Hosting Environment (Daemon)

Generated by Doxygen 1.7.3

Wed Apr 6 2011 14:18:56

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

  Utility base class for ***MCC*** *(p. 252).*

- class **ClientTCP**

  Class for setting up a ***MCC*** *(p. 252) chain for TCP communication.*

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

  Class for setting up a ***MCC*** *(p. 252) 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. *56).*

- class **ExecutionTarget**

  ExecutionTarget *(*p. *177).*

- class **Job**

  Job *(*p. *215).*

- class **JobController**

  Base class for the JobControllers.

- class **JobControllerLoader**
- class **JobControllerPluginArgument**
- class **Range**
- class **ScalableTime**
- class **ScalableTime< int >**
- class **JobIdentificationType**
- class **ExecutableType**
- class **NotificationType**
- class **ApplicationType**
- class **ResourceSlotType**
- class **DiskSpaceRequirementType**
- class **ResourcesType**
- class **FileType**
- class **JobDescription**
- class **JobDescriptionParser**
- class **JobDescriptionParserLoader**
- class **JobState**
- class **JobSupervisor**

  % ***JobSupervisor*** *(*p. *233) class*

- class **Software**

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

- class **SoftwareRequirement**

  Class used to express and resolve version requirements on software.

- class **Submitter**

  Base class for the Submitters.

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

- class **TargetGenerator**

  Target generation class

- class **TargetRetriever**

  TargetRetriever base class

- class **TargetRetrieverLoader**
- class **TargetRetrieverPluginArgument**
- 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 **FileAccess**

  Defines interface for accessing filesystems.

- class **FileLock**

  A general file locking class.

- class **IniConfig**
- class **IntraProcessCounter**

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

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

  A class for log messages.

- class **LogDestination**

  A base class for log destinations.

- class **LogStream**

  A class for logging to ostreams.

- class **LogFile**

  A class for logging to files.

- class **LoggerContext**

  Container for logger configuration.

- class **Logger**

  A logger class.

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

  Base class for per-thread object.

- class **SimpleCondition**

  Simple triggered condition.

- class **SimpleCounter**
- class **TimedMutex**
- class **SharedMutex**
- class **ThreadRegistry**
- class **ThreadInitializer**
- class **URL**
- class **URLLocation**

  *Class to hold a resolved **URL** (p. 387) location.*

- class **PathIterator**

  Class to iterate through elements of path.

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

  User configuration class

- class **CertEnvLocker**
- class **AutoPointer**

  Wrapper for pointer with automatic destruction.

- class **CountedPointer**

  Wrapper for pointer with automatic destruction and mutiple references.

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

  Wrapper for LibXML library Tree interface.

- class **XMLNodeContainer**
- class **CredentialError**
- class **Credential**
- class **VOMSACInfo**
- class **VOMSTrustList**
- class **CredentialStore**
- class **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. 74) class.*

- class **DataBuffer**

  Represents set of buffers.

- class **DataCallback**

- class **DataHandle**

  This class is a wrapper around the ***DataPoint*** *(*p. *122) class.*

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

  This base class is an abstraction of ***URL*** *(*p. *387).*

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

  This is a kind of generalized file handle.

- class **DataPointIndex**

  Complements ***DataPoint*** *(*p. *122) with attributes common for Indexing* ***Service*** *(*p. *341) URLs.*

- class **DataSpeed**

  Keeps track of average and instantaneous transfer speed.

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

  FileInfo *(*p. *191) 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. 319) - extension of **Service** (p. 341) 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 **PluginDesc**

  Description of plugin.

- class **ModuleDesc**

  Description of loadable module.

- class **PluginsFactory**

  Generic ARC plugins loader.

- class **MCCInterface**

  *Interface for communication between **MCC** (p. 252), **Service** (p. 341) and **Plexer** (p. 303) objects.*

- class **MCC**

  ***Message** (p. 262) Chain Component - base class for every **MCC** (p. 252) plugin.*

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

  *A class for communication of **MCC** (p. 252) processing results.*

- class **MCCLoader**

  *Creator of **Message** (p. 262) Component Chains (**MCC** (p. 252)).*

- class **ChainContext**

  Interface to chain specific functionality.

- class **MessagePayload**

  Base class for content of message passed through chain.

- class **MessageContextElement**

  Top class for elements contained in message context.

- class **MessageContext**

  Handler for content of message context.

- class **MessageAuthContext**

  Handler for content of message auth∗ context.

- class **Message**

  Object being passed through chain of MCCs.

- class **AttributeIterator**

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

- class **MessageAttributes**

  A class for storage of attribute values.

- class **MessageAuth**

  Contains authencity information, authorization tokens and decisions.

- class **PayloadRawInterface**

*Random Access Payload for **Message** (p. 262) objects.*

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

  Raw byte multi-buffer.

- class **PayloadSOAP**

  *Payload of **Message** (p. 262) with SOAP content.*

- class **PayloadStreamInterface**

  *Stream-like Payload for **Message** (p. 262) object.*

- class **PayloadStream**

  POSIX handle as Payload.

- class **PlexerEntry**

  *A pair of label (regex) and pointer to **MCC** (p. 252).*

- class **Plexer**

  *The **Plexer** (p. 303) 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. 336) attributes.*

- class **Service**

  ***Service** (p. 341) - last component in a **Message** (p. 262) Chain.*

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

  ***Message** (p. 262) 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. 315).*

- class **UsernameToken**

  *Interface for manipulation of WS-Security according to Username Token **Profile** (p. 315).*

- class **X509Token**

  *Class for manipulating X.509 Token **Profile** (p. 315).*

- class **PayloadWSRF**

  *This class combines **MessagePayload** (p. 271) with **WSRF** (p. 441).*

- 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. *441) message.*

- class **WSRFBaseFault**

  *Base class for **WSRF** (*p. *441) fault messages.*

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

  *Extends **XMLNode** (*p. *465) class to support XML security operation.*

Typedefs

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

Enumerations

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

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

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

Functions

- std::ostream & **operator**<< (std::ostream &, const **Period** &)
- std::ostream & **operator**<< (std::ostream &, const **Time** &)
- std::string **TimeStamp** (const **TimeFormat** &=Time::GetFormat())
- std::string **TimeStamp** (**Time**, const **TimeFormat** &=Time::GetFormat())
- bool **FileCopy** (const std::string &source_path, const std::string &destination_-path, uid_t uid, gid_t gid)
- bool **FileCopy** (const std::string &source_path, const std::string &destination_-path)
- bool **FileCopy** (const std::string &source_path, int destination_handle)
- bool **FileCopy** (int source_handle, const std::string &destination_path)
- bool **FileCopy** (int source_handle, int destination_handle)
- bool **FileRead** (const std::string &filename, std::list< std::string > &data, uid_t uid=0, gid_t gid=0)
- bool **FileCreate** (const std::string &filename, const std::string &data, uid_t uid=0, gid_t gid=0)
- bool **FileStat** (const std::string &path, struct stat ∗st, bool follow_symlinks)
- bool **FileStat** (const std::string &path, struct stat ∗st, uid_t uid, gid_t gid, bool follow_symlinks)
- bool **FileLink** (const std::string &oldpath, const std::string &newpath, bool symbolic)
- bool **FileLink** (const std::string &oldpath, const std::string &newpath, uid_t uid, gid_t gid, bool symbolic)
- std::string **FileReadLink** (const std::string &path)
- std::string **FileReadLink** (const std::string &path, uid_t uid, gid_t gid)
- bool **FileDelete** (const std::string &path)
- bool **FileDelete** (const std::string &path, uid_t uid, gid_t gid)
- bool **DirCreate** (const std::string &path, mode_t mode, bool with_parents=false)
- bool **DirCreate** (const std::string &path, uid_t uid, gid_t gid, mode_t mode, bool with_parents=false)
- bool **DirDelete** (const std::string &path)
- bool **DirDelete** (const std::string &path, uid_t uid, gid_t gid)
- bool **TmpDirCreate** (std::string &path)
- void **GUID** (std::string &guid)
- std::string **UUID** (void)
- std::ostream & **operator**<< (std::ostream &os, **LogLevel** level)
- **LogLevel string_to_level** (const std::string &str)
- bool **istring_to_level** (const std::string &llStr, **LogLevel** &ll)
- bool **string_to_level** (const std::string &str, **LogLevel** &ll)
- std::string **level_to_string** (const **LogLevel** &level)
- **LogLevel old_level_to_level** (unsigned int old_level)
- template<typename T >
  T **stringto** (const std::string &s)
- template<typename T >
  bool **stringto** (const std::string &s, T &t)

- 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=" ", const std::string &start_quotes="", const std::string &end_quotes="")
- std::string **trim** (const std::string &str, const char ∗sep=NULL)
- std::string **strip** (const std::string &str)
- std::string **uri_unescape** (const std::string &str)
- std::string **convert_to_rdn** (const std::string &dn)
- bool **CreateThreadFunction** (void(∗func)(void ∗), void ∗arg, **SimpleCounter** ∗count=NULL)
- std::list< **URL** > **ReadURLList** (const **URL** &urllist)
- std::string **GetEnv** (const std::string &var)
- std::string **GetEnv** (const std::string &var, bool &found)
- bool **SetEnv** (const std::string &var, const std::string &value, bool overwrite=true)
- void **UnsetEnv** (const std::string &var)
- 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< **VOMSACInfo** > &output, bool verify=true)
- bool **parseVOMSAC** (const **Credential** &holder_cred, const std::string &ca_cert_dir, const std::string &ca_cert_file, const **VOMSTrustList** &vomscert_trust_dn, std::vector< **VOMSACInfo** > &output, bool verify=true)
- char ∗ **VOMSDecode** (const char ∗data, int size, int ∗j)
- const std::string **get_property** (const **Arc::Credential** &u, const std::string property)
- 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/`) **JobDescription** (p. 227) The **JobDescription** (p. 227) 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** (p. 228) which tries to parse the passed source using a number of different parsers. The parser method is complemented by the **JobDescription::UnParse** (p. 229) 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. 230) The **JobDescriptionParser** (p. 230) 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.

Credential (p. 100) 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. 100) 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 **Credential** (p. 100) 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. 265) 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. 265) 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::LogFormat

Output formats.

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

5.1.3.2 enum Arc::LogLevel

Logging levels.

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

5.1.3.3 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. 262) does not fit into expected protocol.

BUSY_ERROR There is no destination configured for this message.

SESSION_CLOSE **Message** (p. 262) can't be processed now.

5.1.3.4 enum Arc::WSAFault

WS-Addressing possible faults.

Enumerator:

WSAFaultUnknown This is not a fault

WSAFaultInvalidAddressingHeader This is not a WS-Addressing fault

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. 262) payload.

If no buffer is present or if payload is not of **PayloadRawInterface** (p. 286) type NULL is returned.

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

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. If count parameter not NULL then corresponding object will be incremented before function returns and then decremented then thread finished. 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::DirCreate (const std::string & *path,* uid_t *uid,* gid_t *gid,* mode_t *mode,* bool *with_parents* = `false`)

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

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

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

5.1.4.7 bool Arc::FileCopy (const std::string & *source_path,* const std::string & *destination_path,* uid_t *uid,* gid_t *gid*)

Utility functions for handling files and directories. Those functions offer posibility to access files and directories under user and group ids different from those of current user. Id switching is done in way safe for muti-threaded application. If any of specified ids is 0 then such id is not switched and current id is used instead. Copy file source_-path to file destination_path. Specified uid and gid are used for accessing filesystem.

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

Simple method to create a new file containing given data. Specified uid and gid are used for accessing filesystem. An existing file is overwritten with the new data. Permissions of the created file are determined using the current umask. For more complex file handling or large files, FileOpen() should be used. If protected access is required, **FileLock** (p. 191) should be used in addition to FileRead. If uid/gid are zero then no real switch of uid/gid is done.

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

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

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

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

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

Simple method to read file content from filename. Specified uid and gid are used for accessing filesystem. The content is split into lines with the new line character removed, and the lines are returned in the data list. If protected access is required, **FileLock** (p. 191) should be used in addition to FileRead.

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

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

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

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

5.1.4.14 bool Arc::final_xmlsec (void)

Finalize the xml security library

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

Get certificate in string format from certificate file

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

Get public key in string format from certificate file

5.1.4.17 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-----")

5.1.4.18 xmlSecKey∗ Arc::get_key_from_keyfile (const char ∗ *keyfile*)

Get key in xmlSecKey structure from key file

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

Get key in xmlSecKey structure from key in string format

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

Generate a new child **XMLNode** (p. 465) with specified name

5.1.4.21 const std::string Arc::get_property (const Arc::Credential & *u,* const std::string *property*)

Extract the needed field from the certificate

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

Utilities for generating unique identifiers in the form 12345678-90ab-cdef-1234-567890abcdef.

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

5.1.4.23 bool Arc::init_xmlsec (void)

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

5.1.4.24 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. 38).
ll	a **Arc::LogLevel** (p. 38) reference which will be set to the matching **Arc::LogLevel** (p. 38) upon successful parsing.

Returns

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

See also

LogLevel (p. 38)

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

Load public key from a certificate file into key manager

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

Load public key from a certificate string into key manager

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

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

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

Load trusted certificate from certificate file into key manager

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

Load trusted certificate from cetrtificate string into key manager

5.1.4.30 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

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

Returns true if underlying XML elements have same names

5.1.4.32 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

5.1.4.33 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

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

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

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

Returns true if underlying XML elements belong to same namespaces

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

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

5.1.4.37 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. 57) it is advisable to call it after **ArcLocation::Init()** (p. 57).

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

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

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

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

5.1.4.40 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.

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

Parse the certificate, and output the attributes.

Parameters

holder	The proxy certificate which includes the voms specific formated AC.
ca_cert_dir	The trusted certificates which are used to verify the certificate which is used to sign the AC
ca_cert_file	The same as ca_cert_dir except it is a file instead of a directory. Only one of them need to be set
vomsdir	The directory which include ∗.lsc file for each vo. For instance, a vo called "knowarc.eu" should have file $prefix/vomsdir/knowarc/voms.knowarc.eu.lsc which contains on the first line the DN of the VOMS server, and on the second line the corresponding CA DN: /O=Grid/O=NorduGrid/OU=KnowARC/CN=voms.knowarc.eu /O=Grid/O=NorduGrid/CN=NorduGrid Certification Authority See more in: `https://twiki.cern.ch/twiki/bin/view/LCG/VomsFAQforServiceManagers`
output	The parsed attributes (Role and Generic Attribute) . Each attribute is stored in element of a vector as a string. It is up to the consumer to understand the meaning of the attribute. There are two types of attributes stored in VOMS AC: AC_IETFATTR, AC_FULL_ATTRIBUTES. The AC_IETFATTR will be like /Role=Employee/Group=Tester/Capability=NULL The AC_FULL_ATTRIBUTES will be like knowarc:Degree=PhD (qualifier::name=value) In order to make the output attribute values be identical, the voms server information is added as prefix of the original attributes in AC. for AC_FULL_ATTRIBUTES, the voname + hostname is added: /voname=knowarc.eu/hostname=arthur.hep.lu.se:15001//knowarc.eu/coredev:attribute1=1 for AC_IETFATTR, the 'VO' (voname) is added: /VO=knowarc.eu/Group=coredev/Role=NULL/Capability=NULL /VO=knowarc.eu/Group=testers/Role=NULL/Capability=NULL

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

Parameters

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

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

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

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

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

callback method for inputing passphrase of key file

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

Conversion to string.

Conversion from StatusKind to string.

Parameters

kind	The StatusKind to convert.

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

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

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

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

5.1.4.47 bool Arc::TmpDirCreate (std::string & *path*)

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

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

5.1.4.48 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

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

Makes WS-Addressing fault.

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

5.1.4.50 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. 243) 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. 308) 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:

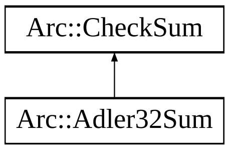
- VOMSAttribute.h

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:

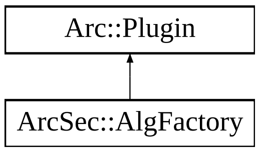
- CheckSum.h

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. 54) is in charge of creating **CombiningAlg** (p. 84) 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

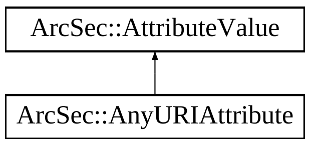
The object of **CombiningAlg** (p. 84)

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 bool **equal** (**AttributeValue** ∗other, bool check_id=true)
- 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. 64).

