **ComparisonOperatorEnum** &co)
- static std::string **toString** (**ComparisonOperator** co)

### Static Public Attributes

- static const std::string **VERSIONTOKENS**

## Friends

- std::ostream & **operator**<< (std::ostream &out, const **Software** &sw)

### 6.255.1 Detailed Description

Used to represent software (names and version) and comparison. The **Software** (p. 417) class is used to represent the name of a piece of software internally. Generally software are identified by a name and possibly a version number. Some software can also be categorized by type or family (compilers, operating system, etc.). A software object can be compared to other software objects using the comparison operators contained in this class. The basic usage of this class is to test if some specified software requirement (**SoftwareRequirement** (p. 425)) are fulfilled, by using the comparability of the class.

Internally the **Software** (p. 417) object is represented by a family and name identifier, and the software version is tokenized at the characters defined in VERSIONTOKENS, and stored as a list of tokens.

### 6.255.2 Member Typedef Documentation

#### 6.255.2.1 typedef bool(Software::∗ Arc::Software::ComparisonOperator)(const Software &) const

Definition of a comparison operator method pointer. This `typedef` defines a comparison operator method pointer.

**See also:**

**operator==** (p. 422),
**operator!=** (p. 421),
**operator>** (p. 422),
**operator<** (p. 421),
**operator>=** (p. 423),
**operator<=** (p. 421),
**ComparisonOperatorEnum** (p. 418).

### 6.255.3 Member Enumeration Documentation

#### 6.255.3.1 enum Arc::Software::ComparisonOperatorEnum

Comparison operator enum. The **ComparisonOperatorEnum** (p. 418) enumeration is a 1-1 correspondance between the defined comparison method operators (**Software::ComparisonOperator** (p. 418)), and can be used in circumstances where method pointers are not supported.

**Enumerator:**

***NOTEQUAL*** see **operator!=** (p. 421)

***EQUAL*** see **operator==** (p. 422)

***GREATERTHAN*** see **operator>** (p. 422)

***LESSTHAN*** see **operator<** (p. 421)

***GREATERTHANOREQUAL*** see **operator>=** (p. 423)

***LESSTHANOREQUAL*** see **operator<=** (p. 421)

## 6.255.4 Constructor & Destructor Documentation

#### 6.255.4.1 Arc::Software::Software () `[inline]`

Dummy constructor. This constructor creates a empty object.

#### 6.255.4.2 Arc::Software::Software (const std::string & *name_version*)

Create a **Software** (p. 417) object. Create a **Software** (p. 417) object from a single string composed of a name and a version part. The created object will contain a empty family part. The name and version part of the string will be split at the first occurence of a dash (-) which is followed by a digit (0-9). If the string does not contain such a pattern, the passed string will be taken to be the name and version will be empty.

**Parameters:**

***name_version*** should be a string composed of the name and version of the software to represent.

#### 6.255.4.3 Arc::Software::Software (const std::string & *name*, const std::string & *version*)

Create a **Software** (p. 417) object. Create a **Software** (p. 417) object with the specified name and version. The family part will be left empty.

**Parameters:**

***name*** the software name to represent.

***version*** the software version to represent.

#### 6.255.4.4 Arc::Software::Software (const std::string & *family*, const std::string & *name*, const std::string & *version*)

Create a **Software** (p. 417) object. Create a **Software** (p. 417) object with the specified family, name and version.

**Parameters:**

***family*** the software family to represent.

***name*** the software name to represent.

***version*** the software version to represent.

## 6.255.5 Member Function Documentation

#### 6.255.5.1 static ComparisonOperator Arc::Software::convert (const ComparisonOperatorEnum & *co*) `[static]`

Convert a **ComparisonOperatorEnum** (p. 418) value to a comparison method pointer. The passed **ComparisonOperatorEnum** (p. 418) will be converted to a comparison method pointer defined by the **Software::ComparisonOperator** (p. 418) typedef.

This static method is not defined in language bindings created with Swig, since method pointers are not supported by Swig.

**Parameters:**

***co*** a **ComparisonOperatorEnum** (p. 418) value.

**Returns:**

A method pointer to a comparison method is returned.

#### 6.255.5.2 bool Arc::Software::empty () const `[inline]`

Indicates whether the object is empty.

**Returns:**

`true` if the name of this object is empty, otherwise `false`.

#### 6.255.5.3 const std::string& Arc::Software::getFamily () const `[inline]`

Get family.

**Returns:**

The family the represented software belongs to is returned.

#### 6.255.5.4 const std::string& Arc::Software::getName () const `[inline]`

Get name.

**Returns:**

The name of the represented software is returned.

#### 6.255.5.5 const std::string& Arc::Software::getVersion () const `[inline]`

Get version.

**Returns:**

The version of the represented software is returned.

#### 6.255.5.6 Arc::Software::operator std::string (void) const `[inline]`

Cast to string. This casting operator behaves exactly as **operator()()** (p. 421) does. The cast is used like (std::string) <software-object>.

**See also:**

**operator()()** (p. 421).

References operator()().

#### 6.255.5.7 bool Arc::Software::operator!= (const Software & *sw*) const `[inline]`

Inequality operator (non-trivial behaviour). The inequality operator should be used to test if two **Software** (p. 417) objects are of different versions but share the same name and family. So it should not be used to test if two **Software** (p. 417) objects differ in either name, version or family. Two **Software** (p. 417) objects are inequal if they share the same name and family but have different versions and the versions are non-empty.

**Parameters:**

*sw* is the RHS **Software** (p. 417) object.

**Returns:**

`true` when the two objects are inequal, otherwise `false`.

#### 6.255.5.8 std::string Arc::Software::operator() () const

Get string representation. Returns the string representation of this object, which is 'family'-'name'-'version'.

**Returns:**

The string representation of this object is returned.

**See also:**

operator std::string().

Referenced by operator std::string().

#### 6.255.5.9 bool Arc::Software::operator< (const Software & *sw*) const `[inline]`

Less-than operator. The behaviour of this less-than operator is equivalent to the greater-than operator (**operator**>() (p. 422)) with the LHS and RHS swapped.

**Parameters:**

*sw* is the RHS object.

**Returns:**

`true` if the LHS is less than the RHS, otherwise `false`.

**See also:**

**operator**>() (p. 422).

#### 6.255.5.10 bool Arc::Software::operator<= (const Software & *sw*) const `[inline]`

Less-than or equal operator. The LHS object is greater than or equal to the RHS object if the LHS equal the RHS (**operator==()** (p. 422)) or if the LHS is greater than the RHS (**operator**>() (p. 422)).

**Parameters:**

*sw* is the RHS object.

**Returns:**

`true` if the LHS is less than or equal the RHS, otherwise `false`.

**See also:**

**operator==()** (p. 422),
**operator<()** (p. 421).

#### 6.255.5.11 bool Arc::Software::operator== (const Software & *sw*) const `[inline]`

Equality operator. Two **Software** (p. 417) objects are equal if they are of the same family, and if they have the same name. If BOTH objects specifies a version they must also equal, for the objects to be equal. Otherwise the two objects does not equal. This operator can also be represented by the **Software::EQUAL** (p. 418) **ComparisonOperatorEnum** (p. 418) value.

**Parameters:**

*sw* is the RHS **Software** (p. 417) object.

**Returns:**

`true` when the two objects equals, otherwise `false`.

#### 6.255.5.12 bool Arc::Software::operator> (const Software & *sw*) const

Greater-than operator. For the LHS object to be greater than the RHS object they must first share the same family and name and have non-empty versions. Then, the first version token of each object is compared and if they are identical, the two next version tokens will be compared. If not identical, the two tokens will be parsed as integers, and if parsing fails the LHS is not greater than the RHS. If parsing succeeds and the integers equals, the two next tokens will be compared, otherwise the comparison is resolved by the integer comparison.

If the LHS contains more version tokens than the RHS, and the comparison have not been resolved at the point of equal number of tokens, then if the additional tokens contains a token which cannot be parsed to a integer the LHS is not greater than the RHS. If the parsed integer is not 0 then the LHS is greater than the RHS. If the rest of the additional tokens are 0, the LHS is not greater than the RHS.

If the RHS contains more version tokens than the LHS and comparison have not been resolved at the point of equal number of tokens, or simply if comparison have not been resolved at the point of equal number of tokens, then the LHS is not greater than the RHS.

**Parameters:**

*sw* is the RHS object.

**Returns:**

`true` if the LHS is greater than the RHS, otherwise `false`.

#### 6.255.5.13 bool Arc::Software::operator>= (const Software & *sw*) const `[inline]`

Greater-than or equal operator. The LHS object is greater than or equal to the RHS object if the LHS equal the RHS (**operator==()** (p. 422)) or if the LHS is greater than the RHS (**operator>()** (p. 422)).

**Parameters:**

*sw* is the RHS object.

**Returns:**

`true` if the LHS is greated than or equal the RHS, otherwise `false`.

**See also:**

**operator==()** (p. 422),
**operator>()** (p. 422).

#### 6.255.5.14 static std::string Arc::Software::toString (ComparisonOperator *co*) `[static]`

Convert **Software::ComparisonOperator** (p. 418) to a string. This method is not available in language bindings created by Swig, since method pointers are not supported by Swig.

**Parameters:**

*co* is a **Software::ComparisonOperator** (p. 418).

**Returns:**

The string representation of the passed **Software::ComparisonOperator** (p. 418) is returned.

### 6.255.6 Friends And Related Function Documentation

#### 6.255.6.1 std::ostream& operator<< (std::ostream & *out*, const Software & *sw*) `[friend]`

Write **Software** (p. 417) string representation to a std::ostream. Write the string representation of a **Software** (p. 417) object to a std::ostream.

**Parameters:**

*out* is a std::ostream to write the string representation of the **Software** (p. 417) object to.

*sw* is the **Software** (p. 417) object to write to the std::ostream.

**Returns:**

The passed std::ostream *out* is returned.

### 6.255.7 Field Documentation

#### 6.255.7.1 const std::string Arc::Software::VERSIONTOKENS `[static]`

Tokens used to split version string. This string constant specifies which tokens will be used to split the version string.

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

- Software.h

## 6.256 Arc::SoftwareRequirement Class Reference

Class used to express and resolve version requirements on software.

```
#include <Software.h>
```

### Public Member Functions

- **SoftwareRequirement** (bool requiresAll=false)
- **SoftwareRequirement** (const **Software** &sw, **Software::ComparisonOperator** swComOp=&Software::operator==, bool requiresAll=false)
- **SoftwareRequirement** (const **Software** &sw, **Software::ComparisonOperatorEnum** co, bool requiresAll=false)
- **SoftwareRequirement** & **operator=** (const **SoftwareRequirement** &sr)
- **SoftwareRequirement** (const **SoftwareRequirement** &sr)
- void **add** (const **Software** &sw, **Software::ComparisonOperator** swComOp=&Software::operator==)
- void **add** (const **Software** &sw, **Software::ComparisonOperatorEnum** co)
- bool **isRequiringAll** () const
- void **setRequirement** (bool all)
- bool **isSatisfied** (const **Software** &sw) const
- bool **isSatisfied** (const std::list< **Software** > &swList) const
- bool **isSatisfied** (const std::list< **ApplicationEnvironment** > &swList) const
- bool **selectSoftware** (const **Software** &sw)
- bool **selectSoftware** (const std::list< **Software** > &swList)
- bool **selectSoftware** (const std::list< **ApplicationEnvironment** > &swList)
- bool **isResolved** () const
- bool **empty** () const
- void **clear** ()
- const std::list< **Software** > & **getSoftwareList** () const
- const std::list< **Software::ComparisonOperator** > & **getComparisonOperatorList** () const

### 6.256.1 Detailed Description

Class used to express and resolve version requirements on software. A requirement in this class is defined as a pair composed of a **Software** (p. 417) object and either a **Software::ComparisonOperator** (p. 418) method pointer or equally a **Software::ComparisonOperatorEnum** (p. 418) enum value. A **SoftwareRequirement** (p. 425) object can contain multiple of such requirements, and then it can specified if all these requirements should be satisfied, or if it is enough to satisfy only one of them. The requirements can be satisfied by a single **Software** (p. 417) object or a list of either **Software** (p. 417) or **ApplicationEnvironment** (p. 65) objects, by using the method **isSatisfied()** (p. 429). This class also contain a number of methods (**selectSoftware()** (p. 431)) to select **Software** (p. 417) objects which are satisfying the requirements, and in this way resolving requirements.

### 6.256.2 Constructor & Destructor Documentation

#### 6.256.2.1 Arc::SoftwareRequirement::SoftwareRequirement (bool *requiresAll* = `false`) `[inline]`

Create a empty **SoftwareRequirement** (p. 425) object. The created **SoftwareRequirement** (p. 425) object will contain no requirements.

**Parameters:**

***requiresAll*** indicates whether the all requirements have to be satisfied (`true`) or if only a single one (`false`), the default is that only a single requirement need to be satisfied.

#### 6.256.2.2 Arc::SoftwareRequirement::SoftwareRequirement (const Software & *sw*, Software::ComparisonOperator *swComOp* = `&Software::operator==`, bool *requiresAll* = `false`)

Create a **SoftwareRequirement** (p. 425) object. The created **SoftwareRequirement** (p. 425) object will contain one requirement specified by the **Software** (p. 417) object *sw*, and the **Software::ComparisonOperator** (p. 418) *swComOp*.

This constructor is not available in language bindings created by Swig, since method pointers are not supported by Swig, see **SoftwareRequirement(const Software&, Software::ComparisonOperatorEnum, bool)** (p. 426) instead.

**Parameters:**

***sw*** is the **Software** (p. 417) object of the requirement to add.

***swComOp*** is the **Software::ComparisonOperator** (p. 418) of the requirement to add.

***requiresAll*** indicates whether the all requirements have to be satisfied (`true`) or if only a single one (`false`), the default is that only a single requirement need to be satisfied.

#### 6.256.2.3 Arc::SoftwareRequirement::SoftwareRequirement (const Software & *sw*, Software::ComparisonOperatorEnum *co*, bool *requiresAll* = `false`)

Create a **SoftwareRequirement** (p. 425) object. The created **SoftwareRequirement** (p. 425) object will contain one requirement specified by the **Software** (p. 417) object *sw*, and the **Software::ComparisonOperatorEnum** (p. 418) *co*.

**Parameters:**

***sw*** is the **Software** (p. 417) object of the requirement to add.

***co*** is the **Software::ComparisonOperatorEnum** (p. 418) of the requirement to add.

***requiresAll*** indicates whether the all requirements have to be satisfied (`true`) or if only a single one (`false`), the default is that only a single requirement need to be satisfied.

#### 6.256.2.4 Arc::SoftwareRequirement::SoftwareRequirement (const SoftwareRequirement & *sr*) `[inline]`

Copy constructor. Create a **SoftwareRequirement** (p. 425) object from another **SoftwareRequirement** (p. 425) object.

**Parameters:**

***sr*** is the **SoftwareRequirement** (p. 425) object to make a copy of.

### 6.256.3 Member Function Documentation

#### 6.256.3.1 void Arc::SoftwareRequirement::add (const Software & *sw*, Software::ComparisonOperatorEnum *co*)

Add a **Software** (p. 417) object a corresponding comparion operator to this object. Adds software name and version to list of requirements and associates the comparison operator with it (equality by default).

**Parameters:**

*sw* is the **Software** (p. 417) object to add as part of a requirement.

*co* is the **Software::ComparisonOperatorEnum** (p. 418) value to add as part of a requirement, the default enum will be **Software::EQUAL** (p. 418).

#### 6.256.3.2 void Arc::SoftwareRequirement::add (const Software & *sw*, Software::ComparisonOperator *swComOp* = `&Software::operator==`)

Add a **Software** (p. 417) object a corresponding comparion operator to this object. Adds software name and version to list of requirements and associates the comparison operator with it (equality by default).

This method is not available in language bindings created by Swig, since method pointers are not supported by Swig, see **add(const Software&, Software::ComparisonOperatorEnum)** (p. 427) instead.

**Parameters:**

*sw* is the **Software** (p. 417) object to add as part of a requirement.

*swComOp* is the **Software::ComparisonOperator** (p. 418) method pointer to add as part of a requirement, the default operator will be **Software::operator==()** (p. 422).

#### 6.256.3.3 void Arc::SoftwareRequirement::clear () `[inline]`

Clear the object. The requirements in this object will be cleared when invoking this method.

#### 6.256.3.4 bool Arc::SoftwareRequirement::empty () const `[inline]`

Test if the object is empty.

**Returns:**

`true` if this object do no contain any requirements, otherwise `false`.

#### 6.256.3.5 const std::list<Software::ComparisonOperator>& Arc::SoftwareRequirement::getComparisonOperatorList () const `[inline]`

Get list of comparison operators.

**Returns:**

The list of internally stored comparison operators is returned.

**See also:**

**Software::ComparisonOperator** (p. 418),
**getSoftwareList** (p. 428).

#### 6.256.3.6 const std::list<Software>& Arc::SoftwareRequirement::getSoftwareList () const `[inline]`

Get list of **Software** (p. 417) objects.

**Returns:**

The list of internally stored **Software** (p. 417) objects is returned.

**See also:**

**Software** (p. 417),
**getComparisonOperatorList** (p. 427).

#### 6.256.3.7 bool Arc::SoftwareRequirement::isRequiringAll () const `[inline]`

Indicates whether all requirments has to be satisfied. This method returns `true` if all requirements has to be satisfied. If only one requirement has to be satisfied, `false` is returned.

**Returns:**

`true` if all requirements has to be satisfied, otherwise `false`.

**See also:**

**setRequirement** (p. 431).

#### 6.256.3.8 bool Arc::SoftwareRequirement::isResolved () const

Indicates whether requirements have been resolved or not. If specified that only one requirement has to be satisfied, then for this object to be resolved it can only contain one requirement and it has use the equal operator (**Software::operator==** (p. 422)).

If specified that all requirements has to be satisfied, then for this object to be resolved each requirement must have a **Software** (p. 417) object with a unique family/name composition, i.e. no other requirements have a **Software** (p. 417) object with the same family/name composition, and each requirement must use the equal operator (**Software::operator==** (p. 422)).

If this object has been resolved then `true` is returned when invoking this method, otherwise `false` is returned.

**Returns:**

`true` if this object have been resolved, otherwise `false`.

#### 6.256.3.9 bool Arc::SoftwareRequirement::isSatisfied (const std::list< ApplicationEnvironment > & *swList*) const

Test if requirements are satisfied. This method behaves in exactly the same way as the **isSatisfied(const Software&) const** (p. 429)method does.

**Parameters:**

*swList* is the list of **ApplicationEnvironment** (p. 65) objects which should be used to try satisfy the requirements.

**Returns:**

`true` if requirements are satisfied, otherwise `false`.

**See also:**

**isSatisfied(const Software&) const** (p. 429),
**isSatisfied(const std::list<Software>&) const** (p. 429),
**selectSoftware(const std::list<ApplicationEnvironment>&)** (p. 430),
**isResolved() const** (p. 428).

#### 6.256.3.10 bool Arc::SoftwareRequirement::isSatisfied (const std::list< Software > & *swList*) const

Test if requirements are satisfied. Returns `true` if stored requirements are satisfied by software specified in *swList*, otherwise `false` is returned.

Note that if all requirements must be satisfied and multiple requirements exist having identical name and family all these requirements should be satisfied by a single **Software** (p. 417) object.

**Parameters:**

*swList* is the list of **Software** (p. 417) objects which should be used to try satisfy the requirements.

**Returns:**

`true` if requirements are satisfied, otherwise `false`.

**See also:**

**isSatisfied(const Software&) const** (p. 429),
**isSatisfied(const std::list<ApplicationEnvironment>&) const** (p. 429),
**selectSoftware(const std::list<Software>&)** (p. 430),
**isResolved() const** (p. 428).

#### 6.256.3.11 bool Arc::SoftwareRequirement::isSatisfied (const Software & *sw*) const `[inline]`

Test if requirements are satisfied. Returns `true` if the requirements are satisfied by the specified **Software** (p. 417) *sw*, otherwise `false` is returned.

**Parameters:**

*sw* is the **Software** (p. 417) which should satisfy the requirements.

**Returns:**

`true` if requirements are satisfied, otherwise `false`.

**See also:**

**isSatisfied(const std::list<Software>&) const** (p. 429),
**isSatisfied(const std::list<ApplicationEnvironment>&) const** (p. 429),
**selectSoftware(const Software&)** (p. 431),
**isResolved() const** (p. 428).

References isSatisfied().

Referenced by isSatisfied().

#### 6.256.3.12 SoftwareRequirement& Arc::SoftwareRequirement::operator= (const SoftwareRequirement & *sr*)

Assignment operator. Set this object equal to that of the passed **SoftwareRequirement** (p. 425) object *sr*.

**Parameters:**

***sr*** is the **SoftwareRequirement** (p. 425) object to set object equal to.

#### 6.256.3.13 bool Arc::SoftwareRequirement::selectSoftware (const std::list< ApplicationEnvironment > & *swList*)

Select software. This method behaves exactly as the **selectSoftware(const std::list<Software>&)** (p. 430) method does.

**Parameters:**

***swList*** is a list of **ApplicationEnvironment** (p. 65) objects used to satisfy requirements.

**Returns:**

`true` if requirements are satisfied, otherwise `false`.

**See also:**

**selectSoftware(const Software&)** (p. 431),
**selectSoftware(const std::list<Software>&)** (p. 430),
**isSatisfied(const std::list<ApplicationEnvironment>&) const** (p. 429),
**isResolved() const** (p. 428).