6.20.1.2 virtual bool ArcSec::AnyURIAttribute::equal (AttributeValue * *value*, bool *check_id* =true) [virtual]

Evluate whether "this" equale to the parameter value

Implements **ArcSec::AttributeValue** (p. 64).

6.20.1.3 std::string ArcSec::AnyURIAttribute::getId ( ) [inline, virtual]

Get the AttributeId of the <Attribute>

Implements **ArcSec::AttributeValue** (p. 64).

6.20.1.4 virtual std::string ArcSec::AnyURIAttribute::getType ( ) [inline, virtual]

Get the DataType of the <Attribute>

Implements **ArcSec::AttributeValue** (p. 65).

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

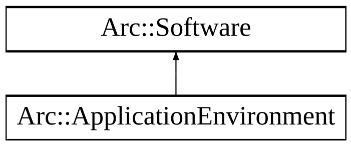
- AnyURIAttribute.h

6.21 Arc::ApplicationEnvironment Class Reference

ApplicationEnvironment (p. 56).

`#include <ExecutionTarget.h>`

Inheritance diagram for Arc::ApplicationEnvironment:



6.21.1 Detailed Description

ApplicationEnvironment (p. 56). The ApplicationEnviroment is closely related to the definition given in GLUE2. By extending the **Software** (p. 347) 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::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** ()

6.23.1 Detailed Description

Determines ARC installation location.

6.23.2 Member Function Documentation

6.23.2.1 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.

6.23.2.2 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

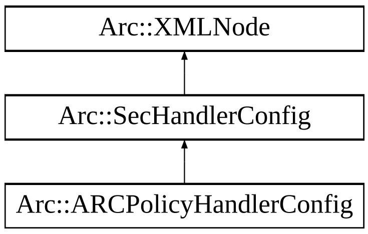
6.24 ArcSec::ArcPeriod Struct Reference

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

- DateTimeAttribute.h

6.25 Arc::ARCPolicyHandlerConfig Class Reference

Inheritance diagram for Arc::ARCPolicyHandlerConfig:



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

- ClientInterface.h

6.26 ArcSec::Attr Struct Reference

Attr (p. 58) contains a tuple of attribute type and value.

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

6.26.1 Detailed Description

Attr (p. 58) contains a tuple of attribute type and value.

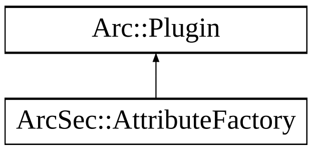
The documentation for this struct was generated from the following file:

- Request.h

6.27 ArcSec::AttributeFactory Class Reference

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

Inheritance diagram for ArcSec::AttributeFactory:



6.27.1 Detailed Description

Base attribute factory class

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

- AttributeFactory.h

6.28 Arc::AttributeIterator Class Reference

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

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

Public Member Functions

- **AttributeIterator** ()
- const std::string & **operator∗** () const
- const std::string ∗ **operator->** () const
- const std::string & **key** (void) const
- const **AttributeIterator** & **operator++** ()
- **AttributeIterator operator++** (int)
- bool **hasMore** () const

Protected Member Functions

- **AttributeIterator** (**AttrConstIter** begin, **AttrConstIter** end)

Protected Attributes

- **AttrConstIter current_**
- **AttrConstIter end_**

Friends

- class **MessageAttributes**

6.28.1 Detailed Description

A const iterator class for accessing multiple values of an attribute. This is an iterator class that is used when accessing multiple values of an attribute. The getAll() method of the **MessageAttributes** (p. 265) class returns an **AttributeIterator** (p. 59) 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;
```

6.28.2 Constructor & Destructor Documentation

6.28.2.1 Arc::AttributeIterator::AttributeIterator ( )

Default constructor.

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

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

Protected constructor used by the **MessageAttributes** (p. 265) 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. 265) class.

Parameters

begin	A const_iterator pointing to the first matching key-value pair in the internal multimap of the **MessageAttributes** (p. 265) class.
end	A const_iterator pointing to the first key-value pair in the internal multimap of the **MessageAttributes** (p. 265) class where the key is larger than the key searched for.

6.28.3 Member Function Documentation

6.28.3.1 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.

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

The key of attribute.

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

6.28.3.3 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.

6.28.3.4 const **AttributeIterator**& Arc::AttributeIterator::operator++ ( )

The prefix advance operator.

Advances the iterator to the next value. Works intuitively.

Returns

A const reference to this iterator.

6.28.3.5 **AttributeIterator** Arc::AttributeIterator::operator++ (int)

The postfix advance operator.

Advances the iterator to the next value. Works intuitively.

Returns

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

6.28.3.6 const std::string$*$ Arc::AttributeIterator::operator-$>$ ( ) const

The arrow operator.

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

6.28.4 Friends And Related Function Documentation

6.28.4.1 friend class **MessageAttributes** `[friend]`

The **MessageAttributes** (p. 265) class is a friend.

The constructor that creates an **AttributeIterator** (p. 59) that is connected to the internal multimap of the **MessageAttributes** (p. 265) class should not be exposed to the outside, but it still needs to be accessible from the getAll() method of the **MessageAttributes** (p. 265) class. Therefore, that class is a friend.

6.28.5 Field Documentation

6.28.5.1 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. 265) class.

6.28.5.2 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. 265) class where the key is larger than the key searched for.

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

- MessageAttributes.h

6.29 ArcSec::AttributeProxy Class Reference

Interface for creating the **AttributeValue** (p. 63) object, it will be used by **AttributeFactory** (p. 58).

`#include <AttributeProxy.h>`

Public Member Functions

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

6.29.1 Detailed Description

Interface for creating the **AttributeValue** (p. 63) object, it will be used by **AttributeFactory** (p. 58). The **AttributeProxy** (p. 62) 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. 63)

6.29.2 Member Function Documentation

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

Create a **AttributeValue** (p. 63) object according to the information inside the XMLNode as parameter.

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

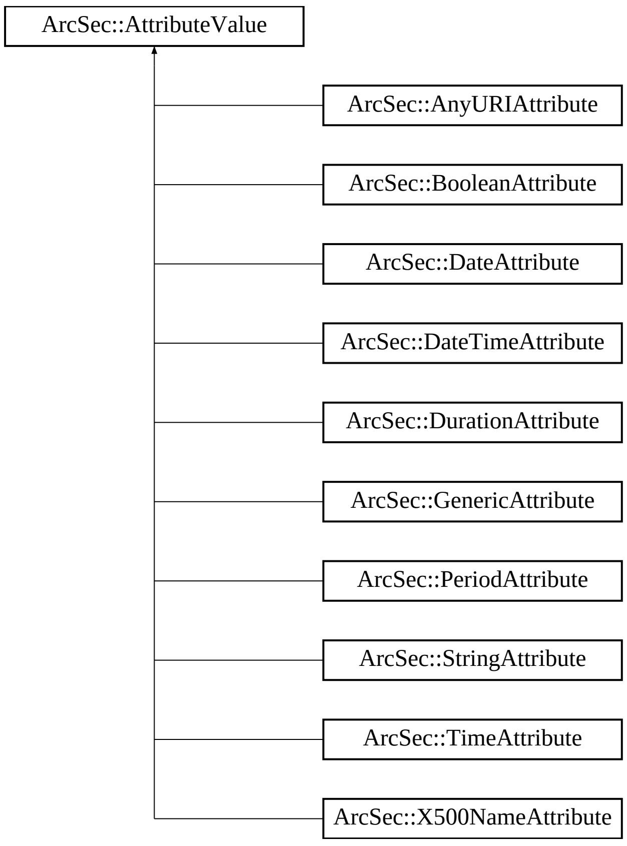
- AttributeProxy.h

6.30 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

6.30.1 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. 366), **DateAttribute** (p. 155), **TimeAttribute** (p. 386), **DurationAttribute** (p. 168), **PeriodAttribute** (p. 301), **AnyURIAttribute** (p. 55), **X500NameAttribute** (p. 462)

6.30.2 Member Function Documentation

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

encode the value in a string format

Implemented in **ArcSec::AnyURIAttribute** (p. 55), **ArcSec::BooleanAttribute** (p. 69), **ArcSec::DateTimeAttribute** (p. 156), **ArcSec::TimeAttribute** (p. 386), **ArcSec::DateAttribute** (p. 155), **ArcSec::DurationAttribute** (p. 168), **ArcSec::PeriodAttribute** (p. 301), **ArcSec::GenericAttribute** (p. 196), **ArcSec::StringAttribute** (p. 367), and **ArcSec::X500NameAttribute** (p. 462).

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

Evluate whether "this" equale to the parameter value

Implemented in **ArcSec::AnyURIAttribute** (p. 56), **ArcSec::BooleanAttribute** (p. 69), **ArcSec::DateTimeAttribute** (p. 156), **ArcSec::TimeAttribute** (p. 387), **ArcSec::DateAttribute** (p. 155), **ArcSec::DurationAttribute** (p. 169), **ArcSec::PeriodAttribute** (p. 301), **ArcSec::GenericAttribute** (p. 196), **ArcSec::StringAttribute** (p. 367), and **ArcSec::X500NameAttribute** (p. 462).

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

Get the AttributeId of the <Attribute>

Implemented in **ArcSec::AnyURIAttribute** (p. 56), **ArcSec::BooleanAttribute** (p. 69), **ArcSec::DateTimeAttribute** (p. 156), **ArcSec::TimeAttribute** (p. 387), **ArcSec::DateAttribute** (p. 155), **ArcSec::DurationAttribute** (p. 169), **ArcSec::PeriodAttribute** (p. 301),

ArcSec::GenericAttribute (p. 196), **ArcSec::StringAttribute** (p. 367), and **ArcSec::X500NameAttribute** (p. 462).

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

Get the DataType of the <Attribute>

Implemented in **ArcSec::AnyURIAttribute** (p. 56), **ArcSec::BooleanAttribute** (p. 69), **ArcSec::DateTimeAttribute** (p. 157), **ArcSec::TimeAttribute** (p. 387), **ArcSec::DateAttribute** (p. 155), **ArcSec::DurationAttribute** (p. 169), **ArcSec::PeriodAttribute** (p. 302), **ArcSec::GenericAttribute** (p. 196), **ArcSec::StringAttribute** (p. 367), and **ArcSec::X500NameAttribute** (p. 462).

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

- AttributeValue.h

6.31 ArcSec::Attrs Class Reference

Attrs (p. 65) is a container for one or more **Attr** (p. 58).

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

6.31.1 Detailed Description

Attrs (p. 65) is a container for one or more **Attr** (p. 58). **Attrs** (p. 65) 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

6.32 ArcSec::AuthzRequest Struct Reference

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

- PDP.h

6.33 ArcSec::AuthzRequestSection Struct Reference

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

6.33.1 Detailed Description

These structure are based on the request schema for **PDP** (p. 297), 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

6.34 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

6.34.1 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

6.35 Arc::Base64 Class Reference

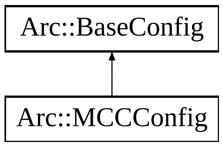
The documentation for this class was generated from the following file:

- Base64.h

6.36 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

6.36.1 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.

6.36.2 Member Function Documentation

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

Add CA directory

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

Add CA file

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

Add certificate

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

Add configuration overlay

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

Adds non-standard location of plugins

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

Add private key

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

Add credentials proxy

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

Read overlay from file

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

Adds configuration part corresponding to stored information into common configuration tree supplied in 'cfg' argument. Returns reference to XML node representing configuration of **ModuleManager** (p. 271)

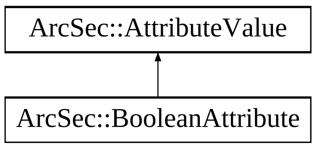
Reimplemented in **Arc::MCCConfig** (p. 258).

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

- ArcConfig.h

6.37 ArcSec::BooleanAttribute Class Reference

Inheritance diagram for ArcSec::BooleanAttribute:



Public Member Functions

- virtual bool **equal** (**AttributeValue** $*$o, bool check_id=true)
- virtual std::string **encode** ()
- std::string **getId** ()
- std::string **getType** ()

6.37.1 Member Function Documentation

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

encode the value in a string format

Implements **ArcSec::AttributeValue** (p. 64).

6.37.1.2 virtual bool ArcSec::BooleanAttribute::equal (AttributeValue $*$ *value*, bool *check_id* = `true`) `[virtual]`

Evluate whether "this" equale to the parameter value

Implements **ArcSec::AttributeValue** (p. 64).

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

Get the AttributeId of the <Attribute>

Implements **ArcSec::AttributeValue** (p. 64).

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

Get the DataType of the <Attribute>

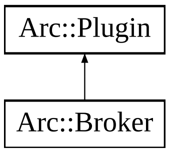
Implements **ArcSec::AttributeValue** (p. 65).

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

- BooleanAttribute.h

6.38 Arc::Broker Class Reference

Inheritance diagram for Arc::Broker:



Public Member Functions

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

Protected Member Functions

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

Protected Attributes

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

6.38.1 Member Function Documentation

6.38.1.1 const ExecutionTarget∗ Arc::Broker::GetBestTarget ( )

Returns next target from the list of **ExecutionTarget** (p. 177) objects.

When first called this method will sort its list of **ExecutionTarget** (p. 177) 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. 177) in the list is returned.

6.38.1.2 void Arc::Broker::PreFilterTargets (std::list< **ExecutionTarget** > & *targets*, const **JobDescription** & *jobdesc*, const std::list< **URL** > & *rejectTargets* = `std::list<` **URL** `>()`)

Filter **ExecutionTarget** (p. 177) objects according to attributes in **JobDescription** (p. 227) object.

Each of the **ExecutionTarget** (p. 177) objects in the passed list will be matched against attributes in the passed **JobDescription** (p. 227) object. For a list of which attributes are considered for matchmaking see appendix B of the libarcclient technical manual (NORDUGRID-TECH-20). If a **ExecutionTarget** (p. 177) object matches the job description, it is added to the internal list of **ExecutionTarget** (p. 177) objects. NOTE: The list of **ExecutionTarget** (p. 177) objects must be available through out the scope of this **Broker** (p. 69) object.

Parameters

targets	A list of **ExecutionTarget** (p. 177) objects to be considered for addition to the **Broker** (p. 69).
jobdesc	**JobDescription** (p. 227) object holding requirements.
rejectTargets	

6.38.1.3 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. 177) which has the shortest queue lenght will be at the begining of the PossibleTargets list

6.38.2 Field Documentation

6.38.2.1 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

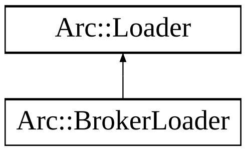
The documentation for this class was generated from the following file:

- Broker.h

6.39 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

6.39.1 Detailed Description

Class responsible for loading **Broker** (p. 69) plugins The **Broker** (p. 69) objects returned by a **BrokerLoader** (p. 71) must not be used after the **BrokerLoader** (p. 71) goes out of scope.

6.39.2 Constructor & Destructor Documentation

6.39.2.1 Arc::BrokerLoader::BrokerLoader ( )

Constructor Creates a new **BrokerLoader** (p. 71).

6.39.2.2 Arc::BrokerLoader::∼BrokerLoader ( )

Destructor Calling the destructor destroys all Brokers loaded by the **BrokerLoader** (p. 71) instance.

6.39.3 Member Function Documentation

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

Retrieve the list of loaded Brokers.

Returns

A reference to the list of Brokers.

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

Load a new **Broker** (p. 69)

Parameters

name	The name of the **Broker** (p. 69) to load.
usercfg	The **UserConfig** (p. 399) object for the new **Broker** (p. 69).

Returns

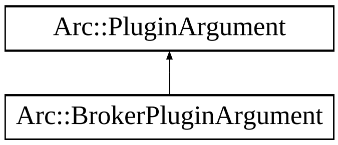
A pointer to the new **Broker** (p. 69) (NULL on error).

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

- Broker.h

6.40 Arc::BrokerPluginArgument Class Reference

Inheritance diagram for Arc::BrokerPluginArgument:



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

- Broker.h

6.41 Arc::ByteArray Class Reference

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

- ByteArray.h

6.42 Arc::CacheParameters Struct Reference

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

6.42.1 Detailed Description

Contains data on the parameters of a cache.

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

- FileCache.h

6.43 ArcCredential::cert_verify_context Struct Reference

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

- CertUtil.h

6.44 Arc::CertEnvLocker Class Reference

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

- UserConfig.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. 260) object. It is accessible for **MCC** (p. 252) and **Service** (p. 341) components and provides an interface to manipulate chains stored in **Loader** (p. 238). 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. 308) object

References Arc::Loader::factory_.

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

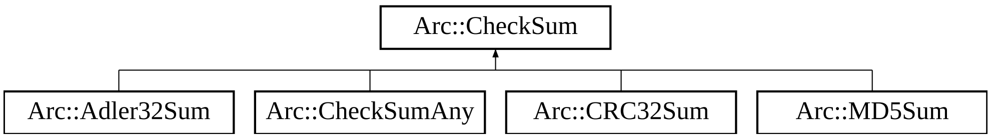
- MCCLoader.h

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. 113) class

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

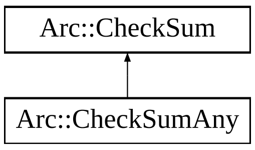
- CheckSum.h

6.47 Arc::CheckSumAny Class Reference

Wraper for **CheckSum** (p. 74) class.

`#include <CheckSum.h>`

Inheritance diagram for Arc::CheckSumAny:



6.47.1 Detailed Description

Wraper for **CheckSum** (p. 74) 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:

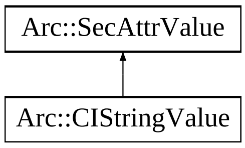
- CheckSum.h

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. 338). 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

Reimplemented from **Arc::SecAttrValue** (p. 338).

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

This function returns false if the string is empty or uninitialized

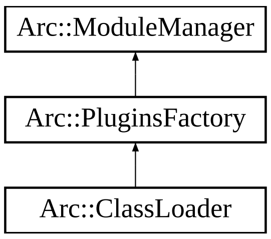
Reimplemented from **Arc::SecAttrValue** (p. 339).

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

- CIStringValue.h

6.49 Arc::ClassLoader Class Reference

Inheritance diagram for Arc::ClassLoader:

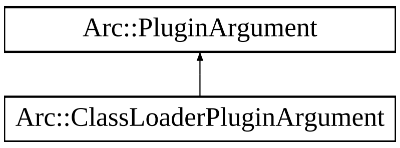


The documentation for this class was generated from the following file:

- ClassLoader.h

6.50 Arc::ClassLoaderPluginArgument Class Reference

Inheritance diagram for Arc::ClassLoaderPluginArgument:



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

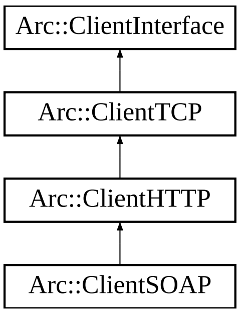
- ClassLoader.h

6.51 Arc::ClientHTTP Class Reference

Class for setting up a **MCC** (p. 252) chain for HTTP communication.

`#include <ClientInterface.h>`

Inheritance diagram for Arc::ClientHTTP:



6.51.1 Detailed Description

Class for setting up a **MCC** (p. 252) chain for HTTP communication. The **ClientHTTP** (p. 78) class inherits from the **ClientTCP** (p. 82) class and adds an HTTP **MCC** (p. 252) 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. 252) 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:

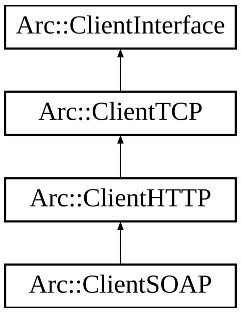
- ClientSAML2SSO.h

6.53 Arc::ClientInterface Class Reference

Utility base class for **MCC** (p. 252).

`#include <ClientInterface.h>`

Inheritance diagram for Arc::ClientInterface:



6.53.1 Detailed Description

Utility base class for **MCC** (p. 252). The **ClientInterface** (p. 79) class is a utility base class used for configuring a client side **Message** (p. 262) Chain Component (**MCC** (p. 252)) chain and loading it into memory. It has several specializations of increasing complexity of the **MCC** (p. 252) chains.

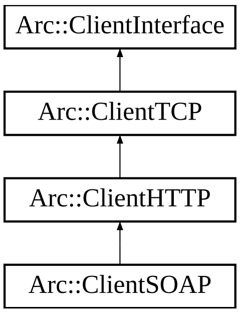
The documentation for this class was generated from the following file:

- 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 bool **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. 252) chain and making an entry point.

6.54.2 Constructor & Destructor Documentation

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

Constructor creates **MCC** (p. 252) 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. 252)

Reimplemented from **Arc::ClientHTTP** (p. 78).

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

Returns entry point to SOAP **MCC** (p. 252) in configured chain. To initialize entry point **Load()** (p. 81) method must be called.

Reimplemented from **Arc::ClientHTTP** (p. 78).

6.54.3.3 virtual bool Arc::ClientSOAP::Load ( ) [virtual]

Instantiates pluggable elements according to generated configuration

Reimplemented from **Arc::ClientHTTP** (p. 78).

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

Send SOAP request and receive response.

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

Send SOAP request with specified SOAP action and receive response.

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

- ClientInterface.h

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. 252) chain and connects to server.

6.55.2 Member Function Documentation

6.55.2.1 **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.

6.55.2.2 **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.

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

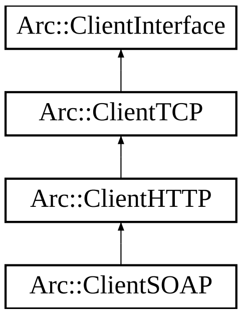
- ClientSAML2SSO.h

6.56 Arc::ClientTCP Class Reference

Class for setting up a **MCC** (p. 252) chain for TCP communication.

`#include <ClientInterface.h>`

Inheritance diagram for Arc::ClientTCP:



6.56.1 Detailed Description

Class for setting up a **MCC** (p. 252) chain for TCP communication. The **ClientTCP** (p. 82) class is a specialization of the **ClientInterface** (p. 79) which sets up a client **MCC** (p. 252) 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. 252) 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:

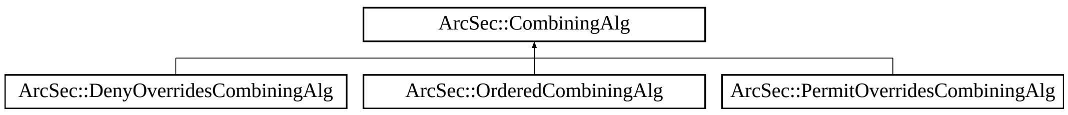
- ClientX509Delegation.h

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. 166), and **ArcSec::PermitOverridesCombiningAlg** (p. 302).

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. 166), and **ArcSec::PermitOverridesCombiningAlg** (p. 303).

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

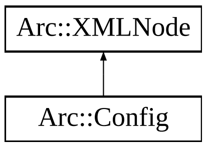
- CombiningAlg.h

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. 465) 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. 100) 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. 87) for DN dn and given keysize Basically Confusa cert handler wraps around **Credential** (p. 100)

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:

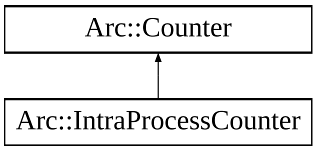
- Utils.h

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. 98) 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. 91) is an abstract class, it should only be used by subclasses. Therefore it is protected. Furthermore, since the **Counter** (p. 91) 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. 91) class. Since the **Counter** (p. 91) 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. 98) that corresponds to the reservation.

Parameters

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

Implemented in **Arc::IntraProcessCounter** (p. 211).

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. 211).

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. 211).

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. 98) 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. 212).

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. 98) class.

This method acts as a relay for one of the constructors of the **CounterTicket** (p. 98) class. That constructor is private, but needs to be accessible from the subclasses of **Counter** (p. 91) (bot not from anywhere else). In order not to have to declare every possible subclass of **Counter** (p. 91) as a friend of **CounterTicket** (p. 98), only the base class **Counter** (p. 91) 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. 98).
expiryTime	the expiry time of the reservation corresponding to the **CounterTicket** (p. 98).
counter	The **Counter** (p. 91) 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. 212).

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

A "relay method" for the constructor of **ExpirationReminder** (p. 181).

This method acts as a relay for one of the constructors of the **ExpirationReminder** (p. 181) class. That constructor is private, but needs to be accessible from the subclasses of **Counter** (p. 91) (bot not from anywhere else). In order not to have to declare every possible subclass of **Counter** (p. 91) as a friend of **ExpirationReminder** (p. 181), only the base class **Counter** (p. 91) 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. 181).
resID	The identity number of the reservation corresponding to the **ExpirationReminder** (p. 181).

Returns

The **ExpirationReminder** (p. 181) 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. 212).

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. 213).

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

Implemented in **Arc::IntraProcessCounter** (p. 213).

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. 213).

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. 214).

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. 91), 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. 98) 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. 91).

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. 98).

This method checks whether a **CounterTicket** (p. 98) is valid. The ticket was probably returned earlier by the reserve() method of a **Counter** (p. 91) but the corresponding reservation may have expired.

Returns

The validity of the ticket.

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

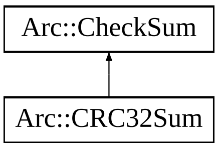
- Counter.h

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 &ext-sect, const std::string &passphrase4key="")
- **Credential** (**Time** start, **Period** lifetime=**Period**("PT12H"), int keybits=1024, std::string proxyversion="rfc", std::string policylang="inheritAll", std::string pol-icy="", int pathlength=-1)
- **Credential** (const std::string &cert, const std::string &key, const std::string &cadir, const std::string &cafile, const std::string &passphrase4key="", const bool is_-file=true)
- **Credential** (const **UserConfig** &usercfg, const std::string &passphrase4key="")
- void **AddCertExtObj** (std::string &sn, std::string &oid)
- void **LogError** (void) const
- bool **GetVerification** (void) const
- EVP_PKEY ∗ **GetPrivKey** (void) const
- EVP_PKEY ∗ **GetPubKey** (void) const
- X509 ∗ **GetCert** (void) const
- X509_REQ ∗ **GetCertReq** (void) const
- **STACK_OF** (X509)∗GetCertChain(void) const
- int **GetCertNumofChain** (void) const
- Credformat **getFormat** (BIO ∗in, const bool is_file=true) const
- std::string **GetDN** (void) const
- std::string **GetIdentityName** (void) const
- **ArcCredential::certType** **GetType** (void) const
- std::string **GetIssuerName** (void) const
- std::string **GetProxyPolicy** (void) const

- void **SetProxyPolicy** (const std::string &proxyversion, const std::string &policylang, const std::string &policy, int pathlength)
- bool **OutputPrivatekey** (std::string &content, bool encryption=false, const std::string &passphrase="")
- bool **OutputPublickey** (std::string &content)
- bool **OutputCertificate** (std::string &content, bool is_der=false)
- bool **OutputCertificateChain** (std::string &content, bool is_der=false)
- **Period GetLifeTime** (void) const
- **Time GetStartTime** () const
- **Time GetEndTime** () const
- void **SetLifeTime** (const **Period** &period)
- void **SetStartTime** (const **Time** &start_time)
- bool **IsValid** (void)
- bool **AddExtension** (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)
- static bool **IsCredentialsValid** (const **UserConfig** &usercfg)

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
life-time,lifetime	of proxy certificate
key-bits,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

passphrase4k	the password for descrypting private key (if needed). If value is empty then password will be asked interrctively. To avoid askig for password use value provided by NoPassword() method.
is_-file,specifies	if the cert/key are from file, otherwise they are supposed to be from string. default is from file

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

Constructor, specific constructor for usual certificate, constructing from information in **UserConfig** (p. 399) object. Only acts as a container for parsing the certificate and key files, is meaningless for any other use. this constructor will parse the credential ∗ information, and put them into "this" object

Parameters

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

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*, 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

6.66.2.3 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)

6.66.2.4 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

6.66.2.5 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 (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

6.66.2.7 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

6.66.2.8 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 (const char ∗ *filename*, bool *if_der* = `false`)

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

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

Get the certificate attached to this object

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

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

6.66.2.12 X509_REQ* Arc::Credential::GetCertReq (void) const

Get the certificate request, if there is any

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

Get the DN of the certificate attached to this object

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

Returns validity end time of certificate or proxy

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

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

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

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

6.66.2.17 std::string Arc::Credential::GetIssuerName (void) const

Get issuer of the certificate attached to this object

6.66.2.18 Period Arc::Credential::GetLifeTime (void) const

Returns lifetime of certificate or proxy

6.66.2.19 EVP_PKEY* Arc::Credential::GetPrivKey (void) const

Get the private key attached to this object

6.66.2.20 std::string Arc::Credential::GetProxyPolicy (void) const

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

6.66.2.21 EVP_PKEY∗ Arc::Credential::GetPubKey (void) const

Get the public key attached to this object

6.66.2.22 Time Arc::Credential::GetStartTime ( ) const

Returns validity start time of certificate or proxy

6.66.2.23 ArcCredential::certType Arc::Credential::GetType (void) const

Get type of the certificate attached to this object

6.66.2.24 bool Arc::Credential::GetVerification (void) const [inline]

Get the verification result about certificate chain checking

6.66.2.25 static void Arc::Credential::InitProxyCertInfo (void) [static]

Initiate nid for proxy certificate extension

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

Inquire the certificate request from a file

6.66.2.29 static bool Arc::Credential::IsCredentialsValid (const UserConfig & *usercfg*) `[static]`

Returns true if credentials are valid. Credentials are read from locations specified in **UserConfig** (p. 399) object. This method is deprecated. **User** (p. 398) per-instance method **IsValid()** (p. 107) instead.

6.66.2.30 bool Arc::Credential::IsValid (void)

Returns true if credentials are valid

6.66.2.31 void Arc::Credential::LogError (void) const

Log error information related with openssl

6.66.2.32 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

6.66.2.33 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

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

Output the private key into string

Parameters

encryp-tion,whether	encrypt the output private key or not
passphrase,th	passphrase to encrypt the output private key

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

Output the public key into string

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

Set lifetime of certificate or proxy

6.66.2.37 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

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

Set start time of certificate or proxy

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

Sign request and output the signed certificate to a file

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

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

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

Sign request and output the signed certificate to a string

6.66.2.42 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

6.66.2.43 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

6.66.2.44 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

6.66.2.45 Arc::Credential::STACK_OF (X509) const

Get the certificate chain attached to this object

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

- Credential.h

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. 100) 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. 110) class.

Parameters

what	An explanation of the error.

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

- Credential.h

6.68 Arc::CredentialStore Class Reference