#### 6.256.3.14 bool Arc::SoftwareRequirement::selectSoftware (const std::list< Software > & *swList*)

Select software. If the passed list of **Software** (p. 417) objects *swList* do not satisfy the requirements `false` is returned and this object is not modified. If however the list of **Software** (p. 417) objects *swList* do satisfy the requirements `true` is returned and the **Software** (p. 417) objects satisfying the requirements will replace these with the equality operator (**Software::operator==** (p. 422)) used as the comparator for the new requirements.

Note that if all requirements must be satisfied and multiple requirements exist having identical name and family all these requirements should be satisfied by a single **Software** (p. 417) object and it will replace all these requirements.

**Parameters:**

***swList*** is a list of **Software** (p. 417) objects used to satisfy requirements.

**Returns:**

`true` if requirements are satisfied, otherwise `false`.

**See also:**

**selectSoftware(const Software&)** (p. 431),
**selectSoftware(const std::list<ApplicationEnvironment>&)** (p. 430),
**isSatisfied(const std::list<Software>&) const** (p. 429),
**isResolved() const** (p. 428).

#### 6.256.3.15 bool Arc::SoftwareRequirement::selectSoftware (const Software & *sw*) `[inline]`

Select software. If the passed **Software** (p. 417) *sw* do not satisfy the requirements `false` is returned and this object is not modified. If however the **Software** (p. 417) object *sw* do satisfy the requirements `true` is returned and the requirements are set to equal the *sw* **Software** (p. 417) object.

**Parameters:**

***sw*** is the **Software** (p. 417) object used to satisfy requirements.

**Returns:**

`true` if requirements are satisfied, otherwise `false`.

**See also:**

**selectSoftware(const std::list<Software>&)** (p. 430),
**selectSoftware(const std::list<ApplicationEnvironment>&)** (p. 430),
**isSatisfied(const Software&) const** (p. 429),
**isResolved() const** (p. 428).

References selectSoftware().

Referenced by selectSoftware().

#### 6.256.3.16 void Arc::SoftwareRequirement::setRequirement (bool *all*) `[inline]`

Set relation between requirements. Specifies if all requirements stored need to be satisfied or if it is enough to satisfy only one of them.

**Parameters:**

***all*** is a boolean specifying if all requirements has to be satified.

**See also:**

**isRequiringAll()** (p. 428).

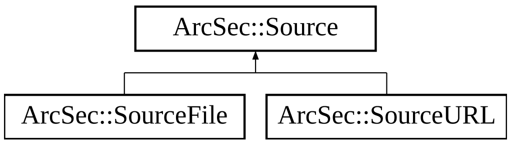
The documentation for this class was generated from the following file:

- Software.h

## 6.257 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)

### 6.257.1 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.

### 6.257.2 Constructor & Destructor Documentation

#### 6.257.2.1 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.

#### 6.257.2.2 ArcSec::Source::Source (Arc::URL & *url*)

Fetch XML document from specified url and parse it. This constructor is not implemented yet.

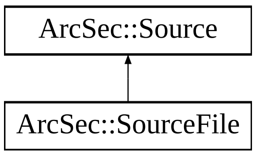
The documentation for this class was generated from the following file:

- Source.h

## 6.258 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)

### 6.258.1 Detailed Description

Convenience class for obtaining XML document from file.

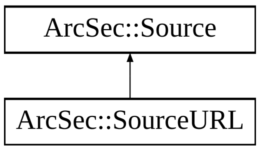
The documentation for this class was generated from the following file:

- Source.h

## 6.259 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)

### 6.259.1 Detailed Description

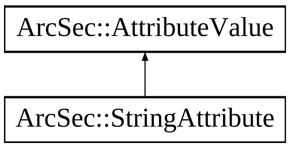
Convenience class for obtaining XML document from remote URL.

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

- Source.h

## 6.260 ArcSec::StringAttribute Class Reference

Inheritance diagram for ArcSec::StringAttribute::



### Public Member Functions

- virtual std::string **encode** ()
- virtual std::string **getType** ()
- virtual std::string **getId** ()

### 6.260.1 Member Function Documentation

#### 6.260.1.1 virtual std::string ArcSec::StringAttribute::encode () `[inline, virtual]`

encode the value in a string format

Implements **ArcSec::AttributeValue** (p. 78).

#### 6.260.1.2 virtual std::string ArcSec::StringAttribute::getId () `[inline, virtual]`

Get the AttributeId of the <Attribute>

Implements **ArcSec::AttributeValue** (p. 78).

#### 6.260.1.3 virtual std::string ArcSec::StringAttribute::getType () `[inline, virtual]`

Get the DataType of the <Attribute>

Implements **ArcSec::AttributeValue** (p. 78).

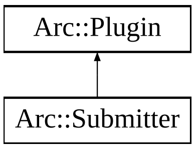
The documentation for this class was generated from the following file:

- StringAttribute.h

## 6.261 Arc::Submitter Class Reference

Base class for the Submitters.

`#include <Submitter.h>`Inheritance diagram for Arc::Submitter::



### Public Member Functions

- virtual **URL Submit** (const **JobDescription** &jobdesc, const **ExecutionTarget** &et) const =0
- virtual **URL Migrate** (const **URL** &jobid, const **JobDescription** &jobdesc, const **ExecutionTarget** &et, bool forcemigration) const =0

### 6.261.1 Detailed Description

Base class for the Submitters. **Submitter** (p. 436) is the base class for Grid middleware specialized **Submitter** (p. 436) objects. The class submits job(s) to the computing resource it represents and uploads (needed by the job) local input files.

### 6.261.2 Member Function Documentation

#### 6.261.2.1 virtual URL Arc::Submitter::Migrate (const URL & *jobid*, const JobDescription & *jobdesc*, const ExecutionTarget & *et*, bool *forcemigration*) const `[pure virtual]`

This virtual method should be overridden by plugins which should be capable of migrating jobs. The active job which should be migrated is pointed to by the **URL** (p. 456) jobid, and is represented by the **JobDescription** (p. 257) jobdesc. The forcemigration boolean specifies if the migration should succeed if the active job cannot be terminated. The protected method AddJob can be used to save job information. This method should return the **URL** (p. 456) of the migrated job. In case migration fails an empty **URL** (p. 456) should be returned.

#### 6.261.2.2 virtual URL Arc::Submitter::Submit (const JobDescription & *jobdesc*, const ExecutionTarget & *et*) const `[pure virtual]`

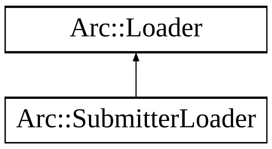
This virtual method should be overridden by plugins which should be capable of submitting jobs, defined in the **JobDescription** (p. 257) jobdesc, to the **ExecutionTarget** (p. 207) et. The protected convenience method AddJob can be used to save job information. This method should return the **URL** (p. 456) of the submitted job. In case submission fails an empty **URL** (p. 456) should be returned.

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

- Submitter.h

## 6.262 Arc::SubmitterLoader Class Reference

`#include <Submitter.h>`Inheritance diagram for Arc::SubmitterLoader::



### Public Member Functions

- **SubmitterLoader** ()
- ∼**SubmitterLoader** ()
- **Submitter** ∗ **load** (const std::string &name, const **UserConfig** &usercfg)
- const std::list< **Submitter** ∗ > & **GetSubmitters** () const

### 6.262.1 Detailed Description

Class responsible for loading **Submitter** (p. 436) plugins The **Submitter** (p. 436) objects returned by a **SubmitterLoader** (p. 437) must not be used after the **SubmitterLoader** (p. 437) goes out of scope.

### 6.262.2 Constructor & Destructor Documentation

#### 6.262.2.1 Arc::SubmitterLoader::SubmitterLoader ()

Constructor Creates a new **SubmitterLoader** (p. 437).

#### 6.262.2.2 Arc::SubmitterLoader::∼SubmitterLoader ()

Destructor Calling the destructor destroys all Submitters loaded by the **SubmitterLoader** (p. 437) instance.

### 6.262.3 Member Function Documentation

#### 6.262.3.1 const std::list<Submitter∗>& Arc::SubmitterLoader::GetSubmitters () const `[inline]`

Retrieve the list of loaded Submitters.

**Returns:**

A reference to the list of Submitters.

#### 6.262.3.2 Submitter∗ Arc::SubmitterLoader::load (const std::string & *name*, const UserConfig & *usercfg*)

Load a new **Submitter** (p. 436)

**Parameters:**

| | |
|---|---|
| ***name*** | The name of the **Submitter** (p. 436) to load. |
| ***usercfg*** | The **UserConfig** (p. 468) object for the new **Submitter** (p. 436). |

**Returns:**

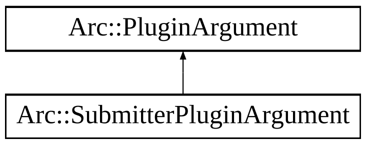
A pointer to the new **Submitter** (p. 436) (NULL on error).

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

- Submitter.h

## 6.263 Arc::SubmitterPluginArgument Class Reference

Inheritance diagram for Arc::SubmitterPluginArgument::



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

- Submitter.h

## 6.264 Arc::TargetGenerator Class Reference

Target generation class

```
#include <TargetGenerator.h>
```

### Public Member Functions

- **TargetGenerator** (const **UserConfig** &usercfg)
- void **GetTargets** (int targetType, int detailLevel)
- const std::list< **ExecutionTarget** > & **FoundTargets** () const
- std::list< **ExecutionTarget** > & **ModifyFoundTargets** ()
- const std::list< **XMLNode** ∗ > & **FoundJobs** () const
- bool **AddService** (const **URL** &url)
- bool **AddIndexServer** (const **URL** &url)
- void **AddTarget** (const **ExecutionTarget** &target)
- void **AddJob** (const **XMLNode** &job)
- void **RetrieverDone** ()
- void **PrintTargetInfo** (bool longlist) const

### 6.264.1 Detailed Description

Target generation class The **TargetGenerator** (p. 440) class is the umbrella class for resource discovery and information retrieval (index servers and computing clusters). It can also be used to locate user Grid jobs but does not collect the job details (see the arcsync CLI)). The **TargetGenerator** (p. 440) loads **TargetRetriever** (p. 443) plugins (which implements the actual information retrieval) from **URL** (p. 456) objects found in the **UserConfig** (p. 468) object passed to its constructor using the custem **TargetRetrieverLoader** (p. 445). E.g. if an **URL** (p. 456) pointing to an ARC1 computing resource is found in the **UserConfig** (p. 468) object the TargetRetrieverARC1 is loaded.

### 6.264.2 Constructor & Destructor Documentation

#### 6.264.2.1 Arc::TargetGenerator::TargetGenerator (const UserConfig & *usercfg*)

Create a **TargetGenerator** (p. 440) object. Default constructor to create a TargeGenerator. The constructor reads the computing and index service **URL** (p. 456) objects from the passed **UserConfig** (p. 468) object using the **UserConfig** (p. 468):GetSelectedServices method. From each **URL** (p. 456) a matching specialized **TargetRetriever** (p. 443) plugin is loaded using the **TargetRetrieverLoader** (p. 445).

**Parameters:**

***usercfg*** Reference to **UserConfig** (p. 468) object with **URL** (p. 456) objects to computing and/or index services and paths to user credentials.

### 6.264.3 Member Function Documentation

#### 6.264.3.1 bool Arc::TargetGenerator::AddIndexServer (const URL & *url*)

Add a new index server to the foundIndexServers list. Method to add a new index server to the list of foundIndexServers in a thread secure way. Compares the argument **URL** (p. 456) against the servers returned by **UserConfig::GetRejectedServices** (p. 479) and only allows to add the service if not specifically rejected.

**Parameters:**

***url*** **URL** (p. 456) pointing to the index server.

#### 6.264.3.2 void Arc::TargetGenerator::AddJob (const XMLNode & *job*)

Add a new **Job** (p. 251) to the foundJobs list. Method to add a new **Job** (p. 251) (usually discovered by a **TargetRetriever** (p. 443)) to the list of foundJobs in a thread secure way.

**Parameters:**

***job*** **XMLNode** (p. 547) describing the job.

#### 6.264.3.3 bool Arc::TargetGenerator::AddService (const URL & *url*)

Add a new computing service to the foundServices list. Method to add a new service to the list of foundServices in a thread secure way. Compares the argument **URL** (p. 456) against the services returned by **UserConfig::GetRejectedServices** (p. 479) and only allows to add the service if not specifically rejected.

**Parameters:**

***url*** **URL** (p. 456) pointing to the information system of the computing service.

#### 6.264.3.4 void Arc::TargetGenerator::AddTarget (const ExecutionTarget & *target*)

Add a new **ExecutionTarget** (p. 207) to the foundTargets list. Method to add a new **ExecutionTarget** (p. 207) (usually discovered by a **TargetRetriever** (p. 443)) to the list of foundTargets in a thread secure way.

**Parameters:**

***target*** **ExecutionTarget** (p. 207) to be added.

#### 6.264.3.5 const std::list<XMLNode∗>& Arc::TargetGenerator::FoundJobs () const

Return Grid jobs found by GetTargets. Method to return the list of Grid jobs found by a call to the GetTargets method.

#### 6.264.3.6 const std::list<ExecutionTarget>& Arc::TargetGenerator::FoundTargets () const

Return targets found by GetTargets. Method to return a const list of **ExecutionTarget** (p. 207) objects (currently only supported Target type) found by the GetTarget method.

#### 6.264.3.7 void Arc::TargetGenerator::GetTargets (int *targetType*, int *detailLevel*)

Find available targets. Method to prepare a list of chosen Targets with a specified detail level. Current implementation supports finding computing clusters (**ExecutionTarget** (p. 207)) with full detail level and Grid jobs with limited detail level.

**Parameters:**

***targetType*** 0 = **ExecutionTarget** (p. 207), 1 = Grid jobs

***detailLevel*** 1 = All details, 2 = Limited details (not implemented)

#### 6.264.3.8 std::list<ExecutionTarget>& Arc::TargetGenerator::ModifyFoundTargets ()

Return targets found by GetTargets. Method to return the list of **ExecutionTarget** (p. 207) objects (currently only supported Target type) found by the GetTarget method.

#### 6.264.3.9 void Arc::TargetGenerator::PrintTargetInfo (bool *longlist*) const

Prints target information. Method to print information of the found targets to std::cout.

**Parameters:**

***longlist*** false for minimal information, true for detailed information

#### 6.264.3.10 void Arc::TargetGenerator::RetrieverDone ()

Decrement the threadCounter by 1. Method to decrement the threadCounter by 1 in a thread secure way.

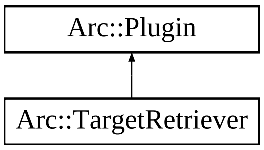
The documentation for this class was generated from the following file:

- TargetGenerator.h

## 6.265 Arc::TargetRetriever Class Reference

TargetRetriever base class

`#include <TargetRetriever.h>`Inheritance diagram for Arc::TargetRetriever::



### Public Member Functions

- virtual void **GetTargets** (**TargetGenerator** &mom, int targetType, int detailLevel)=0

### Protected Member Functions

- **TargetRetriever** (const **UserConfig** &usercfg, const **URL** &url, ServiceType st, const std::string &flavour)

### 6.265.1 Detailed Description

TargetRetriever base class The **TargetRetriever** (p. 443) class is a pure virtual base class to be used for grid flavour specializations. It is designed to work in conjunction with the **TargetGenerator** (p. 440).

### 6.265.2 Constructor & Destructor Documentation

#### 6.265.2.1 Arc::TargetRetriever::TargetRetriever (const UserConfig & *usercfg*, const URL & *url*, ServiceType *st*, const std::string & *flavour*) `[protected]`

**TargetRetriever** (p. 443) constructor. Default constructor to create a TargeGenerator. The constructor reads the computing and index service **URL** (p. 456) objects from the

**Parameters:**

***usercfg***

***url***

***st***

***flavour***

### 6.265.3 Member Function Documentation

#### 6.265.3.1 virtual void Arc::TargetRetriever::GetTargets (TargetGenerator & *mom*, int *targetType*, int *detailLevel*) `[pure virtual]`

Method for collecting targets. Pure virtual method for collecting targets. Implementation depends on the Grid middleware in question and is thus left to the specialized class.

**Parameters:**

***mom*** is the reference to the **TargetGenerator** (p. 440) which has loaded the **TargetRetriever** (p. 443)

***targetType*** is the identificaion of targets to find (0=ExecutionTargets, 1=Grid Jobs)

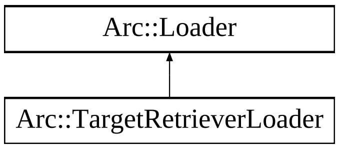
***detailLevel*** is the required level of details (1 = All details, 2 = Limited details)

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

- TargetRetriever.h

## 6.266 Arc::TargetRetrieverLoader Class Reference

`#include <TargetRetriever.h>`Inheritance diagram for Arc::TargetRetrieverLoader::



### Public Member Functions

- **TargetRetrieverLoader** ()
- ∼**TargetRetrieverLoader** ()
- **TargetRetriever** ∗ **load** (const std::string &name, const **UserConfig** &usercfg, const **URL** &url, const ServiceType &st)
- const std::list< **TargetRetriever** ∗ > & **GetTargetRetrievers** () const

### 6.266.1 Detailed Description

Class responsible for loading **TargetRetriever** (p. 443) plugins The **TargetRetriever** (p. 443) objects returned by a **TargetRetrieverLoader** (p. 445) must not be used after the **TargetRetrieverLoader** (p. 445) goes out of scope.

### 6.266.2 Constructor & Destructor Documentation

#### 6.266.2.1 Arc::TargetRetrieverLoader::TargetRetrieverLoader ()

Constructor Creates a new **TargetRetrieverLoader** (p. 445).

#### 6.266.2.2 Arc::TargetRetrieverLoader::∼TargetRetrieverLoader ()

Destructor Calling the destructor destroys all TargetRetrievers loaded by the **TargetRetrieverLoader** (p. 445) instance.

### 6.266.3 Member Function Documentation

#### 6.266.3.1 const std::list<TargetRetriever∗>& Arc::TargetRetrieverLoader::GetTargetRetrievers () const `[inline]`

Retrieve the list of loaded TargetRetrievers.

**Returns:**

A reference to the list of TargetRetrievers.

#### 6.266.3.2 TargetRetriever∗ Arc::TargetRetrieverLoader::load (const std::string & *name*, const UserConfig & *usercfg*, const URL & *url*, const ServiceType & *st*)

Load a new **TargetRetriever** (p. 443)

**Parameters:**

***name*** The name of the **TargetRetriever** (p. 443) to load.

***usercfg*** The **UserConfig** (p. 468) object for the new **TargetRetriever** (p. 443).

***url*** The **URL** (p. 456) used to contact the target.

***st*** specifies service type of the target.

**Returns:**

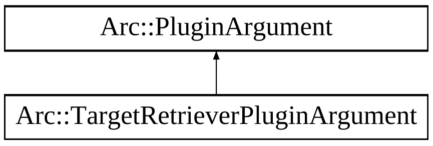
A pointer to the new **TargetRetriever** (p. 443) (NULL on error).

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

- TargetRetriever.h

## 6.267 Arc::TargetRetrieverPluginArgument Class Reference

Inheritance diagram for Arc::TargetRetrieverPluginArgument::

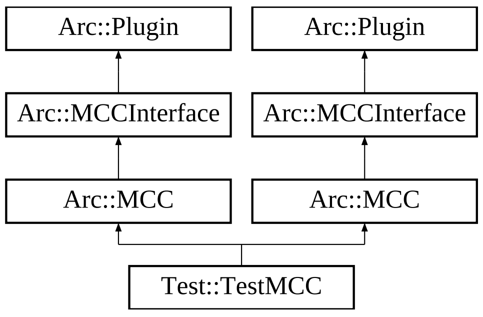


The documentation for this class was generated from the following file:

- TargetRetriever.h

## 6.268 Test::TestMCC Class Reference

Inheritance diagram for Test::TestMCC::

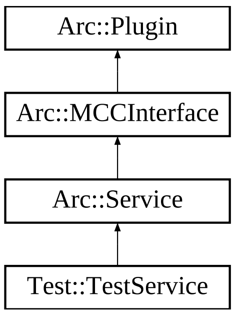


The documentation for this class was generated from the following files:

- loader/TestMCC.h
- message/TestMCC.h

## 6.269 Test::TestService Class Reference

Inheritance diagram for Test::TestService::



### Public Member Functions

- virtual **Arc::MCC_Status process** (**Arc::Message** &request, **Arc::Message** &response)

### 6.269.1 Member Function Documentation

#### 6.269.1.1 virtual Arc::MCC_Status Test::TestService::process (Arc::Message & *request*, Arc::Message & *response*) `[virtual]`

Method for processing of requests and responses. This method is called by preceeding MCC in chain when a request needs to be processed. This method must call similar method of next MCC in chain unless any failure happens. Result returned by call to next MCC should be processed and passed back to previous MCC. In case of failure this method is expected to generate valid error response and return it back to previous MCC without calling the next one.

**Parameters:**

***request*** The request that needs to be processed.

***response*** A Message object that will contain the response of the request when the method returns.

**Returns:**

An object representing the status of the call.

Implements **Arc::MCCInterface** (p. 284).

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

- TestService.h

## 6.270 Arc::ThreadInitializer Class Reference

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

- Thread.h

## 6.271 Arc::ThreadRegistry Class Reference

```
#include <Thread.h>
```

### Public Member Functions

- void **RegisterThread** (void)
- void **UnregisterThread** (void)
- bool **WaitOrCancel** (int timeout)
- bool **WaitForExit** (int timeout=-1)

### 6.271.1 Detailed Description

This class is a set of conditions, mutexes, etc. conveniently exposed to moitor running child threads and to wait till they exit. There are no protections against race conditions. So use it carefully.