```
#include <CredentialStore.h>
```

6.68.1 Detailed Description

This class provides functionality for storing delegated crdentials and retrieving them from some store services. This is very preliminary implementation and currently support only one type of credentials - X.509 proxies, and only one type of store service - MyProxy. Later it will be extended to support at least following services: ARC delegation service, VOMS service, local file system.

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

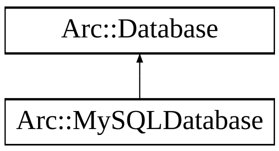
- CredentialStore.h

6.69 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

6.69.1 Detailed Description

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

6.69.2 Constructor & Destructor Documentation

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

Default constructor

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

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

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

Copy constructor

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

Deconstructor

6.69.3 Member Function Documentation

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

Close the connection with database server

Implemented in **Arc::MySQLDatabase** (p. 275).

6.69.3.2 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. 275).

6.69.3.3 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. 276).

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

Get the connection status

Implemented in **Arc::MySQLDatabase** (p. 276).

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

Ask database server to shutdown

Implemented in **Arc::MySQLDatabase** (p. 276).

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

- DBInterface.h

6.70 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**

6.70.1 Detailed Description

Represents set of buffers. This class is used used during data transfer using **DataPoint** (p. 122) classes.

6.70.2 Constructor & Destructor Documentation

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

Contructor

Parameters

size	size of every buffer in bytes.
blocks	number of buffers.

6.70.2.2 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. 113) itself.

6.70.3 Member Function Documentation

6.70.3.1 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. 113) itself.

Returns

integer position in the list of checksum objects.

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

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

6.70.3.3 const CheckSum∗ Arc::DataBuffer::checksum_object ( ) const

Returns **CheckSum** (p. 74) object specified in constructor, returns NULL if index is not in list.

Parameters

index	of the checksum in question.

6.70.3.4 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.

6.70.3.5 bool Arc::DataBuffer::eof_read ( )

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

6.70.3.6 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.

6.70.3.7 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.

6.70.3.8 bool Arc::DataBuffer::eof_write ( )

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

6.70.3.9 bool Arc::DataBuffer::error ( )

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

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

Informs object if error accured on 'read' side.

Parameters

v	true if error.

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

Informs object if error accured on 'write' side.

Parameters

v	true if error.

6.70.3.12 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

6.70.3.13 bool Arc::DataBuffer::for_read ( )

Check if there are buffers which can be taken by **for_read()** (p. 117). This function checks only for buffers and does not take eof and error conditions into account.

6.70.3.14 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.

6.70.3.15 bool Arc::DataBuffer::for_write ( )

Check if there are buffers which can be taken by **for_write()** (p. 117). This function checks only for buffers and does not take eof and error conditions into account.

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

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

Parameters

handle	buffer's number.

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

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

Parameters

buf	- address of buffer

6.70.3.18 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.

6.70.3.19 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.

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

Informs object that data was written from buffer.

Parameters

handle	buffer's number.

6.70.3.21 bool Arc::DataBuffer::is_written (char ∗ *buf*)

Informs object that data was written from buffer.

Parameters

buf	- address of buffer

6.70.3.22 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. 113) itself.

6.70.3.23 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

6.71 Arc::DataCallback Class Reference

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

6.71.1 Detailed Description

This class is used by **DataHandle** (p. 119) 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

6.72 Arc::DataHandle Class Reference

This class is a wrapper around the **DataPoint** (p. 122) class.

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

6.72.1 Detailed Description

This class is a wrapper around the **DataPoint** (p. 122) class. It simplifies the construction, use and destruction of **DataPoint** (p. 122) objects.

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

- DataHandle.h

6.73 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)

6.73.1 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. 121). Instance represents only attributes used during transfer.

6.73.2 Member Function Documentation

6.73.2.1 bool Arc::DataMover::checks ( )

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

6.73.2.2 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).

6.73.2.3 void Arc::DataMover::force_to_meta (bool)

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

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

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

6.73.2.5 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. 149) class.

6.73.2.6 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. 149) class.

6.73.2.7 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. 149) class.

6.73.2.8 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. 387).
destination	destination **URL** (p. 387).
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. 387) mapping/convertion table (for 'source' **URL** (p. 387)).
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.

6.73.2.9 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.

6.73.2.10 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. 121) is NULL.

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

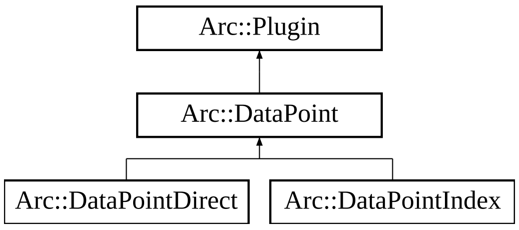
- DataMover.h

6.74 Arc::DataPoint Class Reference

This base class is an abstraction of **URL** (p. 387).

```
#include <DataPoint.h>
```

Inheritance diagram for Arc::DataPoint:



Public Types

- enum **DataPointAccessLatency** { **ACCESS_LATENCY_ZERO**, **ACCESS_-LATENCY_SMALL**, **ACCESS_LATENCY_LARGE** }
- enum **DataPointInfoType** { ,

  INFO_TYPE_NAME = 1, **INFO_TYPE_TYPE** = 2, **INFO_TYPE_TIMES** = 4, **INFO_TYPE_CONTENT** = 8,

  INFO_TYPE_ACCESS = 16, **INFO_TYPE_STRUCT** = 32, **INFO_TYPE_-REST** = 64, **INFO_TYPE_ALL** = 127 }

Public Member Functions

- **DataPoint** (const **URL** &url, const **UserConfig** &usercfg)
- virtual ∼**DataPoint** ()
- virtual const **URL** & **GetURL** () const
- virtual const **UserConfig** & **GetUserConfig** () const
- virtual bool **SetURL** (const **URL** &url)
- virtual std::string **str** () const

- virtual **operator bool** () const
- virtual bool **operator!** () const
- virtual **DataStatus PrepareReading** (unsigned int timeout, unsigned int &wait_-time)
- virtual **DataStatus PrepareWriting** (unsigned int timeout, unsigned int &wait_-time)
- 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 FinishReading** (bool error=false)
- virtual **DataStatus FinishWriting** (bool error=false)
- virtual **DataStatus Check** ()=0
- virtual **DataStatus Remove** ()=0
- virtual **DataStatus Stat** (**FileInfo** &file, **DataPointInfoType** verb=INFO_TYPE_-ALL)=0
- virtual **DataStatus List** (std::list< **FileInfo** > &files, **DataPointInfoType** verb=INFO_-TYPE_ALL)=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 const std::string **DefaultCheckSum** () 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 void **SetAccessLatency** (const **DataPointAccessLatency** &latency)
- virtual **DataPointAccessLatency** **GetAccessLatency** () 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 void **NextTry** (void)
- virtual bool **IsIndex** () const =0
- virtual bool **IsStageable** () const
- virtual bool **AcceptsMeta** ()=0
- virtual bool **ProvidesMeta** ()=0
- virtual void **SetMeta** (const **DataPoint** &p)
- virtual bool **CompareMeta** (const **DataPoint** &p) const
- virtual std::vector< **URL** > **TransferLocations** () const
- virtual const **URL** & **CurrentLocation** () const =0
- virtual const std::string & **CurrentLocationMetadata** () const =0
- virtual **DataStatus** **CompareLocationMetadata** () const =0
- virtual bool **NextLocation** ()=0
- virtual bool **LocationValid** () const =0
- virtual bool **LastLocation** ()=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
- virtual int **AddCheckSumObject** (**CheckSum** ∗cksum)=0
- virtual void **SortLocations** (const std::string &pattern, const **URLMap** &url_-map)=0

Protected Attributes

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

6.74.1 Detailed Description

This base class is an abstraction of **URL** (p. 387). 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. 122) provides means to resolve an indexing service **URL** (p. 387) into multiple URLs and to loop through them.

6.74.2 Member Enumeration Documentation

6.74.2.1 enum Arc::DataPoint::DataPointAccessLatency

Describes the latency to access this **URL** (p. 387).

For now this value is one of a small set specified by the enumeration. In the future with more sophisticated protocols or information it could be replaced by a more fine-grained list of possibilities such as an int value.

Enumerator:

ACCESS_LATENCY_ZERO **URL** (p. 387) can be accessed instantly.

ACCESS_LATENCY_SMALL **URL** (p. 387) has low (but non-zero) access latency, for example staged from disk.

ACCESS_LATENCY_LARGE **URL** (p. 387) has a large access latency, for example staged from tape.

6.74.2.2 enum Arc::DataPoint::DataPointInfoType

Describes type of information about **URL** (p. 387) to request.

Enumerator:

INFO_TYPE_NAME Whatever protocol can get with no additional effort.

INFO_TYPE_TYPE Only name of object (relative).

INFO_TYPE_TIMES Type of object - currently file or dir.

INFO_TYPE_CONTENT Timestamps associated with object.

INFO_TYPE_ACCESS Metadata describing content, like size, checksum, etc.

INFO_TYPE_STRUCT Access control - ownership, permission, etc.

INFO_TYPE_REST Fine structure - replicas, transfer locations, redirections.

INFO_TYPE_ALL All the other parameters.

6.74.3 Constructor & Destructor Documentation

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

Constructor requires **URL** (p. 387) to be provided.

Reference to usercfg argument is stored internally and hence corresponding objects must stay available during whole lifetime of this instance. TODO: do we really need it?

6.74.4 Member Function Documentation

6.74.4.1 virtual int Arc::DataPoint::AddCheckSumObject (CheckSum * *cksum*) `[pure virtual]`

Add a checksum object which will compute checksum during transmission.

Parameters

cksum	object which will compute checksum. Should not be destroyed till Data-Pointer itself.

Returns

integer position in the list of checksum objects.

Implemented in **Arc::DataPointDirect** (p. 136), and **Arc::DataPointIndex** (p. 142).

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

Add **URL** (p. 387) to list.

Parameters

url	Location **URL** (p. 387) to add.
meta	Location meta information.

Implemented in **Arc::DataPointDirect** (p. 137), and **Arc::DataPointIndex** (p. 142).

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

Query (p. 316) the **DataPoint** (p. 122) to check if object is accessible.

If possible this method will also try to provide meta information about the object. It returns positive response if object's content can be retrieved.

Implemented in **Arc::DataPointIndex** (p. 143).

6.74.4.4 virtual DataStatus Arc::DataPoint::CompareLocationMetadata ( ) const `[pure virtual]`

Compare metadata of **DataPoint** (p. 122) and current location.

Returns inconsistency error or error encountered during operation, or success

Implemented in **Arc::DataPointDirect** (p. 137), and **Arc::DataPointIndex** (p. 143).

6.74.4.5 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.

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

Returns meta information used to create current **URL** (p. 387).

Usage differs between different indexing services.

Implemented in **Arc::DataPointDirect** (p. 137), and **Arc::DataPointIndex** (p. 143).

6.74.4.7 virtual DataStatus Arc::DataPoint::FinishReading (bool *error* = false) [virtual]

Finish reading from the **URL** (p. 387).

Must be called after transfer of physical file has completed and if **PrepareReading()** (p. 129) was called, to free resources, release requests that were made during preparation etc.

Parameters

error	If true then action is taken depending on the error.

Reimplemented in **Arc::DataPointIndex** (p. 143).

6.74.4.8 virtual DataStatus Arc::DataPoint::FinishWriting (bool *error* = false) [virtual]

Finish writing to the **URL** (p. 387).

Must be called after transfer of physical file has completed and if **PrepareWriting()** (p. 129) was called, to free resources, release requests that were made during preparation etc.

Parameters

error	If true then action is taken depending on the error.

Reimplemented in **Arc::DataPointIndex** (p. 144).

6.74.4.9 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.

6.74.4.10 virtual **DataStatus** Arc::DataPoint::List (std::list< **FileInfo** > & *files*, **DataPointInfoType** *verb* = `INFO_TYPE_ALL`) `[pure virtual]`

List hierarchical content of this object.

If the **DataPoint** (p. 122) represents a directory or something similar its contents will be listed.

Parameters

files	will contain list of file names and requested attributes. There may be more attributes than requested. There may be less if object can't provide particular information.
verb	defines attribute types which method must try to retrieve. It is not a failure if some attributes could not be retrieved due to limitation of protocol or access control.

6.74.4.11 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. 137), and **Arc::DataPointIndex** (p. 144).

6.74.4.12 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. 137), and **Arc::DataPointIndex** (p. 144).

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

Index **Service** (p. 341) 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. 341) under same name.

Implemented in **Arc::DataPointDirect** (p. 138).

6.74.4.14 virtual DataStatus Arc::DataPoint::PrepareReading (unsigned int *timeout*, unsigned int & *wait_time*) [virtual]

Prepare **DataPoint** (p. 122) for reading.

This method should be implemented by protocols which require preparation or staging of physical files for reading. It can act synchronously or asynchronously (if protocol supports it). In the first case the method will block until the file is prepared or the specified timeout has passed. In the second case the method can return with a ReadPrepareWait status before the file is prepared. The caller should then wait some time (a hint from the remote service may be given in wait_time) and call **PrepareReading()** (p. 129) again to poll for the preparation status, until the file is prepared. In this case it is also up to the caller to decide when the request has taken too long and if so cancel it by calling **FinishReading()** (p. 127). When file preparation has finished, the physical file(s) to read from can be found from **TransferLocations()** (p. 134).

Parameters

timeout	If non-zero, this method will block until either the file has been prepared successfully or the timeout has passed. A zero value means that the caller would like to call and poll for status.
wait_time	If timeout is zero (caller would like asynchronous operation) and ReadPrepareWait is returned, a hint for how long to wait before a subsequent call may be given in wait_time.

Reimplemented in **Arc::DataPointIndex** (p. 144).

6.74.4.15 virtual DataStatus Arc::DataPoint::PrepareWriting (unsigned int *timeout*, unsigned int & *wait_time*) [virtual]

Prepare **DataPoint** (p. 122) for writing.

This method should be implemented by protocols which require preparation of physical files for writing. It can act synchronously or asynchronously (if protocol supports it). In the first case the method will block until the file is prepared or the specified timeout has passed. In the second case the method can return with a WritePrepareWait status before the file is prepared. The caller should then wait some time (a hint from the remote service may be given in wait_time) and call **PrepareWriting()** (p. 129) again to poll for the preparation status, until the file is prepared. In this case it is also up to the caller to decide when the request has taken too long and if so cancel or abort it by calling FinishWriting(true). When file preparation has finished, the physical file(s) to write to can be found from **TransferLocations()** (p. 134).

Parameters

timeout	If non-zero, this method will block until either the file has been prepared successfully or the timeout has passed. A zero value means that the caller would like to call and poll for status.
wait_time	If timeout is zero (caller would like asynchronous operation) and WritePrepareWait is returned, a hint for how long to wait before a subsequent call may be given in wait_time.

Reimplemented in **Arc::DataPointIndex** (p. 145).

6.74.4.16 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. 341).

Implemented in **Arc::DataPointDirect** (p. 138).

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

Index **Service** (p. 341) 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. 341) under same name.

Implemented in **Arc::DataPointDirect** (p. 138).

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

If endpoint can provide at least some meta information directly.

Implemented in **Arc::DataPointDirect** (p. 138), and **Arc::DataPointIndex** (p. 145).

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

Set range of bytes to retrieve.

Default values correspond to whole file.

Implemented in **Arc::DataPointDirect** (p. 139), and **Arc::DataPointIndex** (p. 145).

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

Allow/disallow **DataPoint** (p. 122) to produce scattered data during reading∗ operation.

Parameters

v	true if allowed (default is false).

Implemented in **Arc::DataPointDirect** (p. 139), and **Arc::DataPointIndex** (p. 146).

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

Check if file is registered in Indexing **Service** (p. 341).

Proper value is obtainable only after Resolve.

Implemented in **Arc::DataPointDirect** (p. 139), and **Arc::DataPointIndex** (p. 146).

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

Resolves index service **URL** (p. 387) into list of ordinary URLs.

Also obtains meta information about the file.

Parameters

source	true if **DataPoint** (p. 122) object represents source of information.

Implemented in **Arc::DataPointDirect** (p. 139).

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

Allow/disallow additional checks.

Check for existence 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. 139), and **Arc::DataPointIndex** (p. 146).

6.74.4.24 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.

Reimplemented in **Arc::DataPointIndex** (p. 146).

6.74.4.25 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. 140), and **Arc::DataPointIndex** (p. 146).

6.74.4.26 virtual bool Arc::DataPoint::SetURL (const URL & *url*) [virtual]

Assigns new **URL** (p. 387). Main purpose of this method is to reuse existing connection for accessing different object at same server. Implementation does not have to implement this method. If supplied **URL** (p. 387) is not suitable or method is not implemented false is returned.

6.74.4.27 virtual void Arc::DataPoint::SortLocations (const std::string & *pattern*, const URLMap & *url_map*) [pure virtual]

Sort locations according to the specified pattern.

Parameters

pattern	a set of strings, separated by |, to match against.

Implemented in **Arc::DataPointDirect** (p. 140), and **Arc::DataPointIndex** (p. 147).

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

Start reading data from **URL** (p. 387).

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 **StopReading()** (p. 134) was called and returned.

Implemented in **Arc::DataPointIndex** (p. 147).

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

Start writing data to **URL** (p. 387).

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. 147).

6.74.4.30 virtual DataStatus Arc::DataPoint::Stat (FileInfo & *file*, DataPointInfoType *verb* = INFO_TYPE_ALL) [pure virtual]

Retrieve information about this object.

If the **DataPoint** (p. 122) represents a directory or something similar, information about the object itself and not its contents will be obtained.

Parameters

file	will contain object name and requested attributes. There may be more attributes than requested. There may be less if object can't provide particular information.
verb	defines attribute types which method must try to retrieve. It is not a failure if some attributes could not be retrieved due to limitation of protocol or access control.

6.74.4.31 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. 148).

6.74.4.32 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. 148).

6.74.4.33 virtual std::vector<URL> Arc::DataPoint::TransferLocations ( ) const `[virtual]`

Returns physical file(s) to read/write, if different from **CurrentLocation**() (p. 124)

To be used with protocols which re-direct to different URLs such as Transport URLs (TURLs). The list is initially filled by PrepareReading and PrepareWriting. If this list is non-empty then real transfer should use a **URL** (p. 387) from this list. It is up to the caller to choose the best **URL** (p. 387) and instantiate new **DataPoint** (p. 122) for handling it. For consistency protocols which do not require redirections return original **URL** (p. 387). For protocols which need redirection calling StartReading and StartWriting will use first **URL** (p. 387) in the list.

Reimplemented in **Arc::DataPointIndex** (p. 148).

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

Index **Service** (p. 341) unregistration.

Remove information about file registered in Indexing **Service** (p. 341).

Parameters

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

Implemented in **Arc::DataPointDirect** (p. 140).

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

Returns true if **URL** (p. 387) can accept scattered data for ∗writing∗ operation.

Implemented in **Arc::DataPointDirect** (p. 140), and **Arc::DataPointIndex** (p. 148).

6.74.5 Field Documentation

6.74.5.1 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:

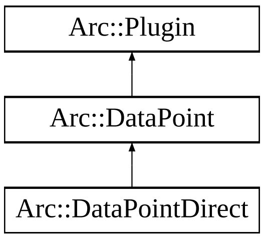
- DataPoint.h

6.75 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 bool **IsStageable** () 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 int **AddCheckSumObject** (**CheckSum** ∗cksum)
- 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 **DataStatus CompareLocationMetadata** () const
- virtual bool **NextLocation** ()
- virtual bool **LocationValid** () const
- virtual bool **HaveLocations** () const
- virtual bool **LastLocation** ()
- virtual **DataStatus AddLocation** (const **URL** &url, const std::string &meta)
- virtual **DataStatus RemoveLocation** ()
- virtual **DataStatus RemoveLocations** (const **DataPoint** &p)
- virtual void **SortLocations** (const std::string &, const **URLMap** &)

6.75.1 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. 113) 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.

6.75.2 Member Function Documentation

6.75.2.1 virtual int Arc::DataPointDirect::AddCheckSumObject (CheckSum ∗ *cksum*) [virtual]

Add a checksum object which will compute checksum during transmission.

Parameters

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

Returns

integer position in the list of checksum objects.

Implements **Arc::DataPoint** (p. 126).

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

Add **URL** (p. 387) to list.

Parameters

url	Location **URL** (p. 387) to add.
meta	Location meta information.

Implements **Arc::DataPoint** (p. 126).

6.75.2.3 virtual DataStatus Arc::DataPointDirect::CompareLocationMetadata ( ) const [virtual]

Compare metadata of **DataPoint** (p. 122) and current location.

Returns inconsistency error or error encountered during operation, or success

Implements **Arc::DataPoint** (p. 127).

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

Returns meta information used to create current **URL** (p. 387).

Usage differs between different indexing services.

Implements **Arc::DataPoint** (p. 127).

6.75.2.5 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. 128).

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

Request passive transfers for FTP-like protocols.

Parameters

true	to request.

Implements **Arc::DataPoint** (p. 128).

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

Index **Service** (p. 341) 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. 341) under same name.

Implements **Arc::DataPoint** (p. 129).

6.75.2.8 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. 341).

Implements **Arc::DataPoint** (p. 130).

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

Index **Service** (p. 341) 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. 341) under same name.

Implements **Arc::DataPoint** (p. 130).

6.75.2.10 virtual bool Arc::DataPointDirect::ProvidesMeta ( ) `[virtual]`

If endpoint can provide at least some meta information directly.

Implements **Arc::DataPoint** (p. 131).

6.75.2.11 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. 131).

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

Allow/disallow **DataPoint** (p. 122) to produce scattered data during reading∗ operation.

Parameters

v	true if allowed (default is false).

Implements **Arc::DataPoint** (p. 131).

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

Check if file is registered in Indexing **Service** (p. 341).

Proper value is obtainable only after Resolve.

Implements **Arc::DataPoint** (p. 131).

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

Resolves index service **URL** (p. 387) into list of ordinary URLs.

Also obtains meta information about the file.

Parameters

source	true if **DataPoint** (p. 122) object represents source of information.

Implements **Arc::DataPoint** (p. 131).

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

Allow/disallow additional checks.

Check for existence 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. 131).

6.75.2.16 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. 132).

6.75.2.17 virtual void Arc::DataPointDirect::SortLocations (const std::string & *pattern*, const URLMap & *url_map*) `[inline, virtual]`

Sort locations according to the specified pattern.

Parameters

pattern	a set of strings, separated by |, to match against.

Implements **Arc::DataPoint** (p. 132).

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

Index **Service** (p. 341) unregistration.

Remove information about file registered in Indexing **Service** (p. 341).

Parameters

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

Implements **Arc::DataPoint** (p. 134).

6.75.2.19 virtual bool Arc::DataPointDirect::WriteOutOfOrder ( ) `[virtual]`

Returns true if **URL** (p. 387) can accept scattered data for ∗writing∗ operation.

Implements **Arc::DataPoint** (p. 134).

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

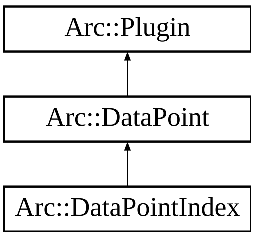
- DataPointDirect.h

6.76 Arc::DataPointIndex Class Reference

Complements **DataPoint** (p. 122) with attributes common for Indexing **Service** (p. 341) URLs.

`#include <DataPointIndex.h>`

Inheritance diagram for Arc::DataPointIndex:



Public Member Functions

- virtual const **URL** & **CurrentLocation** () const
- virtual const std::string & **CurrentLocationMetadata** () const
- virtual **DataStatus CompareLocationMetadata** () const
- virtual bool **NextLocation** ()
- virtual bool **LocationValid** () const
- virtual bool **HaveLocations** () const
- virtual bool **LastLocation** ()
- virtual **DataStatus RemoveLocation** ()
- virtual **DataStatus RemoveLocations** (const **DataPoint** &p)
- virtual **DataStatus AddLocation** (const **URL** &url, const std::string &meta)
- virtual void **SortLocations** (const std::string &pattern, const **URLMap** &url_-map)
- virtual bool **IsIndex** () const
- virtual bool **IsStageable** () const
- virtual bool **AcceptsMeta** ()
- virtual bool **ProvidesMeta** ()
- virtual void **SetMeta** (const **DataPoint** &p)
- virtual void **SetCheckSum** (const std::string &val)
- virtual void **SetSize** (const unsigned long long int val)
- 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 PrepareReading** (unsigned int timeout, unsigned int &wait_-time)
- virtual **DataStatus PrepareWriting** (unsigned int timeout, unsigned int &wait_-time)
- virtual **DataStatus StartReading** (**DataBuffer** &buffer)
- virtual **DataStatus StartWriting** (**DataBuffer** &buffer, **DataCallback** ∗space_-cb=NULL)
- virtual **DataStatus StopReading** ()
- virtual **DataStatus StopWriting** ()
- virtual **DataStatus FinishReading** (bool error=false)
- virtual **DataStatus FinishWriting** (bool error=false)
- virtual std::vector< **URL** > **TransferLocations** () const

- 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 **DataPointAccessLatency GetAccessLatency** () const
- virtual void **Passive** (bool v)
- virtual void **Range** (unsigned long long int start=0, unsigned long long int end=0)
- virtual int **AddCheckSumObject** (**CheckSum** ∗cksum)

6.76.1 Detailed Description

Complements **DataPoint** (p. 122) with attributes common for Indexing **Service** (p. 341) URLs. It should never be used directly. Instead inherit from it to provide a class for specific a Indexing **Service** (p. 341).

6.76.2 Member Function Documentation

6.76.2.1 virtual int Arc::DataPointIndex::AddCheckSumObject (CheckSum ∗ *cksum*) [virtual]

Add a checksum object which will compute checksum during transmission.

Parameters

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

Returns

integer position in the list of checksum objects.

Implements **Arc::DataPoint** (p. 126).

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

Add **URL** (p. 387) to list.

Parameters

url	Location **URL** (p. 387) to add.
meta	Location meta information.

Implements **Arc::DataPoint** (p. 126).

6.76.2.3 virtual DataStatus Arc::DataPointIndex::Check ( ) `[virtual]`

Query (p. 316) the **DataPoint** (p. 122) to check if object is accessible.

If possible this method will also try to provide meta information about the object. It returns positive response if object's content can be retrieved.

Implements **Arc::DataPoint** (p. 126).

6.76.2.4 virtual DataStatus Arc::DataPointIndex::CompareLocationMetadata ( ) const `[virtual]`

Compare metadata of **DataPoint** (p. 122) and current location.

Returns inconsistency error or error encountered during operation, or success

Implements **Arc::DataPoint** (p. 127).

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

Returns meta information used to create current **URL** (p. 387).

Usage differs between different indexing services.

Implements **Arc::DataPoint** (p. 127).

6.76.2.6 virtual DataStatus Arc::DataPointIndex::FinishReading (bool *error* = `false`) `[virtual]`

Finish reading from the **URL** (p. 387).

Must be called after transfer of physical file has completed and if **PrepareReading()** (p. 144) was called, to free resources, release requests that were made during preparation etc.

Parameters

error	If true then action is taken depending on the error.

Reimplemented from **Arc::DataPoint** (p. 127).

6.76.2.7 virtual DataStatus Arc::DataPointIndex::FinishWriting (bool *error* = `false`) `[virtual]`

Finish writing to the **URL** (p. 387).

Must be called after transfer of physical file has completed and if **PrepareWriting()** (p. 145) was called, to free resources, release requests that were made during preparation etc.

Parameters

error	If true then action is taken depending on the error.

Reimplemented from **Arc::DataPoint** (p. 127).

6.76.2.8 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. 128).

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

Request passive transfers for FTP-like protocols.

Parameters

true	to request.

Implements **Arc::DataPoint** (p. 128).

6.76.2.10 virtual DataStatus Arc::DataPointIndex::PrepareReading (unsigned int *timeout*, unsigned int & *wait_time*) `[virtual]`

Prepare **DataPoint** (p. 122) for reading.

This method should be implemented by protocols which require preparation or staging of physical files for reading. It can act synchronously or asynchronously (if protocol supports it). In the first case the method will block until the file is prepared or the specified timeout has passed. In the second case the method can return with a Read-PrepareWait status before the file is prepared. The caller should then wait some time (a hint from the remote service may be given in wait_time) and call **PrepareReading()** (p. 144) again to poll for the preparation status, until the file is prepared. In this case it is also up to the caller to decide when the request has taken too long and if so cancel it by calling **FinishReading()** (p. 143). When file preparation has finished, the physical file(s) to read from can be found from **TransferLocations()** (p. 148).

Parameters

timeout	If non-zero, this method will block until either the file has been prepared successfully or the timeout has passed. A zero value means that the caller would like to call and poll for status.
wait_time	If timeout is zero (caller would like asynchronous operation) and ReadPre-pareWait is returned, a hint for how long to wait before a subsequent call may be given in wait_time.

Reimplemented from **Arc::DataPoint** (p. 129).

6.76.2.11 virtual DataStatus Arc::DataPointIndex::PrepareWriting (unsigned int *timeout*, unsigned int & *wait_time*) [virtual]

Prepare **DataPoint** (p. 122) for writing.

This method should be implemented by protocols which require preparation of physical files for writing. It can act synchronously or asynchronously (if protocol supports it). In the first case the method will block until the file is prepared or the specified timeout has passed. In the second case the method can return with a WritePrepareWait status before the file is prepared. The caller should then wait some time (a hint from the remote service may be given in wait_time) and call **PrepareWriting()** (p. 145) again to poll for the preparation status, until the file is prepared. In this case it is also up to the caller to decide when the request has taken too long and if so cancel or abort it by calling FinishWriting(true). When file preparation has finished, the physical file(s) to write to can be found from **TransferLocations()** (p. 148).

Parameters

timeout	If non-zero, this method will block until either the file has been prepared successfully or the timeout has passed. A zero value means that the caller would like to call and poll for status.
wait_time	If timeout is zero (caller would like asynchronous operation) and WritePrepareWait is returned, a hint for how long to wait before a subsequent call may be given in wait_time.

Reimplemented from **Arc::DataPoint** (p. 129).

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

If endpoint can provide at least some meta information directly.

Implements **Arc::DataPoint** (p. 131).

6.76.2.13 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. 131).

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

Allow/disallow **DataPoint** (p. 122) to produce scattered data during reading$*$ operation.

Parameters

v	true if allowed (default is false).

Implements **Arc::DataPoint** (p. 131).

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

Check if file is registered in Indexing **Service** (p. 341).

Proper value is obtainable only after Resolve.

Implements **Arc::DataPoint** (p. 131).

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

Allow/disallow additional checks.

Check for existence 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. 131).

6.76.2.17 virtual void Arc::DataPointIndex::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.

Reimplemented from **Arc::DataPoint** (p. 132).

6.76.2.18 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. 132).

6.76.2.19 virtual void Arc::DataPointIndex::SortLocations (const std::string & *pattern,* const URLMap & *url_map*) [virtual]

Sort locations according to the specified pattern.

Parameters

pattern	a set of strings, separated by |, to match against.

Implements **Arc::DataPoint** (p. 132).

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

Start reading data from **URL** (p. 387).

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 **StopReading()** (p. 148) was called and returned.

Implements **Arc::DataPoint** (p. 133).

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

Start writing data to **URL** (p. 387).

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. 133).

6.76.2.22 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. 134).

6.76.2.23 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. 134).

6.76.2.24 virtual std::vector<URL> Arc::DataPointIndex::TransferLocations ( ) const [virtual]

Returns physical file(s) to read/write, if different from **CurrentLocation**() (p. 141)

To be used with protocols which re-direct to different URLs such as Transport URLs (TURLs). The list is initially filled by PrepareReading and PrepareWriting. If this list is non-empty then real transfer should use a **URL** (p. 387) from this list. It is up to the caller to choose the best **URL** (p. 387) and instantiate new **DataPoint** (p. 122) for handling it. For consistency protocols which do not require redirections return original **URL** (p. 387). For protocols which need redirection calling StartReading and StartWriting will use first **URL** (p. 387) in the list.

Reimplemented from **Arc::DataPoint** (p. 134).

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

Returns true if **URL** (p. 387) can accept scattered data for ∗writing∗ operation.

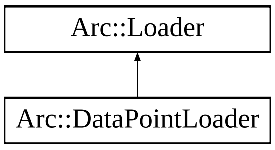
Implements **Arc::DataPoint** (p. 134).

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

- DataPointIndex.h

6.77 Arc::DataPointLoader Class Reference

Inheritance diagram for Arc::DataPointLoader:

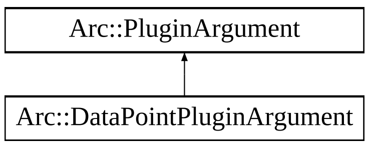


The documentation for this class was generated from the following file:

- DataPoint.h

6.78 Arc::DataPointPluginArgument Class Reference

Inheritance diagram for Arc::DataPointPluginArgument:



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

- DataPoint.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)
- time_t **get_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 **transferred_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 transferred. 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 transferred. 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 transferred (bytes).

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

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

6.79.3.10 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.

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

- DataSpeed.h

6.80 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 , **StatError** = 27 , **NotInitializedError** = 29,

  SystemError = 30, **StageError** = 31 , **InconsistentMetadataError** = 32, **ReadPrepareError** = 32 ,

  ReadPrepareWait = 33, **WritePrepareError** = 34 , **WritePrepareWait** = 35, **ReadFinishError** = 36 ,

  WriteFinishError = 37 , **SuccessCached** = 38, **UnknownError** = 39 }

6.80.1 Detailed Description

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

6.80.2 Member Enumeration Documentation

6.80.2.1 enum Arc::DataStatus::DataStatusType

Enumerator:

Success Operation completed successfully.

ReadAcquireError Source is bad **URL** (p. 387) or can't be used due to some reason.

WriteAcquireError Destination is bad **URL** (p. 387) or can't be used due to some reason.

ReadResolveError Resolving of index service **URL** (p. 387) for source failed.

WriteResolveError Resolving of index service **URL** (p. 387) 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. 387) failed.

PostRegisterError Last stage of registration of index service **URL** (p. 387) failed.

UnregisterError Unregistration of index service **URL** (p. 387) failed.

CacheError Error in caching procedure.

CredentialsExpiredError Error due to provided credentials are expired.

DeleteError Error deleting location or **URL** (p. 387).

NoLocationError No valid location available.

LocationAlreadyExistsError No valid location available.

NotSupportedForDirectDataPointsError Operation has no sense for this kind of **URL** (p. 387).

UnimplementedError Feature is unimplemented.

IsReadingError **DataPoint** (p. 122) is already reading.

IsWritingError **DataPoint** (p. 122) is already writing.

CheckError Access check failed.

ListError File listing failed.

StatError File/dir stating failed.

NotInitializedError Object initialization failed.

SystemError Error in OS.

StageError Staging error.

InconsistentMetadataError Inconsistent metadata.

ReadPrepareError Can't prepare source.

ReadPrepareWait Wait for source to be prepared.

WritePrepareError Can't prepare destination.

WritePrepareWait Wait for destination to be prepared.

ReadFinishError Can't finish source.

WriteFinishError Can't finish destination.

SuccessCached Data was already cached.

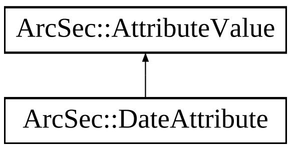
UnknownError Undefined.

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

- DataStatus.h

6.81 ArcSec::DateAttribute Class Reference

Inheritance diagram for ArcSec::DateAttribute:



Public Member Functions

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

6.81.1 Member Function Documentation

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

encode the value in a string format

Implements **ArcSec::AttributeValue** (p. 64).

6.81.1.2 virtual bool ArcSec::DateAttribute::equal (AttributeValue ∗ *value*, bool *check_id* = `true`) `[virtual]`

Evluate whether "this" equale to the parameter value

Implements **ArcSec::AttributeValue** (p. 64).

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

Get the AttributeId of the <Attribute>

Implements **ArcSec::AttributeValue** (p. 64).

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

Get the DataType of the <Attribute>

Implements **ArcSec::AttributeValue** (p. 65).

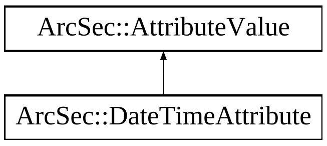
The documentation for this class was generated from the following file:

- DateTimeAttribute.h

6.82 ArcSec::DateTimeAttribute Class Reference

`#include <DateTimeAttribute.h>`

Inheritance diagram for ArcSec::DateTimeAttribute:



Public Member Functions

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

6.82.1 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

6.82.2 Member Function Documentation

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

encode the value in a string format

Implements **ArcSec::AttributeValue** (p. 64).

6.82.2.2 virtual bool ArcSec::DateTimeAttribute::equal (**AttributeValue** ∗ *value*, bool *check_id* = `true`) `[virtual]`

Evluate whether "this" equale to the parameter value

Implements **ArcSec::AttributeValue** (p. 64).

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

Get the AttributeId of the <Attribute>

Implements **ArcSec::AttributeValue** (p. 64).

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

Get the DataType of the <Attribute>

Implements **ArcSec::AttributeValue** (p. 65).

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

- DateTimeAttribute.h

6.83 Arc::DBranch Class Reference

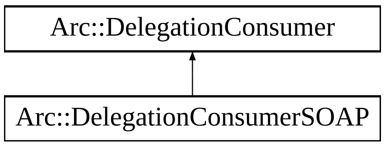
The documentation for this class was generated from the following file:

- DBranch.h

6.84 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)

6.84.1 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. 159) method for generating certificate request followed by call to **Acquire()** (p. 158) method for making complete credentials from certificate chain.

6.84.2 Constructor & Destructor Documentation

6.84.2.1 Arc::DelegationConsumer::DelegationConsumer (void)

Creates object with new private key

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

Creates object with provided private key

6.84.3 Member Function Documentation

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

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

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

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

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

Stores content of this object into a string

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

Private key

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

Return identifier of this object - not implemented

6.84.3.6 void Arc::DelegationConsumer::LogError (void) `[protected]`

Creates private key

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

Make X509 certificate request from internal private key

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

Restores content of object from string

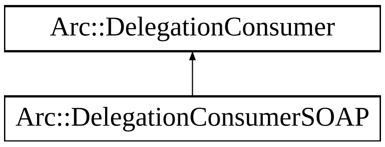
The documentation for this class was generated from the following file:

- DelegationInterface.h

6.85 Arc::DelegationConsumerSOAP Class Reference

`#include <DelegationInterface.h>`

Inheritance diagram for Arc::DelegationConsumerSOAP:



Public Member Functions

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

6.85.1 Detailed Description

This class extends **DelegationConsumer** (p. 157) to support SOAP message exchange. Implements WS interface `http://www.nordugrid.org/schemas/delegation` described in delegation.wsdl.

6.85.2 Constructor & Destructor Documentation

6.85.2.1 Arc::DelegationConsumerSOAP::DelegationConsumerSOAP (void)

Creates object with new private key

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

Creates object with specified private key

6.85.3 Member Function Documentation

6.85.3.1 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.

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

Similar to UpdateCredentials but takes only DelegatedToken XML element

6.85.3.3 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

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

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

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

- DelegationInterface.h

6.86 Arc::DelegationContainerSOAP Class Reference