### 6.271.2 Member Function Documentation

#### 6.271.2.1 bool Arc::ThreadRegistry::WaitForExit (int *timeout* = `-1`)

Wait for registered threads to exit. Leave after timeout miliseconds if failed. Returns true if all registered threads reported their exit.

#### 6.271.2.2 bool Arc::ThreadRegistry::WaitOrCancel (int *timeout*)

Wait for timeout milliseconds or cancel request. Returns true if cancel request received.

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

- Thread.h

## 6.272 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** &)
- **Time** & **operator=** (const char ∗)
- **Time** & **operator=** (const std::string &)
- 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** ()

### 6.272.1 Detailed Description

A class for storing and manipulating times.

### 6.272.2 Constructor & Destructor Documentation

#### 6.272.2.1 Arc::Time::Time ()

Default constructor. The time is put equal the current time.

#### 6.272.2.2 Arc::Time::Time (const time_t &)

Constructor that takes a time_t variable and stores it.

#### 6.272.2.3 Arc::Time::Time (const std::string &)

Constructor that tries to convert a string into a time_t.

### 6.272.3 Member Function Documentation

#### 6.272.3.1 static TimeFormat Arc::Time::GetFormat () `[static]`

Gets the default format for time strings.

#### 6.272.3.2 time_t Arc::Time::GetTime () const

gets the time

#### 6.272.3.3 Arc::Time::operator std::string () const

Returns a string representation of the time, using the default format.

#### 6.272.3.4 bool Arc::Time::operator!= (const Time &) const

Comparing two **Time** (p. 452) objects.

#### 6.272.3.5 Time Arc::Time::operator+ (const Period &) const

Adding **Time** (p. 452) object with **Period** (p. 335) object.

#### 6.272.3.6 Period Arc::Time::operator- (const Time &) const

Subtracting **Time** (p. 452) object from the other **Time** (p. 452) object.

#### 6.272.3.7 Time Arc::Time::operator- (const Period &) const

Subtracting **Period** (p. 335) object from **Time** (p. 452) object.

#### 6.272.3.8 bool Arc::Time::operator< (const Time &) const

Comparing two **Time** (p. 452) objects.

#### 6.272.3.9 bool Arc::Time::operator<= (const Time &) const

Comparing two **Time** (p. 452) objects.

#### 6.272.3.10 Time& Arc::Time::operator= (const std::string &)

Assignment operator from a string.

#### 6.272.3.11 Time& Arc::Time::operator= (const char ∗)

Assignment operator from a char pointer.

#### 6.272.3.12 Time& Arc::Time::operator= (const Time &)

Assignment operator from a **Time** (p. 452).

#### 6.272.3.13 Time& Arc::Time::operator= (const time_t &)

Assignment operator from a time_t.

#### 6.272.3.14 bool Arc::Time::operator== (const Time &) const

Comparing two **Time** (p. 452) objects.

#### 6.272.3.15 bool Arc::Time::operator> (const Time &) const

Comparing two **Time** (p. 452) objects.

#### 6.272.3.16 bool Arc::Time::operator>= (const Time &) const

Comparing two **Time** (p. 452) objects.

#### 6.272.3.17 static void Arc::Time::SetFormat (const TimeFormat &) `[static]`

Sets the default format for time strings.

#### 6.272.3.18 void Arc::Time::SetTime (const time_t &)

sets the time

#### 6.272.3.19 std::string Arc::Time::str (const TimeFormat & = `time_format`) const

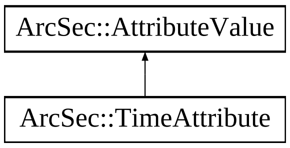
Returns a string representation of the time, using the specified format.

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

- DateTime.h

## 6.273 ArcSec::TimeAttribute Class Reference

`#include <DateTimeAttribute.h>`Inheritance diagram for ArcSec::TimeAttribute::



### Public Member Functions

- virtual std::string **encode** ()
- virtual std::string **getType** ()
- virtual std::string **getId** ()

### 6.273.1 Detailed Description

Format: HHMMSSZ HH:MM:SS HH:MM:SS+HH:MM HH:MM:SSZ

### 6.273.2 Member Function Documentation

#### 6.273.2.1 virtual std::string ArcSec::TimeAttribute::encode () `[virtual]`

encode the value in a string format

Implements **ArcSec::AttributeValue** (p. 78).

#### 6.273.2.2 virtual std::string ArcSec::TimeAttribute::getId () `[inline, virtual]`

Get the AttributeId of the <Attribute>

Implements **ArcSec::AttributeValue** (p. 78).

#### 6.273.2.3 virtual std::string ArcSec::TimeAttribute::getType () `[inline, virtual]`

Get the DataType of the <Attribute>

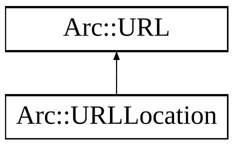
Implements **ArcSec::AttributeValue** (p. 78).

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

- DateTimeAttribute.h

## 6.274 Arc::URL Class Reference

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
- std::map< std::string, std::string > **ParseOptions** (const std::string &optstring, char separator)

## Static Public Member Functions

- static std::string **OptionString** (const std::map< std::string, std::string > &options, char separator)

## 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**
- bool **valid**

## Friends

- std::ostream & **operator<<** (std::ostream &out, const **URL** &u)

## 6.274.1 Member Enumeration Documentation

### 6.274.1.1 enum Arc::URL::Scope

Scope for LDAP URLs

## 6.274.2 Constructor & Destructor Documentation

### 6.274.2.1 Arc::URL::URL ()

Empty constructor. Necessary when the class is part of another class and the like.

#### 6.274.2.2 Arc::URL::URL (const std::string & *url*)

Constructs a new **URL** (p. 456) from a string representation.

#### 6.274.2.3 virtual Arc::URL::~URL () `[virtual]`

**URL** (p. 456) Destructor

### 6.274.3 Member Function Documentation

#### 6.274.3.1 void Arc::URL::AddLDAPAttribute (const std::string & *attribute*)

Adds an LDAP attribute.

#### 6.274.3.2 void Arc::URL::AddOption (const std::string & *option*, const std::string & *value*, bool *overwrite* = `true`)

Adds a **URL** (p. 456) option.

#### 6.274.3.3 static std::string Arc::URL::BaseDN2Path (const std::string &) `[static, protected]`

a private method that converts an ldap basedn to a path.

#### 6.274.3.4 void Arc::URL::ChangeHost (const std::string & *newhost*)

Changes the hostname of the **URL** (p. 456).

#### 6.274.3.5 void Arc::URL::ChangeLDAPFilter (const std::string & *newfilter*)

Changes the LDAP filter.

#### 6.274.3.6 void Arc::URL::ChangeLDAPScope (const Scope *newscope*)

Changes the LDAP scope.

#### 6.274.3.7 void Arc::URL::ChangePath (const std::string & *newpath*)

Changes the path of the **URL** (p. 456).

#### 6.274.3.8 void Arc::URL::ChangePort (int *newport*)

Changes the port of the **URL** (p. 456).

#### 6.274.3.9 void Arc::URL::ChangeProtocol (const std::string & *newprot*)

Changes the protocol of the **URL** (p. 456).

#### 6.274.3.10 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.

#### 6.274.3.11 const std::map<std::string, std::string>& Arc::URL::CommonLocOptions () const

Returns the common location options if any.

#### 6.274.3.12 virtual std::string Arc::URL::ConnectionURL () const `[virtual]`

Returns a string representation with protocol, host and port only

#### 6.274.3.13 std::string Arc::URL::FullPath () const

Returns the path of the **URL** (p. 456) with all options attached.

#### 6.274.3.14 virtual std::string Arc::URL::fullstr () const `[virtual]`

Returns a string representation including options and locations

Reimplemented in **Arc::URLLocation** (p. 465).

#### 6.274.3.15 const std::string& Arc::URL::Host () const

Returns the hostname of the **URL** (p. 456).

#### 6.274.3.16 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.

#### 6.274.3.17 const std::map<std::string, std::string>& Arc::URL::HTTPOptions () const

Returns HTTP options if any.

#### 6.274.3.18 const std::list<std::string>& Arc::URL::LDAPAttributes () const

Returns the LDAP attributes if any.

#### 6.274.3.19 const std::string& Arc::URL::LDAPFilter () const

Returns the LDAP filter.

#### 6.274.3.20 Scope Arc::URL::LDAPScope () const

Returns the LDAP scope.

#### 6.274.3.21 const std::list<URLLocation>& Arc::URL::Locations () const

Returns the locations if any.

#### 6.274.3.22 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.

#### 6.274.3.23 const std::map<std::string, std::string>& Arc::URL::MetaDataOptions () const

Returns metadata options if any.

#### 6.274.3.24 Arc::URL::operator bool () const

Check if instance holds valid **URL** (p. 456)

#### 6.274.3.25 bool Arc::URL::operator< (const URL & *url*) const

Compares one **URL** (p. 456) to another

#### 6.274.3.26 bool Arc::URL::operator== (const URL & *url*) const

Is one **URL** (p. 456) equal to another?

#### 6.274.3.27 const std::string& Arc::URL::Option (const std::string & *option*, const std::string & *undefined* = "") const

Returns the value of a **URL** (p. 456) option.

**Parameters:**

***option*** The option whose value is returned.

***undefined*** This value is returned if the **URL** (p. 456) option is not defined.

#### 6.274.3.28 const std::map<std::string, std::string>& Arc::URL::Options () const

Returns **URL** (p. 456) options if any.

#### 6.274.3.29 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

#### 6.274.3.30 std::map<std::string, std::string> Arc::URL::ParseOptions (const std::string & *optstring*, char *separator*)

Parse a string of options separated by separator into an attribute->value map

#### 6.274.3.31 const std::string& Arc::URL::Passwd () const

Returns the password of the **URL** (p. 456).

#### 6.274.3.32 const std::string& Arc::URL::Path () const

Returns the path of the **URL** (p. 456).

#### 6.274.3.33 static std::string Arc::URL::Path2BaseDN (const std::string &) `[static, protected]`

a private method that converts an ldap path to a basedn.

#### 6.274.3.34 int Arc::URL::Port () const

Returns the port of the **URL** (p. 456).

#### 6.274.3.35 const std::string& Arc::URL::Protocol () const

Returns the protocol of the **URL** (p. 456).

#### 6.274.3.36 virtual std::string Arc::URL::str () const `[virtual]`

Returns a string representation of the **URL** (p. 456).

Reimplemented in **Arc::URLLocation** (p. 465).

#### 6.274.3.37 const std::string& Arc::URL::Username () const

Returns the username of the **URL** (p. 456).

### 6.274.4 Friends And Related Function Documentation

#### 6.274.4.1 std::ostream& operator<< (std::ostream & *out*, const URL & *u*) `[friend]`

Overloaded operator << to print a **URL** (p. 456).

### 6.274.5 Field Documentation

#### 6.274.5.1 std::map<std::string, std::string> Arc::URL::commonlocoptions `[protected]`

common location options for index server URLs.

#### 6.274.5.2 std::string Arc::URL::host `[protected]`

hostname of the url.

#### 6.274.5.3 std::map<std::string, std::string> Arc::URL::httpoptions `[protected]`

HTTP options of the url.

#### 6.274.5.4 std::list<std::string> Arc::URL::ldapattributes `[protected]`

LDAP attributes of the url.

#### 6.274.5.5 std::string Arc::URL::ldapfilter `[protected]`

LDAP filter of the url.

#### 6.274.5.6 Scope Arc::URL::ldapscope `[protected]`

LDAP scope of the url.

#### 6.274.5.7 std::list<URLLocation> Arc::URL::locations `[protected]`

locations for index server URLs.

#### 6.274.5.8 std::map<std::string, std::string> Arc::URL::metadataoptions `[protected]`

Meta data options

#### 6.274.5.9 std::string Arc::URL::passwd `[protected]`

password of the url.

#### 6.274.5.10 std::string Arc::URL::path `[protected]`

the url path.

#### 6.274.5.11 int Arc::URL::port [protected]

portnumber of the url.

#### 6.274.5.12 std::string Arc::URL::protocol [protected]

the url protocol.

#### 6.274.5.13 std::map<std::string, std::string> Arc::URL::urloptions [protected]

options of the url.

#### 6.274.5.14 std::string Arc::URL::username [protected]

username of the url.

#### 6.274.5.15 bool Arc::URL::valid [protected]

flag to describe validity of **URL** (p. 456)

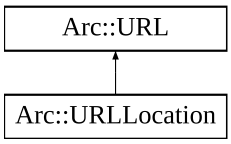
The documentation for this class was generated from the following file:

- **URL.h**

## 6.275 Arc::URLLocation Class Reference

Class to hold a resolved **URL** (p. 456) 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**

### 6.275.1 Detailed Description

Class to hold a resolved **URL** (p. 456) location. It is specific to file indexing service registrations.

### 6.275.2 Constructor & Destructor Documentation

#### 6.275.2.1 Arc::URLLocation::URLLocation (const std::string & *url*)

Creates a **URLLocation** (p. 464) from a string representaion.

#### 6.275.2.2 Arc::URLLocation::URLLocation (const std::string & *url*, const std::string & *name*)

Creates a **URLLocation** (p. 464) from a string representaion and a name.

#### 6.275.2.3 Arc::URLLocation::URLLocation (const URL & *url*)

Creates a **URLLocation** (p. 464) from a **URL** (p. 456).

#### 6.275.2.4 Arc::URLLocation::URLLocation (const URL & *url*, const std::string & *name*)

Creates a **URLLocation** (p. 464) from a **URL** (p. 456) and a name.

#### 6.275.2.5 Arc::URLLocation::URLLocation (const std::map< std::string, std::string > & *options*, const std::string & *name*)

Creates a **URLLocation** (p. 464) from options and a name.

#### 6.275.2.6 virtual Arc::URLLocation::~URLLocation () `[virtual]`

**URLLocation** (p. 464) destructor.

### 6.275.3 Member Function Documentation

#### 6.275.3.1 virtual std::string Arc::URLLocation::fullstr () const `[virtual]`

Returns a string representation including options and locations

Reimplemented from **Arc::URL** (p. 459).

#### 6.275.3.2 const std::string& Arc::URLLocation::Name () const

Returns the **URLLocation** (p. 464) name.

#### 6.275.3.3 virtual std::string Arc::URLLocation::str () const `[virtual]`

Returns a string representation of the **URLLocation** (p. 464).

Reimplemented from **Arc::URL** (p. 461).

### 6.275.4 Field Documentation

#### 6.275.4.1 std::string Arc::URLLocation::name `[protected]`

the **URLLocation** (p. 464) name as registered in the indexing service.

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

- **URL.h**

## 6.276 Arc::URLMap Class Reference

### Data Structures

- class **map_entry**

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

- URLMap.h

## 6.277 Arc::User Class Reference

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

- User.h

## 6.278 Arc::UserConfig Class Reference

User configuration class

`#include <UserConfig.h>`

### Public Member Functions

- **UserConfig** (**initializeCredentialsType** initializeCredentials=**initializeCredentialsType**())
- **UserConfig** (const std::string &conffile, **initializeCredentialsType** initializeCredentials=**initializeCredentialsType**(), bool loadSysConfig=true)
- **UserConfig** (const std::string &conffile, const std::string &jfile, **initializeCredentialsType** initializeCredentials=**initializeCredentialsType**(), bool loadSysConfig=true)
- **UserConfig** (const long int &ptraddr)
- void **InitializeCredentials** ()
- bool **CredentialsFound** () const
- bool **LoadConfigurationFile** (const std::string &conffile, bool ignoreJobListFile=true)
- void **ApplyToConfig** (**BaseConfig** &ccfg) const
- **operator bool** () const
- bool **operator!** () const
- bool **JobListFile** (const std::string &path)
- const std::string & **JobListFile** () const
- bool **AddServices** (const std::list< std::string > &services, ServiceType st)
- bool **AddServices** (const std::list< std::string > &selected, const std::list< std::string > &rejected, ServiceType st)
- const URLListMap & **GetSelectedServices** (ServiceType st) const
- const URLListMap & **GetRejectedServices** (ServiceType st) const
- void **ClearSelectedServices** ()
- void **ClearSelectedServices** (ServiceType st)
- void **ClearRejectedServices** ()
- void **ClearRejectedServices** (ServiceType st)
- bool **Timeout** (int newTimeout)
- int **Timeout** () const
- bool **Verbosity** (const std::string &newVerbosity)
- const std::string & **Verbosity** () const
- bool **Broker** (const std::string &name)
- bool **Broker** (const std::string &name, const std::string &argument)
- const std::pair< std::string, std::string > & **Broker** () const
- bool **Bartender** (const std::vector< **URL** > &urls)
- void **AddBartender** (const **URL** &url)
- const std::vector< **URL** > & **Bartender** () const
- bool **VOMSServerPath** (const std::string &path)
- const std::string & **VOMSServerPath** () const
- bool **UserName** (const std::string &name)
- const std::string & **UserName** () const
- bool **Password** (const std::string &newPassword)
- const std::string & **Password** () const
- bool **ProxyPath** (const std::string &newProxyPath)
- const std::string & **ProxyPath** () const
- bool **CertificatePath** (const std::string &newCertificatePath)

- const std::string & **CertificatePath** () const
- bool **KeyPath** (const std::string &newKeyPath)
- const std::string & **KeyPath** () const
- bool **KeyPassword** (const std::string &newKeyPassword)
- const std::string & **KeyPassword** () const
- bool **KeySize** (int newKeySize)
- int **KeySize** () const
- bool **CACertificatePath** (const std::string &newCACertificatePath)
- const std::string & **CACertificatePath** () const
- bool **CACertificatesDirectory** (const std::string &newCACertificatesDirectory)
- const std::string & **CACertificatesDirectory** () const
- bool **CertificateLifeTime** (const **Period** &newCertificateLifeTime)
- const **Period** & **CertificateLifeTime** () const
- bool **SLCS** (const **URL** &newSLCS)
- const **URL** & **SLCS** () const
- bool **StoreDirectory** (const std::string &newStoreDirectory)
- const std::string & **StoreDirectory** () const
- bool **IdPName** (const std::string &name)
- const std::string & **IdPName** () const
- bool **OverlayFile** (const std::string &path)
- const std::string & **OverlayFile** () const

## Static Public Attributes

- static const std::string **ARCUSERDIRECTORY**
- static const std::string **SYSCONFIG**
- static const std::string **DEFAULTCONFIG**
- static const std::string **EXAMPLECONFIG**
- static const int **DEFAULT_TIMEOUT** = 20
- static const std::string **DEFAULT_BROKER**

## 6.278.1 Detailed Description

User configuration class This class provides a container for a selection of various attributes/parameters which can be configured to needs of the user, and can be read by implementing instances or programs. The class can be used in two ways. One can create a object from a configuration file, or simply set the desired attributes by using the setter method, associated with every setable attribute. The list of attributes which can be configured in this class are:

- certificatepath / **CertificatePath(const std::string&)** (p. 478)
- keypath / **KeyPath(const std::string&)** (p. 483)
- proxypath / **ProxyPath(const std::string&)** (p. 488)
- cacertificatesdirectory / **CACertificatesDirectory(const std::string&)** (p. 476)
- cacertificatepath / **CACertificatePath(const std::string&)** (p. 476)
- timeout / **Timeout(int)** (p. 489)
- joblist / **JobListFile(const std::string&)** (p. 482)

- defaultservices / **AddServices(const std::list<std::string>&, const std::list<std::string>&, ServiceType)** (p. 472)
- rejectservices / **AddServices(const std::list<std::string>&, const std::list<std::string>&, ServiceType)** (p. 472)
- verbosity / **Verbosity(const std::string&)** (p. 491)
- brokername / **Broker(const std::string&)** (p. 475) or **Broker(const std::string&, const std::string&)** (p. 475)
- brokerarguments / **Broker(const std::string&)** (p. 475) or **Broker(const std::string&, const std::string&)** (p. 475)
- bartender / Bartender(const std::list<URL>&)
- vomsserverpath / **VOMSServerPath(const std::string&)** (p. 491)
- username / **UserName(const std::string&)** (p. 490)
- password / **Password(const std::string&)** (p. 487)
- keypassword / **KeyPassword(const std::string&)** (p. 483)
- keysize / **KeySize(int)** (p. 484)
- certificatelifetime / **CertificateLifeTime(const Period&)** (p. 477)
- slcs / **SLCS(const URL&)** (p. 488)
- storedirectory / **StoreDirectory(const std::string&)** (p. 489)
- idpname / **IdPName(const std::string&)** (p. 480)

where the first term is the name of the attribute used in the configuration file, and the second term is the associated setter method (for more information about a given attribute see the description of the setter method).

The configuration file should have a INI-style format and the **IniConfig** (p. 241) class will thus be used to parse the file. The above mentioned attributes should be placed in the common section. Another section is also valid in the configuration file, which is the alias section. Here it is possible to define aliases representing one or multiple services. These aliases can be used in the **AddServices(const std::list<std::string>&, ServiceType)** (p. 473) and **AddServices(const std::list<std::string>&, const std::list<std::string>&, ServiceType)** (p. 472) methods.

The **UserConfig** (p. 468) class also provides a method **InitializeCredentials()** (p. 481) for locating user credentials by searching in different standard locations. The **CredentialsFound()** (p. 479) method can be used to test if locating the credentials succeeded.