```
#include <DelegationInterface.h>
```

Public Member Functions

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

Protected Attributes

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

6.86.1 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. 159) with identifier included in SOAP message used to route execution to one of managed **DelegationConsumerSOAP** (p. 159) instances.

6.86.2 Member Function Documentation

6.86.2.1 bool Arc::DelegationContainerSOAP::DelegateCredentialsInit (const SOAPEnvelope & *in*, SOAPEnvelope & *out*, const std::string & *client* = "")

See **DelegationConsumerSOAP::DelegateCredentialsInit** (p. 160) If 'client' is not empty then all subsequent calls involving access to generated credentials must contain same value in their 'client' arguments.

6.86.2.2 bool Arc::DelegationContainerSOAP::DelegatedToken (std::string & *credentials*, XMLNode *token*, const std::string & *client* = "")

See **DelegationConsumerSOAP::DelegatedToken** (p. 160)

6.86.2.3 bool Arc::DelegationContainerSOAP::UpdateCredentials (std::string & *credentials*, const SOAPEnvelope & *in*, SOAPEnvelope & *out*, const std::string & *client* = "")

See **DelegationConsumerSOAP::UpdateCredentials** (p. 160)

6.86.3 Field Documentation

6.86.3.1 bool **Arc::DelegationContainerSOAP::context_lock_** `[protected]`

If true delegation consumer is deleted when connection context is destroyed

6.86.3.2 int **Arc::DelegationContainerSOAP::max_duration_** `[protected]`

Lifetime of unused delegation consumer

6.86.3.3 int **Arc::DelegationContainerSOAP::max_size_** `[protected]`

Max. number of delegation consumers

6.86.3.4 int **Arc::DelegationContainerSOAP::max_usage_** `[protected]`

Max. times same delegation consumer may accept credentials

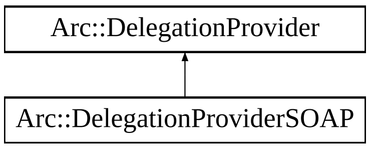
The documentation for this class was generated from the following file:

- DelegationInterface.h

6.87 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())

6.87.1 Detailed Description

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

6.87.2 Constructor & Destructor Documentation

6.87.2.1 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.

6.87.2.2 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.

6.87.3 Member Function Documentation

6.87.3.1 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. 158)

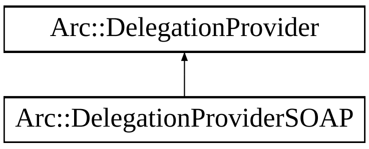
The documentation for this class was generated from the following file:

- DelegationInterface.h

6.88 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)

6.88.1 Detailed Description

Extension of **DelegationProvider** (p. 162) with SOAP exchange interface. This class is also a temporary container for intermediate information used during delegation procedure.

6.88.2 Constructor & Destructor Documentation

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

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

6.88.2.2 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.

6.88.3 Member Function Documentation

6.88.3.1 bool Arc::DelegationProviderSOAP::DelegateCredentialsInit (MCCInterface & *mcc_interface,* MessageContext ∗ *context*)

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

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

Extended version of **DelegateCredentialsInit(MCCInterface&,MessageContext∗)** (p. 164). Additionally takes attributes for request and response message to make fine control on message processing possible.

6.88.3.3 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.

6.88.3.4 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.

6.88.3.5 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.

6.88.3.6 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. 159) instance.

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

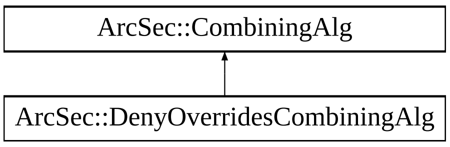
- DelegationInterface.h

6.89 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

6.89.1 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".

6.89.2 Member Function Documentation

6.89.2.1 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. 85).

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

Get the identifier

Implements **ArcSec::CombiningAlg** (p. 85).

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

- DenyOverridesAlg.h

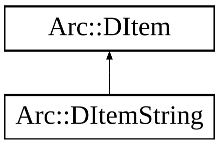
6.90 Arc::DiskSpaceRequirementType Class Reference

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

- JobDescription.h

6.91 Arc::DItem Class Reference

Inheritance diagram for Arc::DItem:

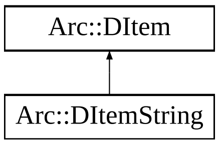


The documentation for this class was generated from the following file:

- DBranch.h

6.92 Arc::DItemString Class Reference

Inheritance diagram for Arc::DItemString:

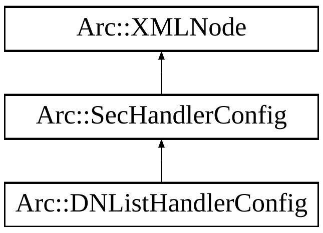


The documentation for this class was generated from the following file:

- DBranch.h

6.93 Arc::DNListHandlerConfig Class Reference

Inheritance diagram for Arc::DNListHandlerConfig:



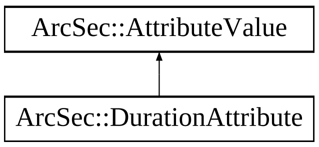
The documentation for this class was generated from the following file:

- ClientInterface.h

6.94 ArcSec::DurationAttribute Class Reference

`#include <DateTimeAttribute.h>`

Inheritance diagram for ArcSec::DurationAttribute:



Public Member Functions

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

6.94.1 Detailed Description

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

6.94.2 Member Function Documentation

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

encode the value in a string format

Implements **ArcSec::AttributeValue** (p. 64).

6.94.2.2 virtual bool ArcSec::DurationAttribute::equal (**AttributeValue** ∗ *value*, bool *check_id* = `true`) `[virtual]`

Evluate whether "this" equale to the parameter value

Implements **ArcSec::AttributeValue** (p. 64).

6.94.2.3 virtual std::string ArcSec::DurationAttribute::getId ( ) `[inline, virtual]`

Get the AttributeId of the <Attribute>

Implements **ArcSec::AttributeValue** (p. 64).

6.94.2.4 virtual std::string ArcSec::DurationAttribute::getType ( ) `[inline, virtual]`

Get the DataType of the <Attribute>

Implements **ArcSec::AttributeValue** (p. 65).

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

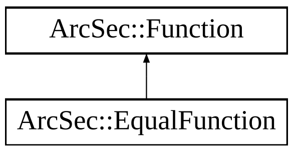
- DateTimeAttribute.h

6.95 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)

6.95.1 Detailed Description

Evaluate whether the two values are equal.

6.95.2 Member Function Documentation

6.95.2.1 virtual **AttributeValue**∗ ArcSec::EqualFunction::evaluate (**AttributeValue** ∗ *arg0*, **AttributeValue** ∗ *arg1*, bool *check_id* = `true`) `[virtual]`

Evaluate two **AttributeValue** (p. 63) objects, and return one **AttributeValue** (p. 63) object

Implements **ArcSec::Function** (p. 195).

6.95.2.2 virtual std::list<**AttributeValue**∗> ArcSec::EqualFunction::evaluate (std::list< **AttributeValue** ∗ > *args*, bool *check_id* = `true`) `[virtual]`

Evaluate a list of **AttributeValue** (p. 63) objects, and return a list of Attribute objects

Implements **ArcSec::Function** (p. 195).

6.95.2.3 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

6.96 ArcSec::EvalResult Struct Reference

Struct to record the xml node and effect, which will be used by **Evaluator** (p. 171) to get the information about which rule/policy(in xmlnode) is satisfied.

`#include <Result.h>`

6.96.1 Detailed Description

Struct to record the xml node and effect, which will be used by **Evaluator** (p. 171) 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

6.97 ArcSec::EvaluationCtx Class Reference

EvaluationCtx (p. 171), in charge of storing some context information for.

`#include <EvaluationCtx.h>`

Public Member Functions

- **EvaluationCtx** (**Request** ∗request)

6.97.1 Detailed Description

EvaluationCtx (p. 171), in charge of storing some context information for.

6.97.2 Constructor & Destructor Documentation

6.97.2.1 ArcSec::EvaluationCtx::EvaluationCtx (**Request** ∗ *request*) `[inline]`

Construct a new **EvaluationCtx** (p. 171) based on the given request

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

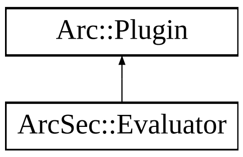
- EvaluationCtx.h

6.98 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

6.98.1 Detailed Description

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

6.98.2 Member Function Documentation

6.98.2.1 virtual void ArcSec::Evaluator::addPolicy (const Source & *policy*, const std::string & *id* = "") [pure virtual]

Add policy from specified source to the evaluator. **Policy** (p. 310) will be marked with id.

6.98.2.2 virtual void ArcSec::Evaluator::addPolicy (Policy ∗ *policy*, const std::string & *id* = "") [pure virtual]

Add policy to the evaluator. **Policy** (p. 310) will be marked with id. The policy object is taken over by this instance and will be destroyed in destructor.

6.98.2.3 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.

6.98.2.4 virtual Response∗ ArcSec::Evaluator::evaluate (Request ∗ *request*) `[pure virtual]`

Evaluates the request by using a **Request** (p. 321) object. Evaluation is done till at least one of policies is satisfied.

6.98.2.5 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.

6.98.2.6 virtual Response∗ ArcSec::Evaluator::evaluate (const Source & *request*) `[pure virtual]`

Evaluates the request by using a specified source

6.98.2.7 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.

6.98.2.8 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.

6.98.2.9 virtual Response∗ ArcSec::Evaluator::evaluate (EvaluationCtx ∗ *ctx*) `[protected, pure virtual]`

Evaluate the request by using the **EvaluationCtx** (p. 171) object (which includes the information about request). The ctx is destroyed inside this method (why?!?!?).

6.98.2.10 virtual AlgFactory∗ ArcSec::Evaluator::getAlgFactory ( ) `[pure virtual]`

Get the **AlgFactory** (p. 54) object

6.98.2.11 virtual AttributeFactory∗ ArcSec::Evaluator::getAttrFactory ( ) `[pure virtual]`

Get the **AttributeFactory** (p. 58) object

6.98.2.12 virtual FnFactory* ArcSec::Evaluator::getFnFactory ( ) [pure virtual]

Get the **FnFactory** (p. 193) object

6.98.2.13 virtual const char* ArcSec::Evaluator::getName (void) const [pure virtual]

Get the name of this evaluator

6.98.2.14 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.

6.98.2.15 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.

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

- Evaluator.h

6.99 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** * ()

6.99.1 Detailed Description

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

6.99.2 Member Function Documentation

6.99.2.1 ArcSec::EvaluatorContext::operator **AlgFactory** ∗ ( ) `[inline]`

Returns associated **AlgFactory** (p. 54) object

6.99.2.2 ArcSec::EvaluatorContext::operator **AttributeFactory** ∗ ( ) `[inline]`

Returns associated **AttributeFactory** (p. 58) object

6.99.2.3 ArcSec::EvaluatorContext::operator **FnFactory** ∗ ( ) `[inline]`

Returns associated **FnFactory** (p. 193) object

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

- Evaluator.h

6.100 ArcSec::EvaluatorLoader Class Reference

EvaluatorLoader (p. 175) is implemented as a helper class for loading different **Evaluator** (p. 171) 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)

6.100.1 Detailed Description

EvaluatorLoader (p. 175) is implemented as a helper class for loading different **Evaluator** (p. 171) objects, like ArcEvaluator. The object loading is based on the configuration information about evaluator, including information for factory class, request, policy and evaluator itself

6.100.2 Member Function Documentation

6.100.2.1 Evaluator∗ ArcSec::EvaluatorLoader::getEvaluator (const std::string & *classname*)

Get evaluator object according to the class name

6.100.2.2 Evaluator∗ ArcSec::EvaluatorLoader::getEvaluator (const Policy ∗ *policy*)

Get evaluator object suitable for presented policy

6.100.2.3 Evaluator∗ ArcSec::EvaluatorLoader::getEvaluator (const Request ∗ *request*)

Get evaluator object suitable for presented request

6.100.2.4 Policy∗ ArcSec::EvaluatorLoader::getPolicy (const Source & *policysource*)

Get proper policy object according to the policy source

6.100.2.5 Policy∗ ArcSec::EvaluatorLoader::getPolicy (const std::string & *classname,* const Source & *policysource*)

Get policy object according to the class name, based on the policy source

6.100.2.6 Request∗ ArcSec::EvaluatorLoader::getRequest (const std::string & *classname,* const Source & *requestsource*)

Get request object according to the class name, based on the request source

6.100.2.7 Request∗ ArcSec::EvaluatorLoader::getRequest (const Source & *requestsource*)

Get request object according to the request source

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

- EvaluatorLoader.h

6.101 Arc::ExecutableType Class Reference

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

- JobDescription.h

6.102 Arc::ExecutionTarget Class Reference

ExecutionTarget (p. 177).

```
#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
- void **SaveToStream** (std::ostream &out, bool longlist) const

Data Fields

- std::string **ComputingShareName**
- int **MaxMainMemory**
- int **MaxVirtualMemory**
- int **MaxDiskSpace**
- std::map< **Period**, int > **FreeSlotsWithDuration**
- **Software OperatingSystem**
- std::list< **ApplicationEnvironment** > **ApplicationEnvironments**

6.102.1 Detailed Description

ExecutionTarget (p. 177). 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).

6.102.2 Constructor & Destructor Documentation

6.102.2.1 Arc::ExecutionTarget::ExecutionTarget ( )

Create an **ExecutionTarget** (p. 177).

Default constructor to create an **ExecutionTarget** (p. 177). Takes no arguments.

6.102.2.2 Arc::ExecutionTarget::ExecutionTarget (const **ExecutionTarget** & *target*)

Create an **ExecutionTarget** (p. 177).

Copy constructor.

Parameters

target	**ExecutionTarget** (p. 177) to copy.

6.102.2.3 Arc::ExecutionTarget::ExecutionTarget (const long int *addrptr*)

Create an **ExecutionTarget** (p. 177).

Copy constructor? Needed from Python?

Parameters

addrptr	

6.102.3 Member Function Documentation

6.102.3.1 Submitter∗ Arc::ExecutionTarget::GetSubmitter (const UserConfig & *ucfg*) const

Get **Submitter** (p. 367) to the computing resource represented by the **ExecutionTarget** (p. 177).

Method which returns a specialized **Submitter** (p. 367) which can be used for submitting jobs to the computing resource represented by the **ExecutionTarget** (p. 177). In order to return the correct specialized **Submitter** (p. 367) the GridFlavour variable must be correctly set.

Parameters

ucfg	**UserConfig** (p. 399) object with paths to user credentials etc.

6.102.3.2 ExecutionTarget& Arc::ExecutionTarget::operator= (const ExecutionTarget & *target*)

Create an **ExecutionTarget** (p. 177).

Assignment operator

Parameters

target	is **ExecutionTarget** (p. 177) to copy.

6.102.3.3 void Arc::ExecutionTarget::Print (bool *longlist*) const

DEPRECATED: Print the **ExecutionTarget** (p. 177) information to std::cout.

This method is deprecated, use the SaveToStream method instead. Method to print the **ExecutionTarget** (p. 177) attributes to std::cout

Parameters

longlist	is true for long list printing.

See also

SaveToStream (p. 179)

6.102.3.4 void Arc::ExecutionTarget::SaveToStream (std::ostream & *out*, bool *longlist*) const

Print the **ExecutionTarget** (p. 177) information to a std::ostream object.

Method to print the **ExecutionTarget** (p. 177) attributes to a std::ostream object.

Parameters

out	is the std::ostream to print the attributes to.
longlist	should be set to true for printing a long list.

6.102.3.5 void Arc::ExecutionTarget::Update (const **JobDescription** & *jobdesc*)

Update **ExecutionTarget** (p. 177) after succesful job submission.

Method to update the **ExecutionTarget** (p. 177) 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.

6.102.4 Field Documentation

6.102.4.1 std::list<**ApplicationEnvironment**> **Arc::ExecutionTarget::ApplicationEnvironments**

ApplicationEnvironments.

The ApplicationEnvironments member is a list of ApplicationEnvironment's, defined in section 6.7 GLUE2.

6.102.4.2 std::string **Arc::ExecutionTarget::ComputingShareName**

ComputingShareName String 0..1.

Human-readable name. This variable represents the ComputingShare.Name attribute of GLUE2.

6.102.4.3 std::map<Period, int> Arc::ExecutionTarget::FreeSlotsWithDuration

FreeSlotsWithDuration std::map<Period, int>

This attribute express the number of free slots with their time limits. The keys in the std::map are the time limit (**Period** (p. 298)) for the number of free slots stored as the value (int). If no time limit has been specified for a set of free slots then the key will equal Period(LONG_MAX).

6.102.4.4 int 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.

6.102.4.5 int 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.

6.102.4.6 int 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.

6.102.4.7 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

6.103 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**

6.103.1 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.

6.103.2 Member Function Documentation

6.103.2.1 Glib::TimeVal Arc::ExpirationReminder::getExpiryTime ( ) const

Returns the expiry time.

This method returns the expiry time of the reservation that this **ExpirationReminder** (p. 181) is associated with.

Returns

The expiry time.

6.103.2.2 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. 181) is associated with.

Returns

The identification number.

6.103.2.3 bool Arc::ExpirationReminder::operator< (const ExpirationReminder & *other*) const

Less than operator, compares "soonness".

This is the less than operator for the **ExpirationReminder** (p. 181) 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

6.104 Arc::FileAccess Class Reference

Defines interface for accessing filesystems.

```
#include <FileAccess.h>
```

Data Structures

- struct **header_t**

Public Member Functions

- bool **ping** (void)
- bool **setuid** (int uid, int gid)
- bool **mkdir** (const std::string &path, mode_t mode)
- bool **mkdirp** (const std::string &path, mode_t mode)
- bool **link** (const std::string &oldpath, const std::string &newpath)
- bool **softlink** (const std::string &oldpath, const std::string &newpath)
- bool **copy** (const std::string &oldpath, const std::string &newpath, mode_t mode)
- bool **chmod** (const std::string &path, mode_t mode)
- bool **stat** (const std::string &path, struct stat &st)
- bool **lstat** (const std::string &path, struct stat &st)
- bool **fstat** (struct stat &st)
- bool **ftruncate** (off_t length)
- off_t **fallocate** (off_t length)
- bool **readlink** (const std::string &path, std::string &linkpath)
- bool **remove** (const std::string &path)
- bool **unlink** (const std::string &path)
- bool **rmdir** (const std::string &path)
- bool **rmdirr** (const std::string &path)
- bool **opendir** (const std::string &path)
- bool **closedir** (void)
- bool **readdir** (std::string &name)

- bool **open** (const std::string &path, int flags, mode_t mode)
- bool **close** (void)
- off_t **lseek** (off_t offset, int whence)
- ssize_t **read** (void ∗buf, size_t size)
- ssize_t **write** (const void ∗buf, size_t size)
- ssize_t **pread** (void ∗buf, size_t size, off_t offset)
- ssize_t **pwrite** (const void ∗buf, size_t size, off_t offset)
- int **geterrno** ()
- **operator bool** (void)
- bool **operator!** (void)

Static Public Member Functions

- static void **testtune** (void)

6.104.1 Detailed Description

Defines interface for accessing filesystems. This class accesses local filesystem through proxy executable which allows to switch user id in multithreaded systems without introducing conflict with other threads. Its methods are mostly replicas of corresponding POSIX functions with some convenience tweaking.

6.104.2 Member Function Documentation

6.104.2.1 bool Arc::FileAccess::copy (const std::string & *oldpath*, const std::string & *newpath*, mode_t *mode*)

Copy file to new location. If new file is created it is assigned secified mode.

6.104.2.2 int Arc::FileAccess::geterrno ( ) `[inline]`

Get errno of last operation. Every operation resets errno.

6.104.2.3 bool Arc::FileAccess::mkdirp (const std::string & *path*, mode_t *mode*)

Make a directory and assign it specified mode. If missing all intermediate directories are created too.

6.104.2.4 bool Arc::FileAccess::open (const std::string & *path*, int *flags*, mode_t *mode*)

Open file. Only one file may be open at a time.

6.104.2.5 bool Arc::FileAccess::opendir (const std::string & *path*)

Open directory. Only one directory may be open at a time.

6.104.2.6 bool Arc::FileAccess::setuid (int *uid,* int *gid*)

Modify user uid and gid. If any is set to 0 then executable is switched to original uid/gid.

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

- FileAccess.h

6.105 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** (std::vector< std::string > caches, std::vector< std::string > remote_-caches, std::vector< std::string > draining_caches, std::string id, uid_t job_uid, gid_t job_gid, int cache_max=100, int cache_min=100)
- **FileCache** ()
- bool **Start** (std::string url, bool &available, bool &is_locked, bool use_remote=true)
- 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, bool executable)
- bool **Copy** (std::string dest_path, std::string url, bool executable=false)
- 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)

6.105.1 Detailed Description

FileCache (p. 184) 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. 189) should be called, so that the cache file can be prepared and locked. When a transfer has finished successfully, **Link()** (p. 188) 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. 189) 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. 387) 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. 387) 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. 387) 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 using the **FileLock** (p. 191) class, which creates a lock file with the '.lock' suffix next to the cache file. Calling **Start()** (p. 189) creates this lock and **Stop()** (p. 189) releases it. All processes calling **Start()** (p. 189) must wait until they successfully obtain the lock before downloading can begin. Once a process obtains a lock it must later release it by calling **Stop()** (p. 189) or **StopAndDelete()** (p. 190).

6.105.2 Constructor & Destructor Documentation

6.105.2.1 Arc::FileCache::FileCache (std::string *cache_path,* std::string *id,* uid_t *job_uid,* gid_t *job_gid*)

Create a new **FileCache** (p. 184) 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

6.105.2.2 Arc::FileCache::FileCache (std::vector< std::string > *caches*, std::string *id*, uid_t *job_uid*, gid_t *job_gid*)

Create a new **FileCache** (p. 184) 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

6.105.2.3 Arc::FileCache::FileCache (std::vector< std::string > *caches*, std::vector< std::string > *remote_caches*, std::vector< std::string > *draining_caches*, std::string *id*, uid_t *job_uid*, gid_t *job_gid*, int *cache_max* = `100`, int *cache_min* = `100`)

Create a new **FileCache** (p. 184) instance with multiple cache dirs, remote caches and draining cache directories.

Parameters

caches	a vector of strings describing caches. The format of each string is "cache_-dir[link_path]".
remote_-caches	Same format as caches. These are the paths to caches which are under the control of other Grid Managers and are read-only for this process.
draining_-caches	Same format as caches. These are the paths to caches which are to be drained.
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
cache_max	maximum used space by cache, as percentage of the file system
cache_min	minimum used space by cache, as percentage of the file system

6.105.2.4 Arc::FileCache::FileCache ( ) `[inline]`

Default constructor. Invalid cache.

6.105.3 Member Function Documentation

6.105.3.1 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

6.105.3.2 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

6.105.3.3 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

6.105.3.4 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

6.105.3.5 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. The session directory is accessed under the uid passed in the constructor, and switching uid involves holding a global lock. Therefore care must be taken in a multi-threaded environment.

This method is deprecated - **Link()** (p. 188) should be used instead with copy set to true.

6.105.3.6 std::string Arc::FileCache::File (std::string *url*)

Returns the full pathname of the file in the cache which corresponds to the given url.

6.105.3.7 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

6.105.3.8 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

6.105.3.9 bool Arc::FileCache::Link (std::string *link_path*, std::string *url*, bool *copy*, bool *executable*)

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. 189).

If cache_link_path is set to "." then files will be copied directly to the session directory rather than via the hard link.

The session directory is accessed under the uid and gid passed in the constructor.

Parameters

link_path	path to the session dir for soft-link or new file
url	url of file to link to or copy
copy	If true the file is copied rather than soft-linked to the session dir
executable	If true then file is copied and given execute permissions in the session dir

6.105.3.10 Arc::FileCache::operator bool (void) [inline]

Returns true if object is useable.

6.105.3.11 bool Arc::FileCache::operator== (const **FileCache** & *a*)

Return true if all attributes are equal

6.105.3.12 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.

6.105.3.13 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

6.105.3.14 bool Arc::FileCache::Start (std::string *url*, bool & *available*, bool & *is_locked*, bool *use_remote* = `true`)

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
remote_-caches	Same format as caches. These are the paths to caches which are under the control of other Grid Managers and are read-only for this process.

6.105.3.15 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. 189) 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. 189)), or if it fails to delete the lock file.

Parameters

url	the url of the file that was downloaded

6.105.3.16 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. 189).

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

6.106 FileCacheHash Class Reference

```
#include <FileCacheHash.h>
```

Static Public Member Functions

- static std::string **getHash** (std::string url)
- static int **maxLength** ()

6.106.1 Detailed Description

FileCacheHash (p. 190) provides methods to make hashes from strings. Currently the md5 hash from the openssl library is used.

6.106.2 Member Function Documentation

6.106.2.1 static std::string FileCacheHash::getHash (std::string *url*) `[static]`

Return a hash of the given URL, according to the current hash scheme.

6.106.2.2 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

6.107 Arc::FileInfo Class Reference

FileInfo (p. 191) stores information about files (metadata).

```
#include <FileInfo.h>
```

6.107.1 Detailed Description

FileInfo (p. 191) stores information about files (metadata).

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

- FileInfo.h

6.108 Arc::FileLock Class Reference

A general file locking class.

```
#include <FileLock.h>
```

Public Member Functions

- **FileLock** (const std::string &filename, unsigned int timeout=**DEFAULT_LOCK_-TIMEOUT**, bool use_pid=true)
- bool **acquire** (bool &lock_removed)
- bool **acquire** ()
- bool **release** (bool force=false)
- bool **check** ()

Static Public Member Functions

- static std::string **getLockSuffix** ()

Static Public Attributes

- static const int **DEFAULT_LOCK_TIMEOUT**
- static const std::string **LOCK_SUFFIX**

6.108.1 Detailed Description

A general file locking class. This class can be used when protected access is required to files which are used by multiple processes or threads. Call **acquire()** (p. 192) to obtain a lock and **release()** (p. 193) to release it when finished. **check()** (p. 193) can be used to verify if a lock is valid for the current process. Locks are independent of **FileLock**

(p. 191) objects - locks are only created and destroyed through **acquire()** (p. 192) and **release()** (p. 193), not on creation or destruction of **FileLock** (p. 191) objects.