## 6.278.2 Constructor & Destructor Documentation

### 6.278.2.1 Arc::UserConfig::UserConfig (initializeCredentialsType *initializeCredentials* = initializeCredentialsType ())

Create a **UserConfig** (p. 468) object. The **UserConfig** (p. 468) object created by this constructor initializes only default values, and if specified by the *initializeCredentials* boolean credentials will be tried initialized using the **InitializeCredentials()** (p. 481) method. The object is only non-valid if initialization of credentials fails which can be checked with the **operator bool()** (p. 486) method.

**Parameters:**

***initializeCredentials*** is a optional boolean indicating if the **InitializeCredentials()** (p. 481) method should be invoked, the default is `true`.

**See also:**

**InitializeCredentials()** (p. 481)
**operator bool()** (p. 486)

#### 6.278.2.2 Arc::UserConfig::UserConfig (const std::string & *conffile*, initializeCredentialsType *initializeCredentials* = initializeCredentialsType `()`, bool *loadSysConfig* = `true`)

Create a **UserConfig** (p. 468) object. The **UserConfig** (p. 468) object created by this constructor will, if specified by the *loadSysConfig* boolean, first try to load the system configuration file by invoking the **LoadConfigurationFile()** (p. 484) method, and if this fails a WARNING is reported. Then the configuration file passed will be tried loaded using the before mentioned method, and if this fails an ERROR is reported, and the created object will be non-valid. Note that if the passed file path is empty the example configuration will be tried copied to the default configuration file path specified by DEFAULTCONFIG. If the example file cannot be copied one or more WARNING messages will be reported and no configration will be loaded. If loading the configurations file succeeded and if *initializeCredentials* is `true` then credentials will be initialized using the **InitializeCredentials()** (p. 481) method, and if no valid credentials are found the created object will be non-valid.

**Parameters:**

***conffile*** is the path to a INI-configuration file.

***initializeCredentials*** is a boolean indicating if credentials should be initialized, the default is `true`.

***loadSysConfig*** is a boolean indicating if the system configuration file should be loaded aswell, the default is `true`.

**See also:**

**LoadConfigurationFile(const std::string&, bool)** (p. 484)
**InitializeCredentials()** (p. 481)
**operator bool()** (p. 486)
**SYSCONFIG** (p. 492)
**EXAMPLECONFIG** (p. 492)

#### 6.278.2.3 Arc::UserConfig::UserConfig (const std::string & *conffile*, const std::string & *jfile*, initializeCredentialsType *initializeCredentials* = initializeCredentialsType `()`, bool *loadSysConfig* = `true`)

Create a **UserConfig** (p. 468) object. The **UserConfig** (p. 468) object created by this constructor does only differ from the UserConfig(const std::string&, bool, bool) constructor in that it is possible to pass the path of the job list file directly to this constructor. If the job list file *joblistfile* is empty, the behaviour of this constructor is exactly the same as the before mentioned, otherwise the job list file will be initilized by invoking the setter method **JobListFile(const std::string&)** (p. 482). If it fails the created object will be non-valid, otherwise the specified configuration file *conffile* will be loaded with the *ignoreJobListFile* argument set to `true`.

**Parameters:**

***conffile*** is the path to a INI-configuration file

***jfile*** is the path to a (non-)existing job list file.

***initializeCredentials*** is a boolean indicating if credentials should be initialized, the default is `true`.

***loadSysConfig*** is a boolean indicating if the system configuration file should be loaded aswell, the default is `true`.

**See also:**

**JobListFile(const std::string&)** (p. 482)
**LoadConfigurationFile(const std::string&, bool)** (p. 484)
**InitializeCredentials**() (p. 481)
**operator bool**() (p. 486)

#### 6.278.2.4 Arc::UserConfig::UserConfig (const long int & *ptraddr*)

Language binding constructor. The passed long int should be a pointer address to a **UserConfig** (p. 468) object, and this address is then casted into this **UserConfig** (p. 468) object.

**Parameters:**

***ptraddr*** is an memory address to a **UserConfig** (p. 468) object.

### 6.278.3 Member Function Documentation

#### 6.278.3.1 void Arc::UserConfig::AddBartender (const URL & *url*) `[inline]`

Set bartenders, used to contact Chelonia. Takes as input a Bartender **URL** (p. 456) and adds this to the list of bartenders.

**Parameters:**

***url*** is a **URL** (p. 456) to be added to the list of bartenders.

**See also:**

Bartender(const std::list<URL>&)
**Bartender() const** (p. 474)

#### 6.278.3.2 bool Arc::UserConfig::AddServices (const std::list< std::string > & *selected*, const std::list< std::string > & *rejected*, ServiceType *st*)

Add selected and rejected services. The only diffence in behaviour of this method compared to the **AddServices(const std::list<std::string>&, ServiceType)** (p. 473) method is the input parameters and the format these parameters should follow. Instead of having an optional '-' in front of the string selected and rejected services should be specified in the two different arguments.

Two attributes are indirectly associated with this setter method 'defaultservices' and 'rejectservices'. The values specified with the 'defaultservices' attribute will be added to the list of selected services, and likewise with the 'rejectservices' attribute.

**Parameters:**

***selected*** is a list of services which will be added to the selected services of this object.

***rejected*** is a list of services which will be added to the rejected services of this object.

***st*** specifies the ServiceType of the services to add.

**Returns:**

This method return `false` in case an alias cannot be resolved. In any other case `true` is returned.

**See also:**

**AddServices(const std::list<std::string>&, ServiceType)** (p. 473)
**GetSelectedServices()** (p. 480)
**GetRejectedServices()** (p. 479)
**ClearSelectedServices()** (p. 479)
**ClearRejectedServices()** (p. 478)
**LoadConfigurationFile()** (p. 484)

#### 6.278.3.3 bool Arc::UserConfig::AddServices (const std::list< std::string > & *services*, ServiceType *st*)

Add selected and rejected services. This method adds selected services and adds services to reject from the specified list *services*, which contains string objects. The syntax of a single element in the list must be expressed in the following two formats:

$$[-] < flavour >:< service\_url > |[-] < alias >$$

where the optional '-' indicate that the service should be added to the private list of services to reject. In the first format the <flavour> part indicates the type of ACC plugin to use when contacting the service, which is specified by the **URL** (p. 456) <service_url>, and in the second format the <alias> part specifies a alias defined in a parsed configuration file, note that the alias must not contain any of the charaters ':', '.', ' ' or '\t'. If a alias cannot be resolved an ERROR will be reported to the logger and the method will return false. If a element in the list *services* cannot be parsed an ERROR will be reported, and the element is skipped.

Two attributes are indirectly associated with this setter method 'defaultservices' and 'rejectservices'. The values specified with the 'defaultservices' attribute will be added to the list of selected services, and likewise with the 'rejectservices' attribute.

**Parameters:**

***services*** is a list of services to either select or reject.

***st*** indicates the type of the specfied services.

**Returns:**

This method returns `false` in case an alias cannot be resolved. In any other case `true` is returned.

**See also:**

AddServices(const std::string&, const std::string&, ServiceType)
**GetSelectedServices()** (p. 480)
**GetRejectedServices()** (p. 479)
**ClearSelectedServices()** (p. 479)
**ClearRejectedServices()** (p. 478)
**LoadConfigurationFile()** (p. 484)

#### 6.278.3.4 void Arc::UserConfig::ApplyToConfig (BaseConfig & *ccfg*) const

Apply credentials to **BaseConfig** (p. 84). This methods sets the **BaseConfig** (p. 84) credentials to the credentials contained in this object. It also passes user defined configuration overlay if any.

**See also:**

**InitializeCredentials()** (p. 481)
**CredentialsFound()** (p. 479)
**BaseConfig** (p. 84)

**Parameters:**

***ccfg*** a **BaseConfig** (p. 84) object which will configured with the credentials of this object.

#### 6.278.3.5 const std::vector<URL>& Arc::UserConfig::Bartender () const `[inline]`

Get bartenders. Returns a list of Bartender URLs

**Returns:**

The list of bartender **URL** (p. 456) objects is returned.

**See also:**

Bartender(const std::list<URL>&)
**AddBartender(const URL&)** (p. 472)

#### 6.278.3.6 bool Arc::UserConfig::Bartender (const std::vector< URL > & *urls*) `[inline]`

Set bartenders, used to contact Chelonia. Takes as input a vector of Bartender URLs.

The attribute associated with this setter method is 'bartender'.

**Parameters:**

***urls*** is a list of **URL** (p. 456) object to be set as bartenders.

**Returns:**

This method always returns `true`.

**See also:**

**AddBartender(const URL&)** (p. 472)
**Bartender() const** (p. 474)

#### 6.278.3.7 const std::pair<std::string, std::string>& Arc::UserConfig::Broker () const `[inline]`

Get the broker and corresponding arguments. The returned pair contains the broker name as the first component and the argument as the second.

**See also:**

**Broker(const std::string&)** (p. 475)
**Broker(const std::string&, const std::string&)** (p. 475)
**DEFAULT_BROKER** (p. 492)

#### 6.278.3.8 bool Arc::UserConfig::Broker (const std::string & *name*, const std::string & *argument*) `[inline]`

Set broker to use in target matching. As opposed to the **Broker(const std::string&)** (p. 475) method this method sets broker name and arguments directly from the passed two arguments.

Two attributes are associated with this setter method 'brokername' and 'brokerarguments'.

**Parameters:**

***name*** is the name of the broker.

***argument*** is the arguments of the broker.

**Returns:**

This method always returns `true`.

**See also:**

**Broker** (p. 87)
**Broker(const std::string&)** (p. 475)
**Broker() const** (p. 474)
**DEFAULT_BROKER** (p. 492)

#### 6.278.3.9 bool Arc::UserConfig::Broker (const std::string & *name*)

Set broker to use in target matching. The string passed to this method should be in the format:

$$< name > [:< argument >]$$

where the <name> is the name of the broker and cannot contain any ':', and the optional <argument> should contain arguments which should be passed to the broker.

Two attributes are associated with this setter method 'brokername' and 'brokerarguments'.

**Parameters:**

***name*** the broker name and argument specified in the format given above.

**Returns:**

This method allways returns `true`.

**See also:**

**Broker** (p. 87)
**Broker(const std::string&, const std::string&)** (p. 475)
**Broker() const** (p. 474)
**DEFAULT_BROKER** (p. 492)

#### 6.278.3.10 const std::string& Arc::UserConfig::CACertificatePath () const `[inline]`

Get path to CA-certificate. Retrieve the path to the file containing CA-certificate. This configuration parameter is deprecated.

**Returns:**

The path to the CA-certificate is returned.

**See also:**

**CACertificatePath(const std::string&)** (p. 476)

#### 6.278.3.11 bool Arc::UserConfig::CACertificatePath (const std::string & *newCACertificatePath*) `[inline]`

Set CA-certificate path. The path to the file containing CA-certificate will be set when calling this method. This configuration parameter is deprecated - use CACertificatesDirectory instead. Only arcslcs uses it.

The attribute associated with this setter method is 'cacertificatepath'.

**Parameters:**

***newCACertificatePath*** is the path to the CA-certificate.

**Returns:**

This method always returns `true`.

**See also:**

**CACertificatePath() const** (p. 476)

#### 6.278.3.12 const std::string& Arc::UserConfig::CACertificatesDirectory () const `[inline]`

Get path to CA-certificate directory. Retrieve the path to the CA-certificate directory.

**Returns:**

The path to the CA-certificate directory is returned.

**See also:**

**InitializeCredentials()** (p. 481)
**CredentialsFound() const** (p. 479)
**CACertificatesDirectory(const std::string&)** (p. 476)

#### 6.278.3.13 bool Arc::UserConfig::CACertificatesDirectory (const std::string & *newCACertificatesDirectory*) `[inline]`

Set path to CA-certificate directory. The path to the directory containing CA-certificates will be set when calling this method. Note that the **InitializeCredentials()** (p. 481) method will also try to set this path, by searching in different locations.

The attribute associated with this setter method is 'cacertificatesdirectory'.

**Parameters:**

***newCACertificatesDirectory*** is the path to the CA-certificate directory.

**Returns:**

This method always returns `true`.

**See also:**

**InitializeCredentials()** (p. 481)
**CredentialsFound() const** (p. 479)
**CACertificatesDirectory() const** (p. 476)

#### 6.278.3.14 const Period& Arc::UserConfig::CertificateLifeTime () const `[inline]`

Get certificate life time. Gets lifetime of user certificate which will be obtained from Short Lived Credentials **Service** (p. 409).

**Returns:**

The certificate life time is returned as a **Period** (p. 335) object.

**See also:**

**CertificateLifeTime(const Period&)** (p. 477)

#### 6.278.3.15 bool Arc::UserConfig::CertificateLifeTime (const Period & *newCertificateLifeTime*) `[inline]`

Set certificate life time. Sets lifetime of user certificate which will be obtained from Short Lived Credentials **Service** (p. 409).

The attribute associated with this setter method is 'certificatelifetime'.

**Parameters:**

***newCertificateLifeTime*** is the life time of a certificate, as a **Period** (p. 335) object.

**Returns:**

This method always returns `true`.

**See also:**

**CertificateLifeTime() const** (p. 477)

#### 6.278.3.16 const std::string& Arc::UserConfig::CertificatePath () const `[inline]`

Get path to certificate. The path to the cerficate is returned when invoking this method.

**Returns:**

The certificate path is returned.

**See also:**

**InitializeCredentials()** (p. 481)
**CredentialsFound() const** (p. 479)
**CertificatePath(const std::string&)** (p. 478)
**KeyPath() const** (p. 483)

#### 6.278.3.17 bool Arc::UserConfig::CertificatePath (const std::string & *newCertificatePath*) `[inline]`

Set path to certificate. The path to user certificate will be set by this method. The path to the correcsponding key can be set with the **KeyPath(const std::string&)** (p. 483) method. Note that the **InitializeCredentials()** (p. 481) method will also try to set this path, by searching in different locations.

The attribute associated with this setter method is ’certificatepath’.

**Parameters:**

*newCertificatePath* is the path to the new certificate.

**Returns:**

This method always returns `true`.

**See also:**

**InitializeCredentials()** (p. 481)
**CredentialsFound() const** (p. 479)
**CertificatePath() const** (p. 477)
**KeyPath(const std::string&)** (p. 483)

#### 6.278.3.18 void Arc::UserConfig::ClearRejectedServices (ServiceType *st*)

Clear rejected services with specified ServiceType. Calling this method will cause the internally stored rejected services with the ServiceType *st* to be cleared.

**See also:**

**ClearRejectedServices()** (p. 478)
**ClearSelectedServices(ServiceType)** (p. 479)
**AddServices(const std::list<std::string>&, ServiceType)** (p. 473)
**AddServices(const std::list<std::string>&, const std::list<std::string>&, ServiceType)** (p. 472)
**GetRejectedServices()** (p. 479)

#### 6.278.3.19 void Arc::UserConfig::ClearRejectedServices ()

Clear selected services. Calling this method will cause the internally stored rejected services to be cleared.

**See also:**

**ClearRejectedServices(ServiceType)** (p. 478)
**ClearSelectedServices()** (p. 479)
**AddServices(const std::list<std::string>&, ServiceType)** (p. 473)
**AddServices(const std::list<std::string>&, const std::list<std::string>&, ServiceType)** (p. 472)
**GetRejectedServices()** (p. 479)

#### 6.278.3.20 void Arc::UserConfig::ClearSelectedServices (ServiceType *st*)

Clear selected services with specified ServiceType. Calling this method will cause the internally stored selected services with the ServiceType *st* to be cleared.

**See also:**

**ClearSelectedServices()** (p. 479)
**ClearRejectedServices(ServiceType)** (p. 478)
**AddServices(const std::list<std::string>&, ServiceType)** (p. 473)
**AddServices(const std::list<std::string>&, const std::list<std::string>&, ServiceType)** (p. 472)
**GetSelectedServices()** (p. 480)

#### 6.278.3.21 void Arc::UserConfig::ClearSelectedServices ()

Clear selected services. Calling this method will cause the internally stored selected services to be cleared.

**See also:**

**ClearSelectedServices(ServiceType)** (p. 479)
**ClearRejectedServices()** (p. 478)
**AddServices(const std::list<std::string>&, ServiceType)** (p. 473)
**AddServices(const std::list<std::string>&, const std::list<std::string>&, ServiceType)** (p. 472)
**GetSelectedServices()** (p. 480)

#### 6.278.3.22 bool Arc::UserConfig::CredentialsFound () const `[inline]`

Validate credential location. Valid credentials consists of a combination of a path to existing CA-certificate directory and either a path to existing proxy or a path to existing user key/certificate pair. If valid credentials are found this method returns `true`, otherwise `false` is returned.

**Returns:**

`true` if valid credentials are found, otherwise `false`.

**See also:**

**InitializeCredentials()** (p. 481)

#### 6.278.3.23 const URLListMap& Arc::UserConfig::GetRejectedServices (ServiceType *st*) const

Get rejected services. Get the rejected services with the ServiceType specified by *st*.

**Parameters:**

*st* specifies which ServiceType should be returned by the method.

**Returns:**

The rejected services is returned.

**See also:**

**AddServices(const std::list<std::string>&, ServiceType)** (p. 473)
**AddServices(const std::list<std::string>&, const std::list<std::string>&, ServiceType)** (p. 472)
GetSelectedServices(ServiceType)
**ClearRejectedServices()** (p. 478)

#### 6.278.3.24 const URLListMap& Arc::UserConfig::GetSelectedServices (ServiceType *st*) const

Get selected services. Get the selected services with the ServiceType specified by *st*.

**Parameters:**

***st*** specifies which ServiceType should be returned by the method.

**Returns:**

The selected services is returned.

**See also:**

**AddServices(const std::list<std::string>&, ServiceType)** (p. 473)
**AddServices(const std::list<std::string>&, const std::list<std::string>&, ServiceType)** (p. 472)
**GetRejectedServices(ServiceType) const** (p. 479)
**ClearSelectedServices()** (p. 479)

#### 6.278.3.25 const std::string& Arc::UserConfig::IdPName () const `[inline]`

Get IdP name. Gets Identity Provider name (Shibboleth) to which user belongs.

**Returns:**

The IdP name

**See also:**

**IdPName(const std::string&)** (p. 480)

#### 6.278.3.26 bool Arc::UserConfig::IdPName (const std::string & *name*) `[inline]`

Set IdP name. Sets Identity Provider name (Shibboleth) to which user belongs. It is used for contacting Short Lived Certificate **Service** (p. 409).

The attribute associated with this setter method is 'idpname'.

**Parameters:**

***name*** is the new IdP name.

**Returns:**

This method always returns `true`.

**See also:**

#### 6.278.3.27 void Arc::UserConfig::InitializeCredentials ()

Initialize user credentials. The location of the user credentials will be tried located when calling this method and stored internally when found. The method searches in different locations. First the user proxy or the user key/certificate pair is tried located in the following order:

- Proxy path specified by the environment variable X509_USER_PROXY
- Key/certificate path specified by the environment X509_USER_KEY and X509_USER_CERT
- Proxy path specified in either configuration file passed to the contructor or explicitly set using the setter method **ProxyPath(const std::string&)** (p. 488)
- Key/certificate path specified in either configuration file passed to the constructor or explicitly set using the setter methods **KeyPath(const std::string&)** (p. 483) and **CertificatePath(const std::string&)** (p. 478)
- ProxyPath with file name x509up_u concatenated with the user ID located in the OS temporary directory.

If the proxy or key/certificate pair have been explicitly specified only the specified path(s) will be tried, and if not found a ERROR is reported. If the proxy or key/certificate have not been specified and it is not located in the temporary directory a WARNING will be reported and the host key/certificate pair is tried and then the Globus key/certificate pair and a ERROR will be reported if not found in any of these locations.

Together with the proxy and key/certificate pair, the path to the directory containing CA certificates is also tried located when invoking this method. The directory will be tried located in the following order:

- Path specified by the X509_CERT_DIR environment variable.
- Path explicitly specified either in a parsed configuration file using the cacertficatecirectory or by using the setter method **CACertificatesDirectory()** (p. 476).
- Path created by concatenating the output of User::Home() with '.globus' and 'certificates' separated by the directory delimeter.
- Path created by concatenating the output of Glib::get_home_dir() with '.globus' and 'certificates' separated by the directory delimeter.
- Path created by concatenating the output of **ArcLocation::Get()** (p. 68), with 'etc' and 'certificates' separated by the directory delimeter.
- Path created by concatenating the output of **ArcLocation::Get()** (p. 68), with 'etc', 'grid-security' and 'certificates' separated by the directory delimeter.
- Path created by concatenating the output of **ArcLocation::Get()** (p. 68), with 'share' and 'certificates' separated by the directory delimeter.
- Path created by concatenating 'etc', 'grid-security' and 'certificates' separated by the directory delimeter.

If the CA certificate directory have explicitly been specified and the directory does not exist a ERROR is reported. If none of the directories above does not exist a ERROR is reported.

**See also:**

**CredentialsFound()** (p. 479)

**ProxyPath(const std::string&)** (p. 488)
**KeyPath(const std::string&)** (p. 483)
**CertificatePath(const std::string&)** (p. 478)
**CACertificatesDirectory(const std::string&)** (p. 476)

#### 6.278.3.28 const std::string& Arc::UserConfig::JobListFile () const `[inline]`

Get a reference to the path of the job list file. The job list file is used to store and fetch information about submitted computing jobs to computing services. This method will return the path to the specified job list file.

**Returns:**

The path to the job list file is returned.

**See also:**

**JobListFile(const std::string&)** (p. 482)

#### 6.278.3.29 bool Arc::UserConfig::JobListFile (const std::string & *path*)

Set path to job list file. The method takes a path to a file which will be used as the job list file for storing and reading job information. If the specified path *path* does not exist a empty job list file will be tried created. If creating the job list file in any way fails *false* will be returned and a ERROR message will be reported. Otherwise *true* is returned. If the directory containing the file does not exist, it will be tried created. The method will also return *false* if the file is not a regular file.

The attribute associated with this setter method is 'joblist'.

**Parameters:**

***path*** the path to the job list file.

**Returns:**

If the job list file is a regular file or if it can be created *true* is returned, otherwise *false* is returned.

**See also:**

**JobListFile() const** (p. 482)

#### 6.278.3.30 const std::string& Arc::UserConfig::KeyPassword () const `[inline]`

Get password for generated key. Get password to be used to encode private key of credentials obtained from Short Lived Credentials **Service** (p. 409).

**Returns:**

The key password is returned.

**See also:**

**KeyPassword(const std::string&)** (p. 483)
**KeyPath() const** (p. 483)
**KeySize() const** (p. 484)

#### 6.278.3.31 bool Arc::UserConfig::KeyPassword (const std::string & *newKeyPassword*) [inline]

Set password for generated key. Set password to be used to encode private key of credentials obtained from Short Lived Credentials **Service** (p. 409).

The attribute associated with this setter method is 'keypassword'.

**Parameters:**

*newKeyPassword* is the new password to the key.

**Returns:**

This method always returns `true`.

**See also:**

**KeyPassword() const** (p. 482)
**KeyPath(const std::string&)** (p. 483)
**KeySize(int)** (p. 484)

#### 6.278.3.32 const std::string& Arc::UserConfig::KeyPath () const [inline]

Get path to key. The path to the key is returned when invoking this method.

**Returns:**

The path to the user key is returned.

**See also:**

**InitializeCredentials()** (p. 481)
**CredentialsFound() const** (p. 479)
**KeyPath(const std::string&)** (p. 483)
**CertificatePath() const** (p. 477)
**KeyPassword() const** (p. 482)
**KeySize() const** (p. 484)

#### 6.278.3.33 bool Arc::UserConfig::KeyPath (const std::string & *newKeyPath*) [inline]

Set path to key. The path to user key will be set by this method. The path to the corresponding certificate can be set with the **CertificatePath(const std::string&)** (p. 478) method. Note that the **InitializeCredentials()** (p. 481) method will also try to set this path, by searching in different locations.

The attribute associated with this setter method is 'keypath'.

**Parameters:**

*newKeyPath* is the path to the new key.

**Returns:**

This method always returns `true`.

**See also:**

**InitializeCredentials()** (p. 481)
**CredentialsFound() const** (p. 479)
**KeyPath() const** (p. 483)
**CertificatePath(const std::string&)** (p. 478)
**KeyPassword(const std::string&)** (p. 483)
**KeySize(int)** (p. 484)

#### 6.278.3.34 int Arc::UserConfig::KeySize () const `[inline]`

Get key size. Get size/strengt of private key of credentials obtained from Short Lived Credentials **Service** (p. 409).

**Returns:**

The key size, as an integer, is returned.

**See also:**

**KeySize(int)** (p. 484)
**KeyPath() const** (p. 483)
**KeyPassword() const** (p. 482)

#### 6.278.3.35 bool Arc::UserConfig::KeySize (int *newKeySize*) `[inline]`

Set key size. Set size/strengt of private key of credentials obtained from Short Lived Credentials **Service** (p. 409).

The attribute associated with this setter method is 'keysize'.

**Parameters:**

***newKeySize*** is the size, an an integer, of the key.

**Returns:**

This method always returns `true`.

**See also:**

**KeySize() const** (p. 484)
**KeyPath(const std::string&)** (p. 483)
**KeyPassword(const std::string&)** (p. 483)

#### 6.278.3.36 bool Arc::UserConfig::LoadConfigurationFile (const std::string & *conffile*, bool *ignoreJobListFile* = `true`)

Load specified configuration file. The configuration file passed is parsed by this method by using the **IniConfig** (p. 241) class. If the parsing is unsuccessful a WARNING is reported.

The format of the configuration file should follow that of INI, and every attribute present in the file is only allowed once, if otherwise a WARNING will be reported. The file can contain at most two sections, one named common and the other name alias. If other sections exist a WARNING will be reported. Only the following attributes is allowed in the common section of the configuration file:

- certificatepath (**CertificatePath(const std::string&)** (p. 478))
- keypath (**KeyPath(const std::string&)** (p. 483))
- proxypath (**ProxyPath(const std::string&)** (p. 488))
- cacertificatesdirectory (**CACertificatesDirectory(const std::string&)** (p. 476))
- cacertificatepath (**CACertificatePath(const std::string&)** (p. 476))
- timeout (**Timeout(int)** (p. 489))
- joblist (**JobListFile(const std::string&)** (p. 482))
- defaultservices (**AddServices(const std::list<std::string>&, const std::list<std::string>&, ServiceType)** (p. 472))
- rejectservices (**AddServices(const std::list<std::string>&, const std::list<std::string>&, ServiceType)** (p. 472))
- verbosity (**Verbosity(const std::string&)** (p. 491))
- brokername (**Broker(const std::string&)** (p. 475) or **Broker(const std::string&, const std::string&)** (p. 475))
- brokerarguments (**Broker(const std::string&)** (p. 475) or **Broker(const std::string&, const std::string&)** (p. 475))
- bartender (Bartender(const std::list<URL>&))
- vomsserverpath (**VOMSServerPath(const std::string&)** (p. 491))
- username (**UserName(const std::string&)** (p. 490))
- password (**Password(const std::string&)** (p. 487))
- keypassword (**KeyPassword(const std::string&)** (p. 483))
- keysize (**KeySize(int)** (p. 484))
- certificatelifetime (**CertificateLifeTime(const Period&)** (p. 477))
- slcs (**SLCS(const URL&)** (p. 488))
- storedirectory (**StoreDirectory(const std::string&)** (p. 489))
- idpname (**IdPName(const std::string&)** (p. 480))

where the method in parentheses is the associated setter method. If other attributes exist in the common section a WARNING will be reported for each of these attributes. In the alias section aliases can be defined, and should represent a selection of services. The alias can then refered to by input to the **AddServices(const std::list<std::string>&, ServiceType)** (p. 473) and **AddServices(const std::list<std::string>&, const std::list<std::string>&, ServiceType)** (p. 472) methods. An alias can not contain any of the characters '.', ':', ' ' or '\t' and should be defined as follows:

$$<alias\_name>=<service\_type>:<flavour>:<service\_url>|<alias\_ref>[...]$$

where <alias_name> is the name of the defined alias, <service_type> is the service type in lower case, <flavour> is the type of middleware plugin to use, <service_url> is the **URL** (p. 456) which should be used to contact the service and <alias_ref> is another defined alias. The parsed aliases will be stored internally and resolved when needed. If a alias already exist, and another alias with the same name is parsed then this other alias will overwrite the existing alias.

**Parameters:**

***conffile*** is the path to the configuration file.

***ignoreJobListFile*** is a optional boolean which indicates whether the joblistfile attribute in the configuration file should be ignored. Default is to ignored it (`true`).

**Returns:**

If loading the configuration file succeeds `true` is returned, otherwise `false` is returned.

#### 6.278.3.37 Arc::UserConfig::operator bool (void) const `[inline]`

Check for validity. The validity of an object created from this class can be checked using this casting operator. An object is valid if the constructor did not encounter any errors.

**See also:**

**operator!()** (p. 486)

#### 6.278.3.38 bool Arc::UserConfig::operator! (void) const `[inline]`

Check for non-validity. See **operator bool()** (p. 486) for a description.

**See also:**

**operator bool()** (p. 486)

#### 6.278.3.39 const std::string& Arc::UserConfig::OverlayFile () const `[inline]`

Get path to configuration overlay file.

**Returns:**

The overlay file path

**See also:**

**OverlayFile(const std::string&)** (p. 486)

#### 6.278.3.40 bool Arc::UserConfig::OverlayFile (const std::string & *path*) `[inline]`

Set path to configuration overlay file. Content of specified file is a backdoor to configuration XML generated from information stored in this class. The content of file is passed to **BaseConfig** (p. 84) class in ApplyToConfig(BaseConfig&) then merged with internal configuration XML representation. This feature is meant for quick prototyping/testing/tuning of functionality without rewriting code. It is meant for developers and most users won't need it.

The attribute associated with this setter method is 'overlayfile'.

**Parameters:**

***path*** is the new overlay file path.

**Returns:**

This method always returns `true`.

**See also:**

#### 6.278.3.41 const std::string& Arc::UserConfig::Password () const `[inline]`

Get password. Get password which is used for requesting credentials from Short Lived Credentials **Service** (p. 409).

**Returns:**

The password is returned.

**See also:**

**Password(const std::string&)** (p. 487)

#### 6.278.3.42 bool Arc::UserConfig::Password (const std::string & *newPassword*) `[inline]`

Set password. Set password which is used for requesting credentials from Short Lived Credentials **Service** (p. 409).

The attribute associated with this setter method is 'password'.

**Parameters:**

***newPassword*** is the new password to set.

**Returns:**

This method always returns true.

**See also:**

**Password() const** (p. 487)

#### 6.278.3.43 const std::string& Arc::UserConfig::ProxyPath () const `[inline]`

Get path to user proxy. Retrieve path to user proxy.

**Returns:**

Returns the path to the user proxy.

**See also:**

**ProxyPath(const std::string&)** (p. 488)

#### 6.278.3.44 bool Arc::UserConfig::ProxyPath (const std::string & *newProxyPath*) `[inline]`

Set path to user proxy. This method will set the path of the user proxy. Note that the **InitializeCredentials()** (p. 481) method will also try to set this path, by searching in different locations.

The attribute associated with this setter method is 'proxypath'

**Parameters:**

***newProxyPath*** is the path to a user proxy.

**Returns:**

This method always returns `true`.

**See also:**

**InitializeCredentials()** (p. 481)
**CredentialsFound()** (p. 479)
**ProxyPath() const** (p. 487)

#### 6.278.3.45 const URL& Arc::UserConfig::SLCS () const `[inline]`

Get the **URL** (p. 456) to the Short Lived Certificate **Service** (p. 409) (SLCS).

**Returns:**

The SLCS is returned.

**See also:**

**SLCS(const URL&)** (p. 488)

#### 6.278.3.46 bool Arc::UserConfig::SLCS (const URL & *newSLCS*) `[inline]`

Set the **URL** (p. 456) to the Short Lived Certificate **Service** (p. 409) (SLCS). The attribute associated with this setter method is 'slcs'.

**Parameters:**

***newSLCS*** is the **URL** (p. 456) to the SLCS

**Returns:**

This method always returns `true`.

**See also:**

**SLCS() const** (p. 488)

#### 6.278.3.47 const std::string& Arc::UserConfig::StoreDirectory () const `[inline]`

Get store diretory. Sets directory which is used to store credentials obtained from Short Lived **Credential** (p. 129) Servide.

**Returns:**

The path to the store directory is returned.

**See also:**

**StoreDirectory(const std::string&)** (p. 489)

#### 6.278.3.48 bool Arc::UserConfig::StoreDirectory (const std::string & *newStoreDirectory*) `[inline]`

Set store directory. Sets directory which will be used to store credentials obtained from Short Lived **Credential** (p. 129) Servide.

The attribute associated with this setter method is 'storedirectory'.

**Parameters:**

***newStoreDirectory*** is the path to the store directory.

**Returns:**

This method always returns `true`.

**See also:**

#### 6.278.3.49 int Arc::UserConfig::Timeout () const `[inline]`

Get timeout. Returns the timeout in seconds.

**Returns:**

timeout in seconds.

**See also:**

**Timeout(int)** (p. 489)
**DEFAULT_TIMEOUT** (p. 492)

#### 6.278.3.50 bool Arc::UserConfig::Timeout (int *newTimeout*)

Set timeout. When communicating with a service the timeout specifies how long, in seconds, the communicating instance should wait for a response. If the response have not been recieved before this period in time, the connection is typically dropped, and an error will be reported.

This method will set the timeout to the specified integer. If the passed integer is less than or equal to 0 then `false` is returned and the timeout will not be set, otherwise `true` is returned and the timeout will be set to the new value.

The attribute associated with this setter method is 'timeout'.

**Parameters:**

***newTimeout*** the new timeout value in seconds.

**Returns:**

`false` in case *newTimeout* <= 0, otherwise `true`.

**See also:**

**Timeout() const** (p. 489)
**DEFAULT_TIMEOUT** (p. 492)

#### 6.278.3.51 const std::string& Arc::UserConfig::UserName () const `[inline]`

Get user-name. Get username which is used for requesting credentials from Short Lived Credentials **Service** (p. 409).

**Returns:**

The username is returned.

**See also:**

**UserName(const std::string&)** (p. 490)

#### 6.278.3.52 bool Arc::UserConfig::UserName (const std::string & *name*) `[inline]`

Set user-name for SLCS. Set username which is used for requesting credentials from Short Lived Credentials **Service** (p. 409).

The attribute associated with this setter method is 'username'.

**Parameters:**

***name*** is the name of the user.

**Returns:**

This method always return true.

**See also:**

**UserName() const** (p. 490)

#### 6.278.3.53 const std::string& Arc::UserConfig::Verbosity () const `[inline]`

Get the user selected level of verbosity. The string representation of the verbosity level specified by the user is returned when calling this method. If the user have not specified the verbosity level the empty string will be referenced.

**Returns:**

the verbosity level, or empty if it has not been set.

**See also:**

**Verbosity(const std::string&)** (p. 491)

#### 6.278.3.54 bool Arc::UserConfig::Verbosity (const std::string & *newVerbosity*)

Set verbosity. The verbosity will be set when invoking this method. If the string passed cannot be parsed into a corresponding LogLevel, using the function a WARNING is reported and `false` is returned, otherwise `true` is returned.

The attribute associated with this setter method is 'verbosity'.

**Returns:**

`true` in case the verbosity could be set to a allowed LogLevel, otherwise `false`.

**See also:**

**Verbosity() const** (p. 490)

#### 6.278.3.55 const std::string& Arc::UserConfig::VOMSServerPath () const `[inline]`

Get path to file containing VOMS configuration. Get path to file which contians list of VOMS services and associated configuration parameters.

**Returns:**

The path to VOMS configuration file is returned.

**See also:**

**VOMSServerPath(const std::string&)** (p. 491)

#### 6.278.3.56 bool Arc::UserConfig::VOMSServerPath (const std::string & *path*) `[inline]`

Set path to file containing VOMS configuration. Set path to file which contians list of VOMS services and associated configuration parameters needed to contact those services. It is used by arcproxy.

The attribute associated with this setter method is 'vomsserverpath'.

**Parameters:**

***path*** the path to VOMS configuration file

**Returns:**

This method always return true.

**See also:**

**VOMSServerPath() const** (p. 491)

### 6.278.4 Field Documentation

#### 6.278.4.1 const std::string Arc::UserConfig::ARCUSERDIRECTORY `[static]`

Path to ARC user home directory. The *ARCUSERDIRECTORY* variable is the path to the ARC home directory of the current user. This path is created using the User::Home() method.

**See also:**

User::Home()

#### 6.278.4.2 const std::string Arc::UserConfig::DEFAULT_BROKER `[static]`

Default broker. The *DEFAULT_BROKER* specifies the name of the broker which should be used in case no broker is explicitly chosen.

**See also:**

**Broker** (p. 87)
**Broker(const std::string&)** (p. 475)
**Broker(const std::string&, const std::string&)** (p. 475)
**Broker() const** (p. 474)

#### 6.278.4.3 const int Arc::UserConfig::DEFAULT_TIMEOUT = 20 `[static]`

Default timeout in seconds. The *DEFAULT_TIMEOUT* specifies interval which will be used in case no timeout interval have been explicitly specified. For a description about timeout see **Timeout(int)** (p. 489).

**See also:**

**Timeout(int)** (p. 489)
**Timeout() const** (p. 489)

#### 6.278.4.4 const std::string Arc::UserConfig::DEFAULTCONFIG `[static]`

Path to default configuration file. The *DEFAULTCONFIG* variable is the path to the default configuration file used in case no configuration file have been specified. The path is created from the ARCUSERDIRECTORY object.

#### 6.278.4.5 const std::string Arc::UserConfig::EXAMPLECONFIG `[static]`

Path to example configuration. The *EXAMPLECONFIG* variable is the path to the example configuration file.

#### 6.278.4.6 const std::string Arc::UserConfig::SYSCONFIG `[static]`

Path to system configuration. The *SYSCONFIG* variable is the path to the system configuration file.

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

- UserConfig.h

## 6.279 Arc::UsernameToken Class Reference

Interface for manipulation of WS-Security according to Username Token **Profile** (p. 357).

`#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)

### 6.279.1 Detailed Description

Interface for manipulation of WS-Security according to Username Token **Profile** (p. 357).

### 6.279.2 Member Enumeration Documentation

#### 6.279.2.1 enum Arc::UsernameToken::PasswordType

SOAP header element

### 6.279.3 Constructor & Destructor Documentation

#### 6.279.3.1 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.

#### 6.279.3.2 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. 493) wsu:ID="...">

***pwdtype*** <wsse:Password Type="...">...</wsse:Password>

#### 6.279.3.3 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. 290) Authentication Code

***iteration*** <wsse11:Iteration>...</wsse11:Iteration>

### 6.279.4 Member Function Documentation

#### 6.279.4.1 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

#### 6.279.4.2 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. 493) class using obtained derived_key.

#### 6.279.4.3 Arc::UsernameToken::operator bool (void)

Returns true of constructor succeeded

#### 6.279.4.4 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

## 6.280 Arc::UserSwitch Class Reference

```
#include <User.h>
```

### 6.280.1 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. 388) class while spawning new process.

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

- User.h

## 6.281 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)

### 6.281.1 Detailed Description

Stores definitions for making decision if VOMS server is trusted

### 6.281.2 Constructor & Destructor Documentation

#### 6.281.2.1 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) .

#### 6.281.2.2 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. 497) and **AddRegex()** (p. 497) for more information.

### 6.281.3 Member Function Documentation

#### 6.281.3.1 VOMSTrustChain& Arc::VOMSTrustList::AddChain (void)

Adds empty chain of trusted DNs to list.

#### 6.281.3.2 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.

#### 6.281.3.3 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

## 6.282 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)

### 6.282.1 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.

### 6.282.2 Constructor & Destructor Documentation

#### 6.282.2.1 Arc::WSAEndpointReference::WSAEndpointReference (const XMLNode & *epr*)

Link to top level EPR XML node Linking to existing EPR in XML tree

#### 6.282.2.2 Arc::WSAEndpointReference::WSAEndpointReference (const WSAEndpointReference & *wsa*)

Copy constructor

#### 6.282.2.3 Arc::WSAEndpointReference::WSAEndpointReference (const std::string & *address*)

Creating independent EPR - not implemented

#### 6.282.2.4 Arc::WSAEndpointReference::WSAEndpointReference (void)

Dummy constructor - creates invalid instance

#### 6.282.2.5 Arc::WSAEndpointReference::∼WSAEndpointReference (void)

Destructor. All empty elements of EPR XML are destroyed here too

## 6.282.3 Member Function Documentation

### 6.282.3.1 void Arc::WSAEndpointReference::Address (const std::string & *uri*)

Assigns new Address value. If EPR had no Address element it is created.

### 6.282.3.2 std::string Arc::WSAEndpointReference::Address (void) const

Returns Address (**URL** (p. 456)) encoded in EPR

### 6.282.3.3 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.

### 6.282.3.4 Arc::WSAEndpointReference::operator XMLNode (void)

Returns reference to EPR top XML node

### 6.282.3.5 WSAEndpointReference& Arc::WSAEndpointReference::operator= (const std::string & *address*)

Same as Address(uri)

### 6.282.3.6 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

## 6.283 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_**

### 6.283.1 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.

### 6.283.2 Constructor & Destructor Documentation

#### 6.283.2.1 Arc::WSAHeader::WSAHeader (SOAPEnvelope & *soap*)

Linking to a header of existing SOAP message

#### 6.283.2.2 Arc::WSAHeader::WSAHeader (const std::string & *action*)

Creating independent SOAP header - not implemented

### 6.283.3 Member Function Documentation

#### 6.283.3.1 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.

#### 6.283.3.2 std::string Arc::WSAHeader::Action (void) const

Returns content of Action element of SOAP Header.

#### 6.283.3.3 static bool Arc::WSAHeader::Check (SOAPEnvelope & *soap*) `[static]`

Tells if specified SOAP message has WSA header

#### 6.283.3.4 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.

#### 6.283.3.5 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.

#### 6.283.3.6 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.

#### 6.283.3.7 std::string Arc::WSAHeader::MessageID (void) const

Returns content of MessageID element of SOAP Header.

#### 6.283.3.8 XMLNode Arc::WSAHeader::NewReferenceParameter (const std::string & *name*)

Creates new ReferenceParameter element with specified name. Returns reference to created element.

#### 6.283.3.9 Arc::WSAHeader::operator XMLNode (void)

Returns reference to SOAP Header - not implemented

#### 6.283.3.10 XMLNode Arc::WSAHeader::ReferenceParameter (const std::string & *name*)

Returns first ReferenceParameter element with specified name

#### 6.283.3.11 XMLNode Arc::WSAHeader::ReferenceParameter (int *n*)

Return n-th ReferenceParameter element

#### 6.283.3.12 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.

#### 6.283.3.13 std::string Arc::WSAHeader::RelatesTo (void) const

Returns content of RelatesTo element of SOAP Header.

#### 6.283.3.14 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.

#### 6.283.3.15 std::string Arc::WSAHeader::RelationshipType (void) const

Returns content of RelationshipType element of SOAP Header.

#### 6.283.3.16 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.

#### 6.283.3.17 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.

#### 6.283.3.18 std::string Arc::WSAHeader::To (void) const

Returns content of To element of SOAP Header.

### 6.283.4 Field Documentation

#### 6.283.4.1 bool Arc::WSAHeader::header_allocated_ `[protected]`

SOAP header element

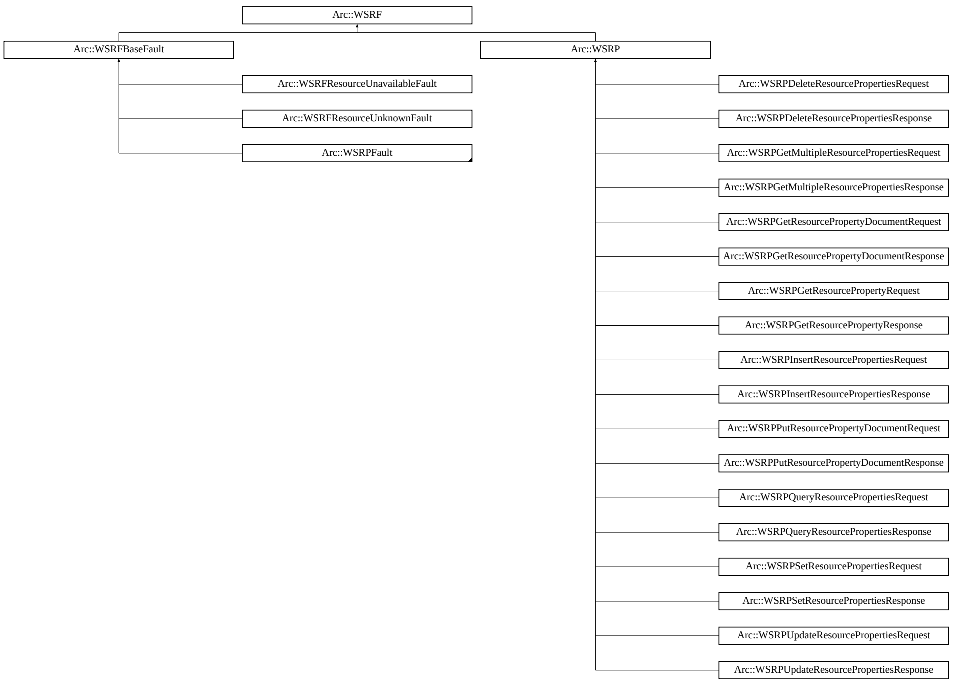
The documentation for this class was generated from the following file:

- WSA.h

## 6.284 Arc::WSRF Class Reference

Base class for every **WSRF** (p. 503) message.

`#include <WSRF.h>`Inheritance diagram for Arc::WSRF::


Arc::WSRF
Arc::WSRFBaseFault
Arc::WSRP
Arc::WSRFResourceUnavailableFault
Arc::WSRFResourceUnknownFault
Arc::WSRPFault
Arc::WSRPDeleteResourcePropertiesRequest
Arc::WSRPDeleteResourcePropertiesResponse
Arc::WSRPGetMultipleResourcePropertiesRequest
Arc::WSRPGetMultipleResourcePropertiesResponse
Arc::WSRPGetResourcePropertyDocumentRequest
Arc::WSRPGetResourcePropertyDocumentResponse
Arc::WSRPGetResourcePropertyRequest
Arc::WSRPGetResourcePropertyResponse
Arc::WSRPInsertResourcePropertiesRequest
Arc::WSRPInsertResourcePropertiesResponse
Arc::WSRPPutResourcePropertyDocumentRequest
Arc::WSRPPutResourcePropertyDocumentResponse
Arc::WSRPQueryResourcePropertiesRequest
Arc::WSRPQueryResourcePropertiesResponse
Arc::WSRPSetResourcePropertiesRequest
Arc::WSRPSetResourcePropertiesResponse
Arc::WSRPUpdateResourcePropertiesRequest
Arc::WSRPUpdateResourcePropertiesResponse

## 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_**

### 6.284.1 Detailed Description

Base class for every **WSRF** (p. 503) message. This class is not intended to be used directly. Use it like reference while passing through unknown **WSRF** (p. 503) message or use classes derived from it.

### 6.284.2 Constructor & Destructor Documentation

#### 6.284.2.1 Arc::WSRF::WSRF (SOAPEnvelope & *soap*, const std::string & *action* = `""`)

Constructor - creates object out of supplied SOAP tree.

#### 6.284.2.2 Arc::WSRF::WSRF (bool *fault* = `false`, const std::string & *action* = `""`)

Constructor - creates new **WSRF** (p. 503) object

### 6.284.3 Member Function Documentation

#### 6.284.3.1 virtual Arc::WSRF::operator bool (void) `[inline, virtual]`

Returns true if instance is valid

References valid_.

#### 6.284.3.2 void Arc::WSRF::set_namespaces (void) `[protected]`

true if object represents valid **WSRF** (p. 503) message set WS Resource namespaces and default prefixes in SOAP message

Reimplemented in **Arc::WSRP** (p. 509), and **Arc::WSRFBaseFault** (p. 505).

#### 6.284.3.3 virtual SOAPEnvelope& Arc::WSRF::SOAP (void) `[inline, virtual]`

Direct access to underlying SOAP element

### 6.284.4 Field Documentation

#### 6.284.4.1 bool Arc::WSRF::allocated_ `[protected]`

Associated SOAP message - it's SOAP message after all

#### 6.284.4.2 bool Arc::WSRF::valid_ `[protected]`

true if soap_ needs to be deleted in destructor

Referenced by operator bool().

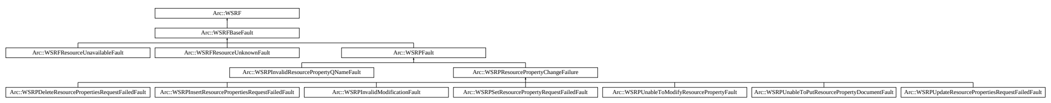
The documentation for this class was generated from the following file:

- WSRF.h

## 6.285 Arc::WSRFBaseFault Class Reference

Base class for **WSRF** (p. 503) fault messages.

`#include <WSRFBaseFault.h>`Inheritance diagram for Arc::WSRFBaseFault::


Arc::WSRF
Arc::WSRFBaseFault
Arc::WSRFResourceUnavailableFault
Arc::WSRFResourceUnknownFault
Arc::WSRPFault
Arc::WSRPInvalidResourcePropertyQNameFault
Arc::WSRPResourcePropertyChangeFailure
Arc::WSRPDeleteResourcePropertiesRequestFailedFault
Arc::WSRPInsertResourcePropertiesRequestFailedFault
Arc::WSRPInvalidModificationFault
Arc::WSRPSetResourcePropertyRequestFailedFault
Arc::WSRPUnableToModifyResourcePropertyFault
Arc::WSRPUnableToPutResourcePropertyDocumentFault
Arc::WSRPUpdateResourcePropertiesRequestFailedFault

### Public Member Functions

- **WSRFBaseFault** (SOAPEnvelope &soap)
- **WSRFBaseFault** (const std::string &type)

### Protected Member Functions

- void **set_namespaces** (void)

### 6.285.1 Detailed Description

Base class for **WSRF** (p. 503) fault messages. Use classes inherited from it for specific faults.

### 6.285.2 Constructor & Destructor Documentation

#### 6.285.2.1 Arc::WSRFBaseFault::WSRFBaseFault (SOAPEnvelope & *soap*)

Constructor - creates object out of supplied SOAP tree.

#### 6.285.2.2 Arc::WSRFBaseFault::WSRFBaseFault (const std::string & *type*)

Constructor - creates new **WSRF** (p. 503) fault

### 6.285.3 Member Function Documentation

#### 6.285.3.1 void Arc::WSRFBaseFault::set_namespaces (void) `[protected]`

set WS-ResourceProperties namespaces and default prefixes in SOAP message

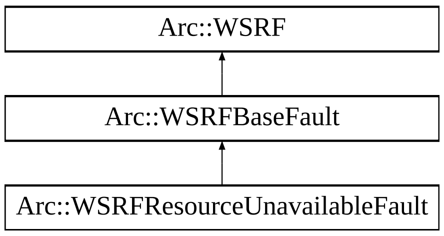
Reimplemented from **Arc::WSRF** (p. 504).

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

- WSRFBaseFault.h

## 6.286 Arc::WSRFResourceUnavailableFault Class Reference

Inheritance diagram for Arc::WSRFResourceUnavailableFault::

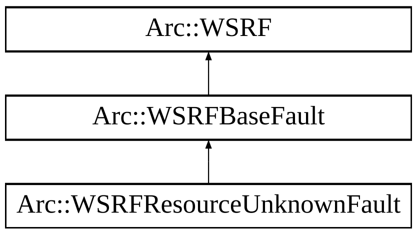


The documentation for this class was generated from the following file:

- WSRFBaseFault.h

## 6.287 Arc::WSRFResourceUnknownFault Class Reference

Inheritance diagram for Arc::WSRFResourceUnknownFault::



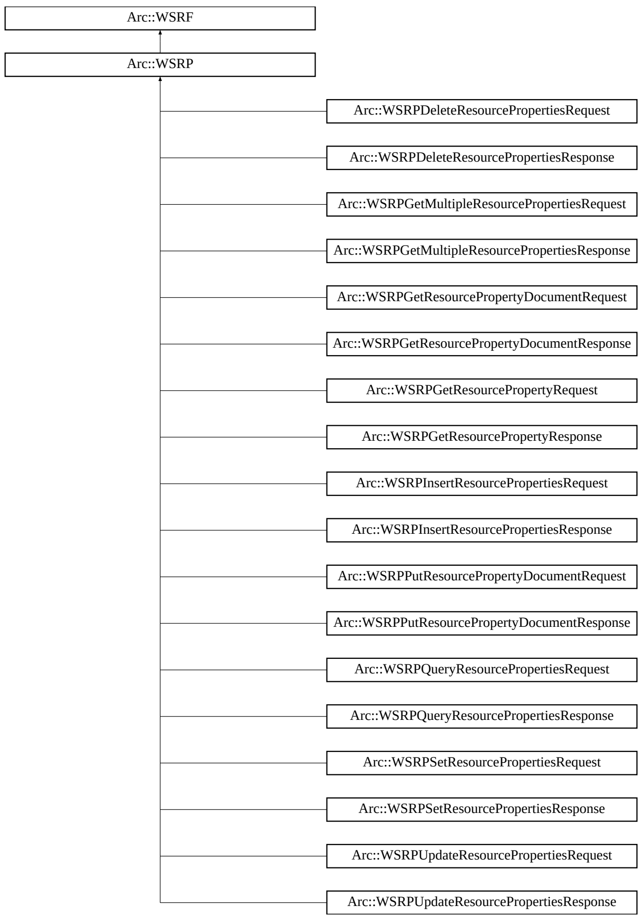
The documentation for this class was generated from the following file:

- WSRFBaseFault.h

## 6.288 Arc::WSRP Class Reference

Base class for WS-ResourceProperties structures.

`#include <WSResourceProperties.h>`Inheritance diagram for Arc::WSRP::


Arc::WSRF
Arc::WSRP
Arc::WSRPDeleteResourcePropertiesRequest
Arc::WSRPDeleteResourcePropertiesResponse
Arc::WSRPGetMultipleResourcePropertiesRequest
Arc::WSRPGetMultipleResourcePropertiesResponse
Arc::WSRPGetResourcePropertyDocumentRequest
Arc::WSRPGetResourcePropertyDocumentResponse
Arc::WSRPGetResourcePropertyRequest
Arc::WSRPGetResourcePropertyResponse
Arc::WSRPInsertResourcePropertiesRequest
Arc::WSRPInsertResourcePropertiesResponse
Arc::WSRPPutResourcePropertyDocumentRequest
Arc::WSRPPutResourcePropertyDocumentResponse
Arc::WSRPQueryResourcePropertiesRequest
Arc::WSRPQueryResourcePropertiesResponse
Arc::WSRPSetResourcePropertiesRequest
Arc::WSRPSetResourcePropertiesResponse
Arc::WSRPUpdateResourcePropertiesRequest
Arc::WSRPUpdateResourcePropertiesResponse

### 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)

### 6.288.1 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.

### 6.288.2 Constructor & Destructor Documentation

#### 6.288.2.1 Arc::WSRP::WSRP (bool *fault* = `false`, const std::string & *action* = `""`)

Constructor - prepares object for creation of new **WSRP** (p. 508) request/response/fault

#### 6.288.2.2 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.

### 6.288.3 Member Function Documentation

#### 6.288.3.1 void Arc::WSRP::set_namespaces (void) `[protected]`

set WS-ResourceProperties namespaces and default prefixes in SOAP message

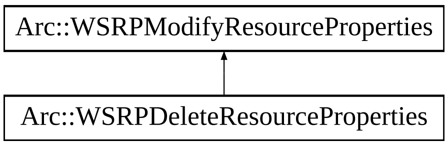
Reimplemented from **Arc::WSRF** (p. 504).

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

- WSResourceProperties.h

## 6.289 Arc::WSRPDeleteResourceProperties Class Reference

Inheritance diagram for Arc::WSRPDeleteResourceProperties::

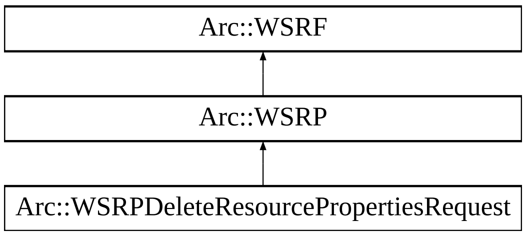

Arc::WSRPModifyResourceProperties
Arc::WSRPDeleteResourceProperties

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

- WSResourceProperties.h

## 6.290 Arc::WSRPDeleteResourcePropertiesRequest Class Reference

Inheritance diagram for Arc::WSRPDeleteResourcePropertiesRequest::

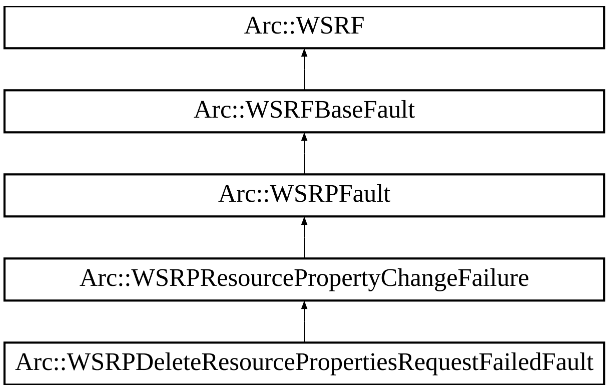


The documentation for this class was generated from the following file:

- WSResourceProperties.h

## 6.291 Arc::WSRPDeleteResourcePropertiesRequestFailedFault Class Reference

Inheritance diagram for Arc::WSRPDeleteResourcePropertiesRequestFailedFault::

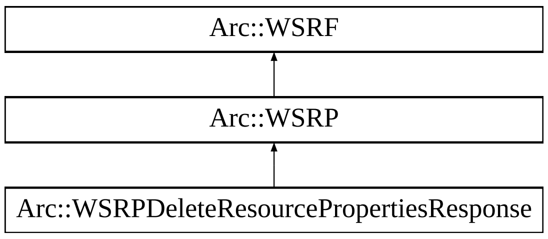

Arc::WSRF
Arc::WSRFBaseFault
Arc::WSRPFault
Arc::WSRPResourcePropertyChangeFailure
Arc::WSRPDeleteResourcePropertiesRequestFailedFault

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

- WSResourceProperties.h

## 6.292 Arc::WSRPDeleteResourcePropertiesResponse Class Reference

Inheritance diagram for Arc::WSRPDeleteResourcePropertiesResponse::



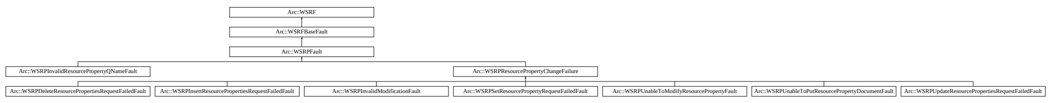
The documentation for this class was generated from the following file:

- WSResourceProperties.h

## 6.293 Arc::WSRPFault Class Reference

Base class for WS-ResourceProperties faults.

`#include <WSResourceProperties.h>`Inheritance diagram for Arc::WSRPFault::


Arc::WSRF
Arc::WSRFBaseFault
Arc::WSRPFault
Arc::WSRPInvalidResourcePropertyQNameFault
Arc::WSRPResourcePropertyChangeFailure
Arc::WSRPDeleteResourcePropertiesRequestFailedFault
Arc::WSRPInsertResourcePropertiesRequestFailedFault
Arc::WSRPInvalidModificationFault
Arc::WSRPSetResourcePropertyRequestFailedFault
Arc::WSRPUnableToModifyResourcePropertyFault
Arc::WSRPUnableToPutResourcePropertyDocumentFault
Arc::WSRPUpdateResourcePropertiesRequestFailedFault

### Public Member Functions

- **WSRPFault** (SOAPEnvelope &soap)
- **WSRPFault** (const std::string &type)

### 6.293.1 Detailed Description

Base class for WS-ResourceProperties faults.

### 6.293.2 Constructor & Destructor Documentation

#### 6.293.2.1 Arc::WSRPFault::WSRPFault (SOAPEnvelope & *soap*)

Constructor - creates object out of supplied SOAP tree.

#### 6.293.2.2 Arc::WSRPFault::WSRPFault (const std::string & *type*)

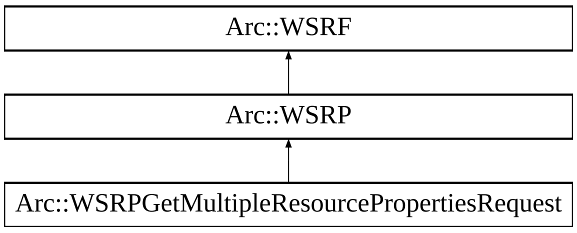
Constructor - creates new **WSRP** (p. 508) fault

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

- WSResourceProperties.h

## 6.294 Arc::WSRPGetMultipleResourcePropertiesRequest Class Reference

Inheritance diagram for Arc::WSRPGetMultipleResourcePropertiesRequest::

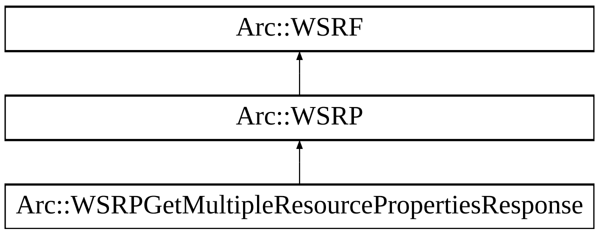


The documentation for this class was generated from the following file:

- WSResourceProperties.h

## 6.295 Arc::WSRPGetMultipleResourcePropertiesResponse Class Reference

Inheritance diagram for Arc::WSRPGetMultipleResourcePropertiesResponse::

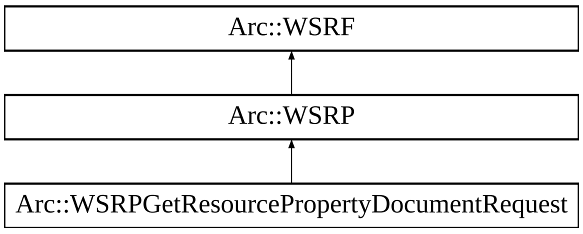


The documentation for this class was generated from the following file:

- WSResourceProperties.h

## 6.296 Arc::WSRPGetResourcePropertyDocumentRequest Class Reference

Inheritance diagram for Arc::WSRPGetResourcePropertyDocumentRequest::

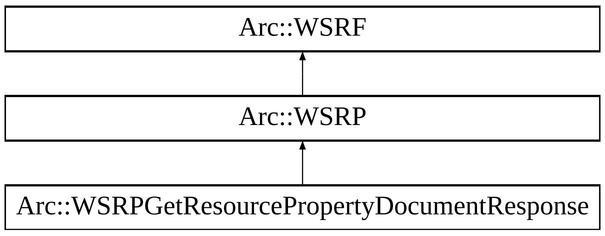


The documentation for this class was generated from the following file:

- WSResourceProperties.h

## 6.297 Arc::WSRPGetResourcePropertyDocumentResponse Class Reference

Inheritance diagram for Arc::WSRPGetResourcePropertyDocumentResponse::

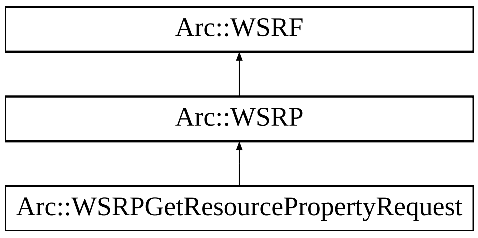


The documentation for this class was generated from the following file:

- WSResourceProperties.h

## 6.298 Arc::WSRPGetResourcePropertyRequest Class Reference

Inheritance diagram for Arc::WSRPGetResourcePropertyRequest::

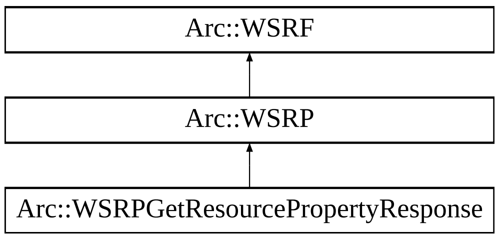


The documentation for this class was generated from the following file:

- WSResourceProperties.h

## 6.299 Arc::WSRPGetResourcePropertyResponse Class Reference

Inheritance diagram for Arc::WSRPGetResourcePropertyResponse::

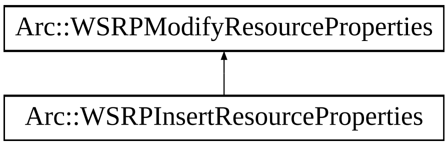


The documentation for this class was generated from the following file:

- WSResourceProperties.h

## 6.300 Arc::WSRPInsertResourceProperties Class Reference

Inheritance diagram for Arc::WSRPInsertResourceProperties::

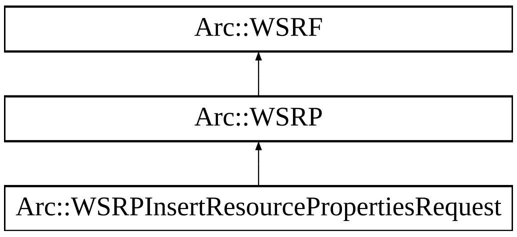

Arc::WSRPModifyResourceProperties
Arc::WSRPInsertResourceProperties

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

- WSResourceProperties.h

## 6.301 Arc::WSRPInsertResourcePropertiesRequest Class Reference

Inheritance diagram for Arc::WSRPInsertResourcePropertiesRequest::

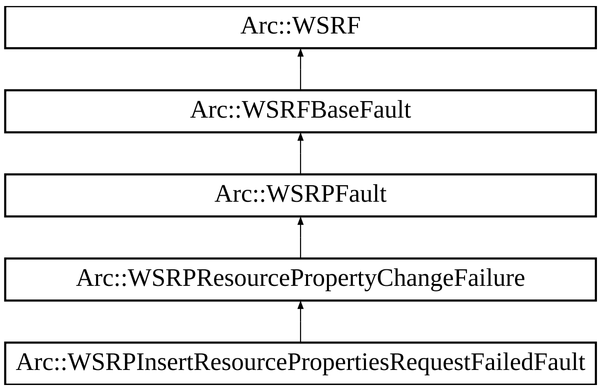


The documentation for this class was generated from the following file:

- WSResourceProperties.h

## 6.302 Arc::WSRPInsertResourcePropertiesRequestFailedFault Class Reference

Inheritance diagram for Arc::WSRPInsertResourcePropertiesRequestFailedFault::

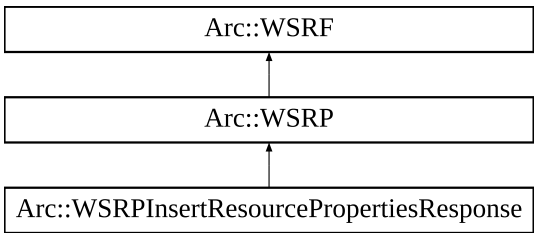

Arc::WSRF
Arc::WSRFBaseFault
Arc::WSRPFault
Arc::WSRPResourcePropertyChangeFailure
Arc::WSRPInsertResourcePropertiesRequestFailedFault

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

- WSResourceProperties.h

## 6.303 Arc::WSRPInsertResourcePropertiesResponse Class Reference

Inheritance diagram for Arc::WSRPInsertResourcePropertiesResponse::

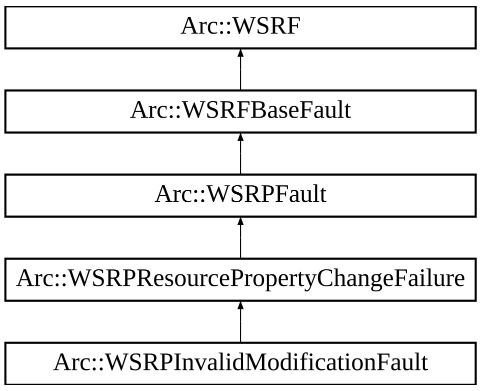


The documentation for this class was generated from the following file:

- WSResourceProperties.h

## 6.304 Arc::WSRPInvalidModificationFault Class Reference

Inheritance diagram for Arc::WSRPInvalidModificationFault::

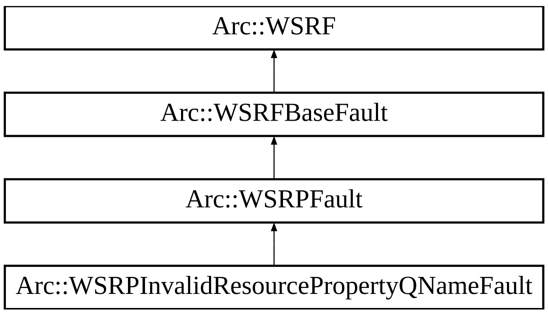

Arc::WSRF
Arc::WSRFBaseFault
Arc::WSRPFault
Arc::WSRPResourcePropertyChangeFailure
Arc::WSRPInvalidModificationFault

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

- WSResourceProperties.h

## 6.305 Arc::WSRPInvalidResourcePropertyQNameFault Class Reference

Inheritance diagram for Arc::WSRPInvalidResourcePropertyQNameFault::

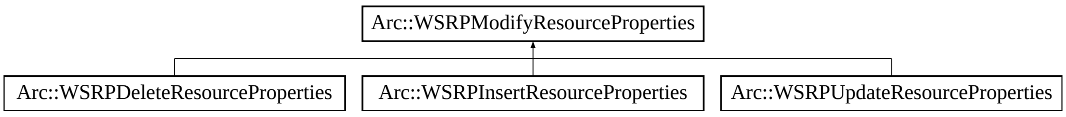


The documentation for this class was generated from the following file:

- WSResourceProperties.h

## 6.306 Arc::WSRPModifyResourceProperties Class Reference

Inheritance diagram for Arc::WSRPModifyResourceProperties::

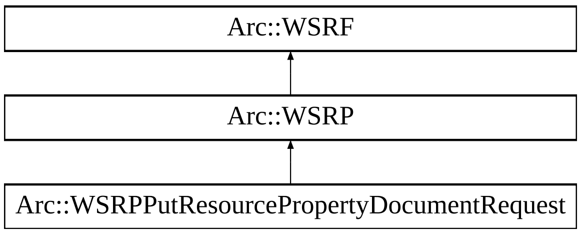

Arc::WSRPModifyResourceProperties
Arc::WSRPDeleteResourceProperties
Arc::WSRPInsertResourceProperties
Arc::WSRPUpdateResourceProperties

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

- WSResourceProperties.h

## 6.307 Arc::WSRPPutResourcePropertyDocumentRequest Class Reference

Inheritance diagram for Arc::WSRPPutResourcePropertyDocumentRequest::

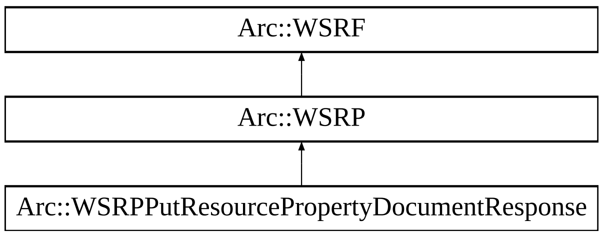


The documentation for this class was generated from the following file:

- WSResourceProperties.h

## 6.308 Arc::WSRPPutResourcePropertyDocumentResponse Class Reference

Inheritance diagram for Arc::WSRPPutResourcePropertyDocumentResponse::

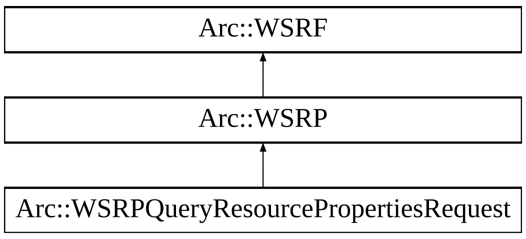


The documentation for this class was generated from the following file:

- WSResourceProperties.h

## 6.309 Arc::WSRPQueryResourcePropertiesRequest Class Reference

Inheritance diagram for Arc::WSRPQueryResourcePropertiesRequest::

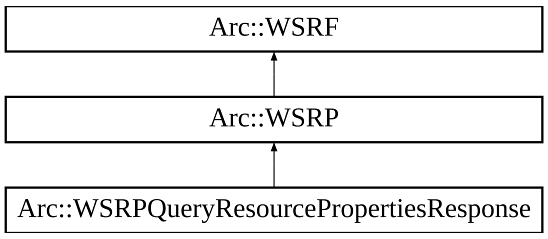


The documentation for this class was generated from the following file:

- WSResourceProperties.h

## 6.310 Arc::WSRPQueryResourcePropertiesResponse Class Reference

Inheritance diagram for Arc::WSRPQueryResourcePropertiesResponse::

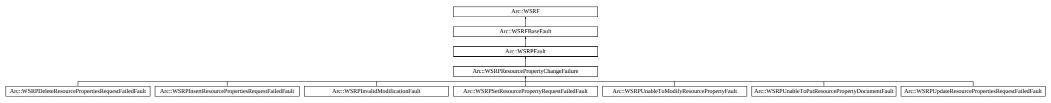


The documentation for this class was generated from the following file:

- WSResourceProperties.h

## 6.311 Arc::WSRPResourcePropertyChangeFailure Class Reference

`#include <WSResourceProperties.h>`Inheritance diagram for Arc::WSRPResourcePropertyChangeFailure::


Arc::WSRF
Arc::WSRFBaseFault
Arc::WSRPFault
Arc::WSRPResourcePropertyChangeFailure
Arc::WSRPDeleteResourcePropertiesRequestFailedFault
Arc::WSRPInsertResourcePropertiesRequestFailedFault
Arc::WSRPInvalidModificationFault
Arc::WSRPSetResourcePropertyRequestFailedFault
Arc::WSRPUnableToModifyResourcePropertyFault
Arc::WSRPUnableToPutResourcePropertyDocumentFault
Arc::WSRPUpdateResourcePropertiesRequestFailedFault

### Public Member Functions

- **WSRPResourcePropertyChangeFailure** (SOAPEnvelope &soap)
- **WSRPResourcePropertyChangeFailure** (const std::string &type)

### 6.311.1 Detailed Description

Base class for WS-ResourceProperties faults which contain ResourcePropertyChangeFailure

### 6.311.2 Constructor & Destructor Documentation

#### 6.311.2.1 Arc::WSRPResourcePropertyChangeFailure::WSRPResourcePropertyChangeFailure (SOAPEnvelope & *soap*) `[inline]`

Constructor - creates object out of supplied SOAP tree.

#### 6.311.2.2 Arc::WSRPResourcePropertyChangeFailure::WSRPResourcePropertyChangeFailure (const std::string & *type*) `[inline]`

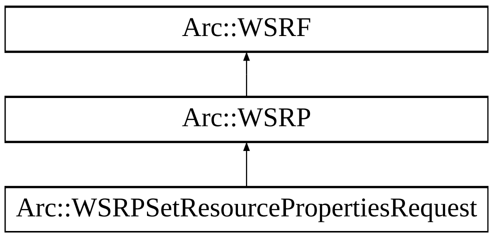
Constructor - creates new **WSRP** (p. 508) fault

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

- WSResourceProperties.h

## 6.312 Arc::WSRPSetResourcePropertiesRequest Class Reference

Inheritance diagram for Arc::WSRPSetResourcePropertiesRequest::

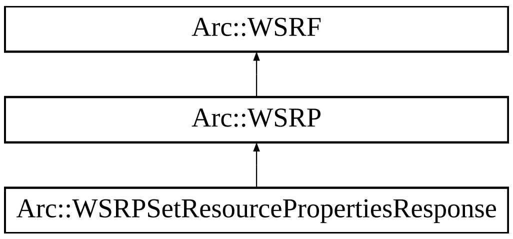


The documentation for this class was generated from the following file:

- WSResourceProperties.h

## 6.313 Arc::WSRPSetResourcePropertiesResponse Class Reference

Inheritance diagram for Arc::WSRPSetResourcePropertiesResponse::

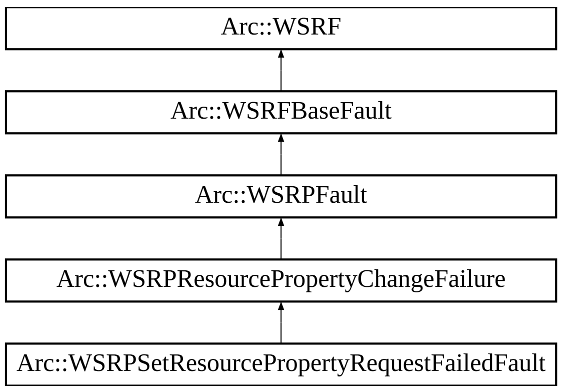


The documentation for this class was generated from the following file:

- WSResourceProperties.h

## 6.314 Arc::WSRPSetResourcePropertyRequestFailedFault Class Reference

Inheritance diagram for Arc::WSRPSetResourcePropertyRequestFailedFault::

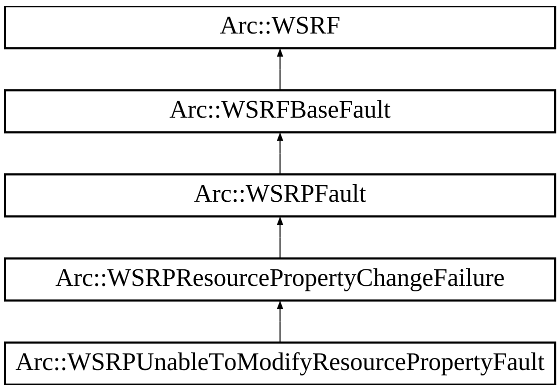

Arc::WSRF
Arc::WSRFBaseFault
Arc::WSRPFault
Arc::WSRPResourcePropertyChangeFailure
Arc::WSRPSetResourcePropertyRequestFailedFault

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

- WSResourceProperties.h

## 6.315 Arc::WSRPUnableToModifyResourcePropertyFault Class Reference

Inheritance diagram for Arc::WSRPUnableToModifyResourcePropertyFault::

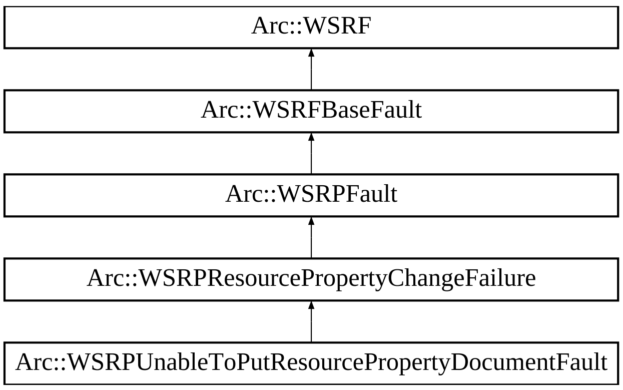

Arc::WSRF
Arc::WSRFBaseFault
Arc::WSRPFault
Arc::WSRPResourcePropertyChangeFailure
Arc::WSRPUnableToModifyResourcePropertyFault

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

- WSResourceProperties.h

## 6.316 Arc::WSRPUnableToPutResourcePropertyDocumentFault Class Reference

Inheritance diagram for Arc::WSRPUnableToPutResourcePropertyDocumentFault::

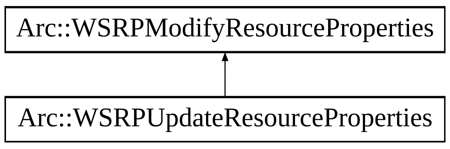

Arc::WSRF
Arc::WSRFBaseFault
Arc::WSRPFault
Arc::WSRPResourcePropertyChangeFailure
Arc::WSRPUnableToPutResourcePropertyDocumentFault

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

- WSResourceProperties.h

## 6.317 Arc::WSRPUpdateResourceProperties Class Reference

Inheritance diagram for Arc::WSRPUpdateResourceProperties::

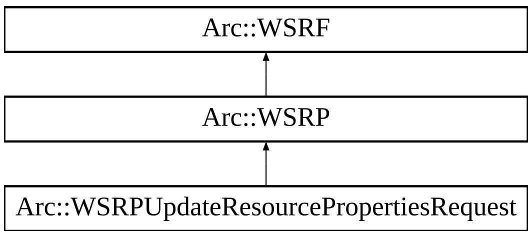

Arc::WSRPModifyResourceProperties
Arc::WSRPUpdateResourceProperties

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

- WSResourceProperties.h

## 6.318 Arc::WSRPUpdateResourcePropertiesRequest Class Reference

Inheritance diagram for Arc::WSRPUpdateResourcePropertiesRequest::

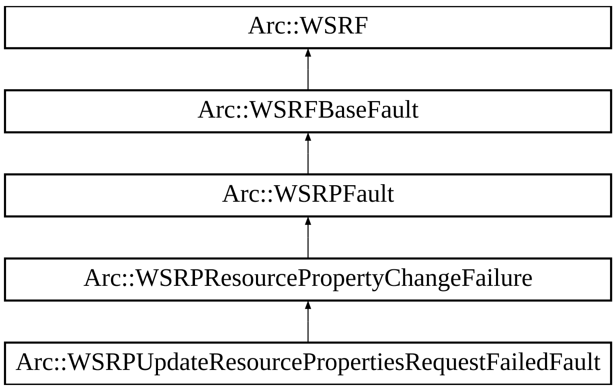


The documentation for this class was generated from the following file:

- WSResourceProperties.h

## 6.319 Arc::WSRPUpdateResourcePropertiesRequestFailedFault Class Reference

Inheritance diagram for Arc::WSRPUpdateResourcePropertiesRequestFailedFault::

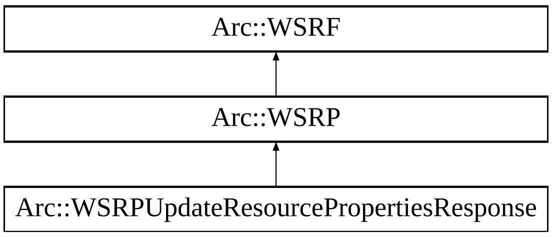

Arc::WSRF
Arc::WSRFBaseFault
Arc::WSRPFault
Arc::WSRPResourcePropertyChangeFailure
Arc::WSRPUpdateResourcePropertiesRequestFailedFault

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

- WSResourceProperties.h

## 6.320 Arc::WSRPUpdateResourcePropertiesResponse Class Reference

Inheritance diagram for Arc::WSRPUpdateResourcePropertiesResponse::

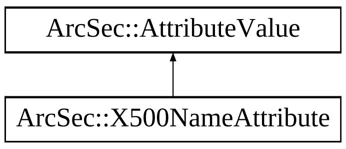


The documentation for this class was generated from the following file:

- WSResourceProperties.h

## 6.321 ArcSec::X500NameAttribute Class Reference

Inheritance diagram for ArcSec::X500NameAttribute::



### Public Member Functions

- virtual std::string **encode** ()
- virtual std::string **getType** ()
- virtual std::string **getId** ()

### 6.321.1 Member Function Documentation

#### 6.321.1.1 virtual std::string ArcSec::X500NameAttribute::encode () `[inline, virtual]`

encode the value in a string format

Implements **ArcSec::AttributeValue** (p. 78).

#### 6.321.1.2 virtual std::string ArcSec::X500NameAttribute::getId () `[inline, virtual]`

Get the AttributeId of the <Attribute>

Implements **ArcSec::AttributeValue** (p. 78).

#### 6.321.1.3 virtual std::string ArcSec::X500NameAttribute::getType () `[inline, virtual]`

Get the DataType of the <Attribute>

Implements **ArcSec::AttributeValue** (p. 78).

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

- X500NameAttribute.h

## 6.322 Arc::X509Token Class Reference

Class for manipulating X.509 Token **Profile** (p. 357).

`#include <X509Token.h>`

### Public Types

- enum **X509TokenType**

### Public Member Functions

- **X509Token** (SOAPEnvelope &soap, const std::string &keyfile="")
- **X509Token** (SOAPEnvelope &soap, const std::string &certfile, const std::string &keyfile, **X509TokenType** token_type=Signature)
- ~**X509Token** (void)
- **operator bool** (void)
- bool **Authenticate** (const std::string &cafile, const std::string &capath)
- bool **Authenticate** (void)

### 6.322.1 Detailed Description

Class for manipulating X.509 Token **Profile** (p. 357). 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.

### 6.322.2 Member Enumeration Documentation

#### 6.322.2.1 enum Arc::X509Token::X509TokenType

X509TokeType is for distinguishing two types of operation. It is used as the parameter of constuctor.

### 6.322.3 Constructor & Destructor Documentation

#### 6.322.3.1 Arc::X509Token::X509Token (SOAPEnvelope & *soap*, const std::string & *keyfile* = `""`)

Constructor.Parse X509 Token information from SOAP header. X509 Token related information is extracted from SOAP header and stored in class variables. And then it the **X509Token** (p. 543) object will be used for authentication if the tokentype is Signature; otherwise if the tokentype is Encryption, the encrypted soap body will be decrypted and replaced by decrypted message. keyfile is only needed when the **X509Token** (p. 543) is encryption token

#### 6.322.3.2 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.

#### 6.322.3.3 Arc::X509Token::∼X509Token (void)

Deconstructor. Nothing to be done except finalizing the xmlsec library.

### 6.322.4 Member Function Documentation

#### 6.322.4.1 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.

#### 6.322.4.2 bool Arc::X509Token::Authenticate (const std::string & *cafile*, const std::string & *capath*)

Check signature by using the certificare information in **X509Token** (p. 543) 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. 543)) 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

#### 6.322.4.3 Arc::X509Token::operator bool (void)

Returns true of constructor succeeded

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

- X509Token.h

## 6.323 Arc::XmlContainer Class Reference

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

- XmlContainer.h

## 6.324 Arc::XmlDatabase Class Reference

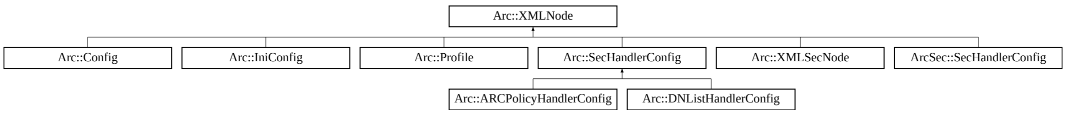
The documentation for this class was generated from the following file:

- XmlDatabase.h

## 6.325 Arc::XMLNode Class Reference

Wrapper for LibXML library Tree interface.

`#include <XMLNode.h>`Inheritance diagram for Arc::XMLNode::



### Public Member Functions

- **XMLNode** (void)
- **XMLNode** (const **XMLNode** &node)
- **XMLNode** (const std::string &xml)
- **XMLNode** (const char ∗xml, int len=-1)
- **XMLNode** (long ptr_addr)
- **XMLNode** (const **NS** &ns, const char ∗name)
- ∼**XMLNode** (void)
- void **New** (**XMLNode** &node) const
- void **Exchange** (**XMLNode** &node)
- void **Move** (**XMLNode** &node)
- void **Swap** (**XMLNode** &node)
- **operator bool** (void) const
- bool **operator!** (void) const
- bool **operator==** (const **XMLNode** &node)
- bool **operator!=** (const **XMLNode** &node)
- bool **Same** (const **XMLNode** &node)
- bool **operator==** (bool val)
- bool **operator!=** (bool val)
- bool **operator==** (const std::string &str)
- bool **operator!=** (const std::string &str)
- bool **operator==** (const char ∗str)
- bool **operator!=** (const char ∗str)
- **XMLNode Child** (int n=0) 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, bool keep=false, int recursion=-1)
- **NS Namespaces** (void)
- std::string **NamespacePrefix** (const char ∗urn)
- **XMLNode NewChild** (const char ∗name, int n=-1, bool global_order=false)
- **XMLNode NewChild** (const std::string &name, int n=-1, bool global_order=false)
- **XMLNode NewChild** (const char ∗name, const **NS** &namespaces, int n=-1, bool global_-order=false)
- **XMLNode NewChild** (const std::string &name, const **NS** &namespaces, int n=-1, bool global_-order=false)
- **XMLNode NewChild** (const **XMLNode** &node, int n=-1, bool global_order=false)
- void **Replace** (const **XMLNode** &node)
- void **Destroy** (void)
- XMLNodeList **Path** (const std::string &path) 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)
- bool **Validate** (const std::string &schema_file, std::string &err_msg)

## Protected Member Functions

- **XMLNode** (xmlNodePtr node)

## Protected Attributes

- bool **is_owner_**
- bool **is_temporary_**

### 6.325.1 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.

### 6.325.2 Constructor & Destructor Documentation

#### 6.325.2.1 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.

#### 6.325.2.2 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.

#### 6.325.2.3 Arc::XMLNode::XMLNode (const XMLNode & *node*) `[inline]`

Copies existing instance. Underlying XML element is NOT copied. Ownership is NOT inherited.

#### 6.325.2.4 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

#### 6.325.2.5 Arc::XMLNode::XMLNode (const char ∗ *xml*, int *len* = `-1`)

Same as previous

#### 6.325.2.6 Arc::XMLNode::XMLNode (long *ptr_addr*)

Copy constructor. Used by language bindigs

#### 6.325.2.7 Arc::XMLNode::XMLNode (const NS & *ns*, const char ∗ *name*)

Creates empty XML document structure with specified namespaces. Created XML contains only root element named 'name'. Created structure is pointed and owned by constructed instance

#### 6.325.2.8 Arc::XMLNode::∼XMLNode (void)

Destructor Also destroys underlying XML document if owned by this instance

### 6.325.3 Member Function Documentation

#### 6.325.3.1 XMLNode Arc::XMLNode::Attribute (const std::string & *name*) const `[inline]`

Returns **XMLNode** (p. 547) instance representing first attribute of node with specified by name

References Attribute().

#### 6.325.3.2 XMLNode Arc::XMLNode::Attribute (const char ∗ *name*) const

Returns **XMLNode** (p. 547) instance representing first attribute of node with specified by name

#### 6.325.3.3 XMLNode Arc::XMLNode::Attribute (int *n* = `0`) const

Returns list of all attributes of node. Returns **XMLNode** (p. 547) instance reresenting n-th attribute of node.

Referenced by Attribute().

#### 6.325.3.4 int Arc::XMLNode::AttributesSize (void) const

Returns number of attributes of node

#### 6.325.3.5 XMLNode Arc::XMLNode::Child (int *n* = `0`) const

Returns **XMLNode** (p. 547) instance representing n-th child of XML element. If such does not exist invalid **XMLNode** (p. 547) instance is returned Returns **XMLNode** (p. 547) instance representing n-th child of XML element. If such does not exist invalid **XMLNode** (p. 547) instance is returned

#### 6.325.3.6 void Arc::XMLNode::Destroy (void)

Destroys underlying XML element. XML element is unlinked from XML tree and destroyed. After this operation **XMLNode** (p. 547) instance becomes invalid

#### 6.325.3.7 void Arc::XMLNode::Exchange (XMLNode & *node*)

Exchanges XML (sub)trees. Following conbinations are possible If either this ir node are refering owned XML tree (top level node) then references are simply excanged. This opearationis fast. If both this and node are refering to XML (sub)tree of different documents then (sub)trees are exchahed between docomments. If both this and node are refering to XML (sub)tree of same document then (sub)trees are moved inside document. The main reason for this method is to provide effective way to insert one XML document inside another. One should take into account that if any of exchanged nodes is top level it must be also owner of document. Otherwise method will fail. If both nodes are top level owners and/or invlaid nodes then this method is identical to **Swap()** (p. 556).

#### 6.325.3.8 std::string Arc::XMLNode::FullName (void) const `[inline]`

Returns prefix:name of XML node

References Name(), and Prefix().

#### 6.325.3.9 XMLNode Arc::XMLNode::Get (const std::string & *name*) const `[inline]`

Same as operator[]

References operator[ ]().

#### 6.325.3.10 void Arc::XMLNode::GetDoc (std::string & *out_xml_str*, bool *user_friendly* = `false`) const

Fills argument with whole XML document textual representation

#### 6.325.3.11 XMLNode Arc::XMLNode::GetRoot (void)

Get the root node from any child node of the tree

#### 6.325.3.12 void Arc::XMLNode::GetXML (std::string & *out_xml_str*, const std::string & *encoding*, bool *user_friendly* = `false`) const

Fills argument with this instance XML subtree textual representation if the XML subtree is corresponding to the encoding format specified in the argument, e.g. utf-8

#### 6.325.3.13 void Arc::XMLNode::GetXML (std::string & *out_xml_str*, bool *user_friendly* = `false`) const

Fills argument with this instance XML subtree textual representation

#### 6.325.3.14 void Arc::XMLNode::Move (XMLNode & *node*)

Moves content of this XML (sub)tree to node This opeartion is similar to New except that XML (sub)tree to refered by this is destroyed. This method is more effective than combination of **New()** (p. 552) and **Destroy()** (p. 550) because internally it is optimized not to copy data if not needed. The main purpose of this is to effectively extract part of XML document.

#### 6.325.3.15 void Arc::XMLNode::Name (const std::string & *name*) `[inline]`

Assigns new name to XML node

References Name().

#### 6.325.3.16 void Arc::XMLNode::Name (const char ∗ *name*)

Assigns new name to XML node

#### 6.325.3.17 std::string Arc::XMLNode::Name (void) const

Returns name of XML node

Referenced by FullName(), and Name().

#### 6.325.3.18 std::string Arc::XMLNode::Namespace (void) const

Returns namespace URI of XML node

#### 6.325.3.19 std::string Arc::XMLNode::NamespacePrefix (const char * *urn*)

Returns prefix of specified namespace. Empty string if no such namespace.

#### 6.325.3.20 NS Arc::XMLNode::Namespaces (void)

Returns namespaces known at this node

#### 6.325.3.21 void Arc::XMLNode::Namespaces (const NS & *namespaces*, bool *keep* = `false`, int *recursion* = `-1`)

Assigns namespaces of XML document at point specified by this instance. If namespace already exists it gets new prefix. New namespaces are added. It is useful to apply this method to XML being processed in order to refer to it's elements by known prefix. If keep is set to false existing namespace definition residing at this instance and below are removed (default beavior). If recursion is set to positive number then depth of prefix replacement is limited by this number (0 limits it to this node only). For unlimted recursion use -1. If recursion is limited then value of keep is ignored and existing namespaces are always kept.

#### 6.325.3.22 void Arc::XMLNode::New (XMLNode & *node*) const

Creates a copy of XML (sub)tree. If object does not represent whole document - top level document is created. 'new_node' becomes a pointer owning new XML document.

#### 6.325.3.23 XMLNode Arc::XMLNode::NewAttribute (const std::string & *name*) `[inline]`

Creates new attribute with specified name.

References NewAttribute().

#### 6.325.3.24 XMLNode Arc::XMLNode::NewAttribute (const char * *name*)

Creates new attribute with specified name.

Referenced by NewAttribute().

#### 6.325.3.25 XMLNode Arc::XMLNode::NewChild (const XMLNode & *node*, int *n* = `-1`, bool *global_order* = `false`)

Link a copy of supplied XML node as child. Returns instance refering to new child. XML element is a copy of supplied one but not owned by returned instance

#### 6.325.3.26 XMLNode Arc::XMLNode::NewChild (const std::string & *name*, const NS & *namespaces*, int *n* = `-1`, bool *global_order* = `false`) `[inline]`

Same as **NewChild(const char*,const NS&,int,bool)** (p. 553)

References NewChild().

#### 6.325.3.27 XMLNode Arc::XMLNode::NewChild (const char * *name*, const NS & *namespaces*, int *n* = `-1`, bool *global_order* = `false`)

Creates new child XML element at specified position with specified name and namespaces. For more information look at **NewChild(const char*,int,bool)** (p. 553)

#### 6.325.3.28 XMLNode Arc::XMLNode::NewChild (const std::string & *name*, int *n* = `-1`, bool *global_order* = `false`) `[inline]`

Same as **NewChild(const char*,int,bool)** (p. 553)

References NewChild().

#### 6.325.3.29 XMLNode Arc::XMLNode::NewChild (const char * *name*, int *n* = `-1`, bool *global_order* = `false`)

Creates new child XML element at specified position with specified name. Default is to put it at end of list. If global order is true position applies to whole set of children, otherwise only to children of same name. Returns created node.

Referenced by NewChild().

#### 6.325.3.30 Arc::XMLNode::operator bool (void) const `[inline]`

Returns true if instance points to XML element - valid instance

References is_temporary_.

#### 6.325.3.31 Arc::XMLNode::operator std::string (void) const

Returns textual content of node excluding content of children nodes

#### 6.325.3.32 bool Arc::XMLNode::operator! (void) const `[inline]`

Returns true if instance does not point to XML element - invalid instance

References is_temporary_.

#### 6.325.3.33 bool Arc::XMLNode::operator!= (const char * *str*) `[inline]`

This operator is needed to avoid ambiguity

#### 6.325.3.34 bool Arc::XMLNode::operator!= (const std::string & *str*) `[inline]`

This operator is needed to avoid ambiguity

#### 6.325.3.35 bool Arc::XMLNode::operator!= (bool *val*) `[inline]`

This operator is needed to avoid ambiguity

#### 6.325.3.36 bool Arc::XMLNode::operator!= (const XMLNode & *node*) `[inline]`

Returns false if 'node' represents same XML element

#### 6.325.3.37 void Arc::XMLNode::operator++ (void)

Convenience operator to switch to next element of same name. If there is no such node this object becomes invalid.

#### 6.325.3.38 void Arc::XMLNode::operator-- (void)

Convenience operator to switch to previous element of same name. If there is no such node this object becomes invalid.

#### 6.325.3.39 XMLNode& Arc::XMLNode::operator= (const XMLNode & *node*)

Make instance refer to another XML node. Ownership is not inherited.

#### 6.325.3.40 XMLNode& Arc::XMLNode::operator= (const std::string & *content*) `[inline]`

Sets textual content of node. All existing children nodes are discarded.

References operator=().

#### 6.325.3.41 XMLNode& Arc::XMLNode::operator= (const char ∗ *content*)

Sets textual content of node. All existing children nodes are discarded.

Referenced by operator=(), and Set().

#### 6.325.3.42 bool Arc::XMLNode::operator== (const char ∗ *str*) `[inline]`

This operator is needed to avoid ambiguity

#### 6.325.3.43 bool Arc::XMLNode::operator== (const std::string & *str*) `[inline]`

This operator is needed to avoid ambiguity

#### 6.325.3.44 bool Arc::XMLNode::operator== (bool *val*) `[inline]`

This operator is needed to avoid ambiguity

#### 6.325.3.45 bool Arc::XMLNode::operator== (const XMLNode & *node*) `[inline]`

Returns true if 'node' represents same XML element

Referenced by Same().

#### 6.325.3.46 XMLNode Arc::XMLNode::operator[ ] (int *n*) const

Returns **XMLNode** (p. 547) 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]

#### 6.325.3.47 XMLNode Arc::XMLNode::operator[ ] (const std::string & *name*) const `[inline]`

Similar to previous method

References operator[ ]().

#### 6.325.3.48 XMLNode Arc::XMLNode::operator[ ] (const char ∗ *name*) const

Returns **XMLNode** (p. 547) 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. 547) instance is returned

Referenced by Get(), and operator[ ]().

#### 6.325.3.49 XMLNode Arc::XMLNode::Parent (void)

Get the parent node from any child node of the tree

#### 6.325.3.50 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.

#### 6.325.3.51 std::string Arc::XMLNode::Prefix (void) const

Returns namespace prefix of XML node

Referenced by FullName().

#### 6.325.3.52 bool Arc::XMLNode::ReadFromFile (const std::string & *file_name*)

Read XML document from file and associate it with this node

#### 6.325.3.53 bool Arc::XMLNode::ReadFromStream (std::istream & *in*)

Read XML document from stream and associate it with this node

#### 6.325.3.54 void Arc::XMLNode::Replace (const XMLNode & *node*)

Makes a copy of supplied XML node and makes this instance refere to it

#### 6.325.3.55 bool Arc::XMLNode::Same (const XMLNode & *node*) `[inline]`

Returns true if 'node' represents same XML element - for bindings

References operator==().

#### 6.325.3.56 bool Arc::XMLNode::SaveToFile (const std::string & *file_name*) const

Save string representation of node to file

#### 6.325.3.57 bool Arc::XMLNode::SaveToStream (std::ostream & *out*) const

Save string representation of node to stream

#### 6.325.3.58 void Arc::XMLNode::Set (const std::string & *content*) `[inline]`

Same as operator=. Used for bindings.

References operator=().

#### 6.325.3.59 int Arc::XMLNode::Size (void) const

Returns number of children nodes

#### 6.325.3.60 void Arc::XMLNode::Swap (XMLNode & *node*)

Swaps XML (sub)trees to this this and node refer. For XML subtrees this method is not anyhow different then using combinaiion **XMLNode** (p. 547) tmp=∗this; ∗this=node; node=tmp; But in case of either this or node owning XML document ownership is swapped too. And this is a main purpose of **Swap()** (p. 556) method.

#### 6.325.3.61 bool Arc::XMLNode::Validate (const std::string & *schema_file*, std::string & *err_msg*)

Remove all eye-candy information leaving only informational parts ∗ void Purify(void); XML schema validation against the schema file defined as argument

#### 6.325.3.62 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. 547) 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** (p. 360) is run on whole XML document but only the elements belonging to this XML subtree are returned.

### 6.325.4 Field Documentation

#### 6.325.4.1 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.

#### 6.325.4.2 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

## 6.326 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)

### 6.326.1 Detailed Description

Container for multiple **XMLNode** (p. 547) elements

### 6.326.2 Constructor & Destructor Documentation

#### 6.326.2.1 Arc::XMLNodeContainer::XMLNodeContainer (void)

Default constructor

#### 6.326.2.2 Arc::XMLNodeContainer::XMLNodeContainer (const XMLNodeContainer &)

Copy constructor. Add nodes from argument. Nodes owning XML document are copied using **AddNew()** (p. 559). Not owning nodes are linked using **Add()** (p. 558) method.

### 6.326.3 Member Function Documentation

#### 6.326.3.1 void Arc::XMLNodeContainer::Add (const std::list< XMLNode > &)

Link multiple XML subtrees to container.

#### 6.326.3.2 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.

#### 6.326.3.3 void Arc::XMLNodeContainer::AddNew (const std::list< XMLNode > &)

Copy multiple XML subtrees to container.

#### 6.326.3.4 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.

#### 6.326.3.5 std::list<XMLNode> Arc::XMLNodeContainer::Nodes (void)

Returns all stored nodes.

#### 6.326.3.6 XMLNodeContainer& Arc::XMLNodeContainer::operator= (const XMLNodeContainer &)

Same as copy constructor with current nodes being deleted first.

#### 6.326.3.7 XMLNode Arc::XMLNodeContainer::operator[ ] (int)

Returns n-th node in a store.

#### 6.326.3.8 int Arc::XMLNodeContainer::Size (void)

Return number of refered/stored nodes.

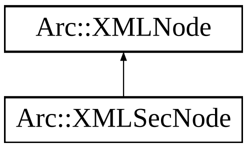
The documentation for this class was generated from the following file:

- XMLNode.h

## 6.327 Arc::XMLSecNode Class Reference

Extends **XMLNode** (p. 547) 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)

### 6.327.1 Detailed Description

Extends **XMLNode** (p. 547) class to support XML security operation. All **XMLNode** (p. 547) methods are exposed by inheriting from **XMLNode** (p. 547). **XMLSecNode** (p. 560) itself does not own node, instead it uses the node from the base class **XMLNode** (p. 547).

### 6.327.2 Constructor & Destructor Documentation

#### 6.327.2.1 Arc::XMLSecNode::XMLSecNode (XMLNode & *node*)

Create a object based on an **XMLNode** (p. 547) instance.

### 6.327.3 Member Function Documentation

#### 6.327.3.1 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

#### 6.327.3.2 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

#### 6.327.3.3 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).

#### 6.327.3.4 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

#### 6.327.3.5 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>

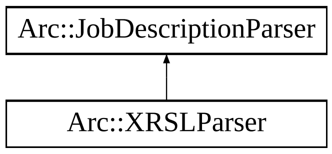
***ca_path*** The CA directory; either ca_file or ca_path should be set.

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

- XMLSecNode.h

## 6.328 Arc::XRSLParser Class Reference

Inheritance diagram for Arc::XRSLParser::



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

- XRSLParser.h

# Chapter 7

# File Documentation

## 7.1 URL.h File Reference

Class to hold general URL's. `#include <iostream>`

`#include <list>`

`#include <map>`

`#include <string>`

### Data Structures

- class **Arc::URL**
- class **Arc::URLLocation**

  *Class to hold a resolved* ***URL*** *(*p. *456) location.*

- class **Arc::PathIterator**

  *Class to iterate through elements of path.*

### Namespaces

- namespace **Arc**

### Defines

- #define **RC_DEFAULT_PORT** 389

### Functions

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

### 7.1.1 Detailed Description

Class to hold general URL's. The URL is split into protocol, hostname, port and path. This class tries to follow RFC 3986 for spliting URLs at least for protocol + host part. It also accepts local file paths which are converted to `file:path`. Usual system dependant file paths are supported. Relative paths are converted to absolute ones by prepending them with current working directory path. File path can't start from # symbol (why?). If string representation of URL starts from '@' then it is treated as path to file containing list of URLs. Simple URL is parsed in following way: [protocol:][//[username:**passwd** (p. 314)@][host][:port]][;urloptions[;...]][/path[?httpoption[&...]][:metadataoption[:...]]] The 'protocol' and 'host' parts are treated as case-insensitive and to avoid confusion are converted to lowercase in constructor. Note that 'path' is always converted to absolute path in constructor. Meaning of 'absolute' may depend upon URL type. For generic URL and local POSIX file paths that means path starts from / like /path/to/file For Windows paths absolute path may look like C: It is important to note that path still can be empty. For referencing local file using absolute path on POSIX filesystem one may use either `file:///path/to/file` or `file:/path/to/file` Relative path will look like `file:to/file` For local Windows files possible URLs are `file:C:\path\to\file` `file:to` URLs representing LDAP resources have different structure of options following 'path' part ldap://host[:port][;urloptions[;...]][/path[?attributes[?scope[?filter]]]] For LDAP URLs paths are converted from /key1=value1/.../keyN=valueN notation to keyN=valueN,...,key1=value1 and hence path does not contain leading /. If LDAP URL initially had path in second notation leading / is treated as separator only and is stripped. URLs of indexing services optionally may have locations specified before 'host' part protocol://[location[;location[;...]]@][host][:port]... The structure of 'location' element is protocol specific.

### 7.1.2 Define Documentation

#### 7.1.2.1 #define RC_DEFAULT_PORT 389

Default ports for different protocols

# Index