Unless use_pid is set false, the process ID and hostname of the calling process are stored in a file filename.lock in the form pid. This information is used to determine whether a lock is still valid. It is also possible to specify a timeout on the lock.

To ensure an atomic locking operation, **acquire()** (p. 192) first creates a temporary lock file filename.lock.XXXXXX, then attempts to rename this file to filename.lock. After a successful rename the lock file is checked to make sure the correct process ID and hostname are inside. This eliminates race conditions where multiple processes compete to obtain the lock.

6.108.2 Constructor & Destructor Documentation

6.108.2.1 Arc::FileLock::FileLock (const std::string & *filename*, unsigned int *timeout* = DEFAULT_LOCK_TIMEOUT, bool *use_pid* = `true`)

Create a new **FileLock** (p. 191) object.

Parameters

filename	The name of the file to be locked
timeout	The timeout of the lock
use_pid	If true, use process id in the lock and to determine lock validity

6.108.3 Member Function Documentation

6.108.3.1 bool Arc::FileLock::acquire (bool & *lock_removed*)

Acquire the lock.

Returns true if the lock was acquired successfully. Locks are acquired if no lock file currently exists, or if the current lock file is invalid. A lock is invalid if the process ID inside the lock no longer exists on the host inside the lock, or the age of the lock file is greater than the lock timeout.

Parameters

lock_-removed	Set to true if an existing lock was removed due to being invalid. In this case the caller may decide to check or delete the file as it is potentially corrupted.

Returns

True if lock is successfully acquired

6.108.3.2 bool Arc::FileLock::acquire ( )

Acquire the lock.

Callers can use this version of **acquire()** (p. 192) if they do not care whether an invalid lock was removed in the process of obtaining the lock.

6.108.3.3 bool Arc::FileLock::check ( )

Check the lock is valid.

Returns true if the lock is valid for the current process

6.108.3.4 bool Arc::FileLock::release (bool *force* = `false`)

Release the lock.

Parameters

force	Remove the lock without checking ownership or timeout

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

- FileLock.h

6.109 Arc::FileType Class Reference

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

- JobDescription.h

6.110 Arc::FinderLoader Class Reference

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

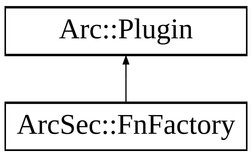
- FinderLoader.h

6.111 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

6.111.1 Detailed Description

Interface for function factory class. **FnFactory** (p. 193) is in charge of creating **Function** (p. 194) 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. 194) objects.

6.111.2 Member Function Documentation

6.111.2.1 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. 194)

Returns

The object of **Function** (p. 194)

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

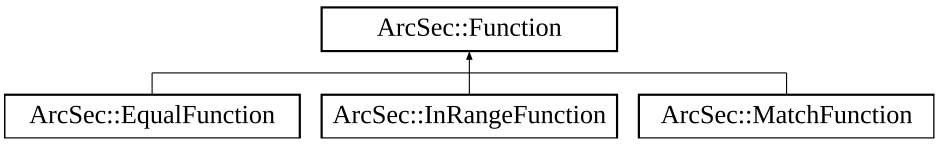
- FnFactory.h

6.112 ArcSec::Function Class Reference

Interface for function, which is in charge of evaluating two **AttributeValue** (p. 63).

`#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

6.112.1 Detailed Description

Interface for function, which is in charge of evaluating two **AttributeValue** (p. 63).

6.112.2 Member Function Documentation

6.112.2.1 virtual AttributeValue∗ ArcSec::Function::evaluate (AttributeValue ∗ *arg0*, AttributeValue ∗ *arg1*, bool *check_id* = `true`) `[pure virtual]`

Evaluate two **AttributeValue** (p. 63) objects, and return one **AttributeValue** (p. 63) object

Implemented in **ArcSec::EqualFunction** (p. 170), **ArcSec::InRangeFunction** (p. 209), and **ArcSec::MatchFunction** (p. 252).

6.112.2.2 virtual std::list<AttributeValue∗> ArcSec::Function::evaluate (std::list< AttributeValue ∗ > *args*, bool *check_id* = `true`) `[pure virtual]`

Evaluate a list of **AttributeValue** (p. 63) objects, and return a list of Attribute objects

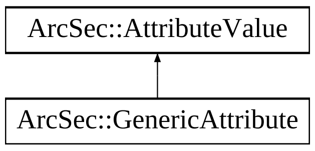
Implemented in **ArcSec::EqualFunction** (p. 170), **ArcSec::InRangeFunction** (p. 209), and **ArcSec::MatchFunction** (p. 252).

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

- Function.h

6.113 ArcSec::GenericAttribute Class Reference

Inheritance diagram for ArcSec::GenericAttribute:



Public Member Functions

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

6.113.1 Member Function Documentation

6.113.1.1 virtual std::string ArcSec::GenericAttribute::encode ( ) [inline, virtual]

encode the value in a string format

Implements **ArcSec::AttributeValue** (p. 64).

6.113.1.2 virtual bool ArcSec::GenericAttribute::equal (AttributeValue * *value*, bool *check_id* = true) [virtual]

Evluate whether "this" equale to the parameter value

Implements **ArcSec::AttributeValue** (p. 64).

6.113.1.3 virtual std::string ArcSec::GenericAttribute::getId ( ) [inline, virtual]

Get the AttributeId of the <Attribute>

Implements **ArcSec::AttributeValue** (p. 64).

6.113.1.4 virtual std::string ArcSec::GenericAttribute::getType ( ) [inline, virtual]

Get the DataType of the <Attribute>

Implements **ArcSec::AttributeValue** (p. 65).

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

- GenericAttribute.h

6.114 Arc::GlobusResult Class Reference

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

- GlobusErrorUtils.h

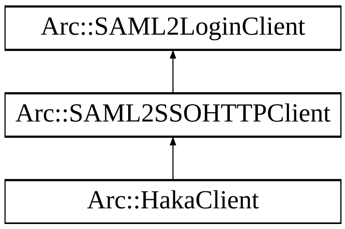
6.115 Arc::GSSCredential Class Reference

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

- GSSCredential.h

6.116 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** ()

6.116.1 Member Function Documentation

6.116.1.1 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. 331).

6.116.1.2 MCC_Status Arc::HakaClient::processIdP2Confusa ( ) `[protected, virtual]`

Redirects the user back from identity provider to the Confusa SP

Implements **Arc::SAML2SSOHTTPClient** (p. 331).

6.116.1.3 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. 332).

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

- HakaClient.h

6.117 Arc::FileAccess::header_t Struct Reference

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

- FileAccess.h

6.118 Arc::HTTPClientInfo Struct Reference

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

- ClientInterface.h

6.119 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)

6.119.1 Detailed Description

Stores XML document in filesystem split into parts.

6.119.2 Constructor & Destructor Documentation

6.119.2.1 Arc::InfoCache::InfoCache (const Config & *cfg,* const std::string & *service_id*)

Creates object according to configuration (see InfoCacheConfig.xsd)

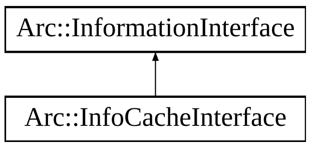
XML configuration is passed in cfg. Argument service_id is used to distiguish between various documents stored under same path - corresponding files will be stored in subdirectory with service_id name.

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

- InfoCache.h

6.120 Arc::InfoCacheInterface Class Reference

Inheritance diagram for Arc::InfoCacheInterface:



Protected Member Functions

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

6.120.1 Member Function Documentation

6.120.1.1 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. 206).

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

- InfoCache.h

6.121 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

6.121.1 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.

6.121.2 Constructor & Destructor Documentation

6.121.2.1 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.

6.121.3 Member Function Documentation

6.121.3.1 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.

6.121.3.2 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.

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

- InfoFilter.h

6.122 Arc::InfoRegister Class Reference

Registration to ISIS interface.

```
#include <InfoRegister.h>
```

6.122.1 Detailed Description

Registration to ISIS interface. This class represents service registering to Information Indexing **Service** (p. 341). 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

6.123 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)

6.123.1 Detailed Description

Singleton class for scanning configuration and storing refernces to registration elements.

6.123.2 Member Function Documentation

6.123.2.1 **InfoRegistrar**∗ Arc::InfoRegisterContainer::addRegistrar (**XMLNode** *doc*)

Adds ISISes to list of handled services.

Supplied configuration document is scanned for **InfoRegistrar** (p. 202) elements and those are turned into **InfoRegistrar** (p. 202) classes for handling connection to ISIS service each.

6.123.2.2 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.

6.123.2.3 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

6.124 Arc::InfoRegisters Class Reference

Handling multiple registrations to ISISes.

`#include <InfoRegister.h>`

Public Member Functions

- **InfoRegisters** (**XMLNode** &cfg, **Service** ∗service_)

6.124.1 Detailed Description

Handling multiple registrations to ISISes.

6.124.2 Constructor & Destructor Documentation

6.124.2.1 Arc::InfoRegisters::InfoRegisters (**XMLNode** & *cfg*, **Service** ∗ *service_*)

Constructor creates **InfoRegister** (p. 200) objects according to configuration.

Inside cfg elements InfoRegistration are found and for each corresponding **InfoRegister** (p. 200) 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

6.125 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** ∗)

6.125.1 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. 202).

6.125.2 Member Function Documentation

6.125.2.1 bool Arc::InfoRegistrar::addService (**InfoRegister** $*$, **XMLNode** &)

Adds new service to list of handled services.

Service (p. 341) is described by it's **InfoRegister** (p. 200) object which must be valid as long as this object is functional.

6.125.2.2 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:

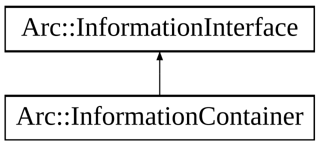
- InfoRegister.h

6.126 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_**

6.126.1 Detailed Description

Information System document container and processor. This class inherits form **InformationInterface** (p. 205) and offers container for storing informational XML document.

6.126.2 Constructor & Destructor Documentation

6.126.2.1 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.

6.126.3 Member Function Documentation

6.126.3.1 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()

6.126.3.2 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.

6.126.3.3 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. 206).

6.126.4 Field Documentation

6.126.4.1 XMLNode Arc::InformationContainer::doc_ [protected]

Either link or container of XML document

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

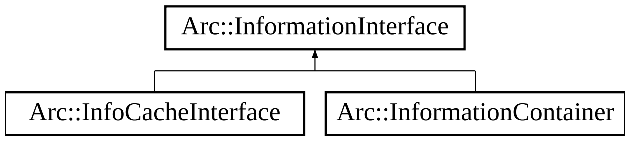
- InformationInterface.h

6.127 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_**

6.127.1 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.

6.127.2 Constructor & Destructor Documentation

6.127.2.1 Arc::InformationInterface::InformationInterface (bool *safe* = `true`)

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

6.127.3 Member Function Documentation

6.127.3.1 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. 199), and **Arc::InformationContainer** (p. 204).

6.127.4 Field Documentation

6.127.4.1 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

6.128 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)

6.128.1 Detailed Description

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

6.128.2 Constructor & Destructor Documentation

6.128.2.1 Arc::InformationRequest::InformationRequest (void)

Dummy constructor

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

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

6.128.2.3 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.

6.128.2.4 Arc::InformationRequest::InformationRequest (XMLNode *query*)

Request for attributes specified by XPath query.

6.128.3 Member Function Documentation

6.128.3.1 SOAPEnvelope∗ Arc::InformationRequest::SOAP (void)

Returns generated SOAP message

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

- InformationInterface.h

6.129 Arc::InformationResponse Class Reference

Informational response from InfoSystem.

```
#include <InformationInterface.h>
```

Public Member Functions

- **InformationResponse** (SOAPEnvelope &soap)
- std::list< **XMLNode** > **Result** (void)

6.129.1 Detailed Description

Informational response from InfoSystem. This is a convenience wrapper analyzing WS-ResourceProperties response from InfoSystem interface of service.

6.129.2 Constructor & Destructor Documentation

6.129.2.1 Arc::InformationResponse::InformationResponse (SOAPEnvelope & *soap*)

Constructor parses WS-ResourceProperties ressponse. Provided SOAPEnvelope object must be valid as long as this object is in use.

6.129.3 Member Function Documentation

6.129.3.1 std::list<XMLNode> Arc::InformationResponse::Result (void)

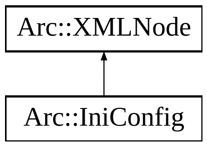
Returns set of attributes which were in SOAP message passed to constructor.

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

- InformationInterface.h

6.130 Arc::IniConfig Class Reference

Inheritance diagram for Arc::IniConfig:



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

- IniConfig.h

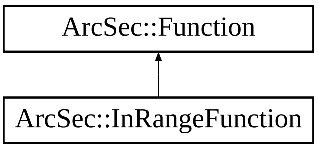
6.131 Arc::initializeCredentialsType Class Reference

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

- UserConfig.h

6.132 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)

6.132.1 Member Function Documentation

6.132.1.1 virtual AttributeValue∗ ArcSec::InRangeFunction::evaluate (AttributeValue ∗ *arg0*, AttributeValue ∗ *arg1*, bool *check_id* = `true`) `[virtual]`

Evaluate two **AttributeValue** (p. 63) objects, and return one **AttributeValue** (p. 63) object

Implements **ArcSec::Function** (p. 195).

6.132.1.2 virtual std::list<AttributeValue∗> ArcSec::InRangeFunction::evaluate (std::list< AttributeValue ∗ > *args*, bool *check_id* = `true`) `[virtual]`

Evaluate a list of **AttributeValue** (p. 63) objects, and return a list of Attribute objects

Implements **ArcSec::Function** (p. 195).

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

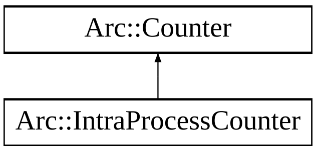
- InRangeFunction.h

6.133 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**)

6.133.1 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. 91) class for further information about counters and examples of usage.

6.133.2 Constructor & Destructor Documentation

6.133.2.1 Arc::IntraProcessCounter::IntraProcessCounter (int *limit*, int *excess*)

Creates an **IntraProcessCounter** (p. 209) 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.

6.133.2.2 virtual Arc::IntraProcessCounter::∼IntraProcessCounter ( ) [virtual]

Destructor.

This is the destructor of the **IntraProcessCounter** (p. 209) class. Does not need to do anything.

6.133.3 Member Function Documentation

6.133.3.1 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. 98) that corresponds to the reservation.

Parameters

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

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

6.133.3.2 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. 94).

6.133.3.3 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. 94).

6.133.3.4 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. 98) 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. 94).

6.133.3.5 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. 95).

6.133.3.6 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. 96).

6.133.3.7 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. 97).

6.133.3.8 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. 98) that can be queried about the status of the reservation as well as for cancellations and extensions.

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

6.133.3.9 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. 97).

6.133.3.10 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. 98).

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

- IntraProcessCounter.h

6.134 Arc::ISIS_description Struct Reference

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

- InfoRegister.h

6.135 Arc::IString Class Reference

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

- IString.h

6.136 Arc::JobDescriptionParserLoader::iterator Class Reference

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

- JobDescriptionParser.h

6.137 Arc::Job Class Reference

Job (p. 215).

`#include <Job.h>`

Public Member Functions

- **Job** ()
- void **Print** (bool longlist) const
- void **SaveToStream** (std::ostream &out, bool longlist) const
- **Job** & **operator=** (**XMLNode** job)
- void **ToXML** (**XMLNode** job) const

Static Public Member Functions

- static bool **ReadAllJobsFromFile** (const std::string &filename, std::list< **Job** > &jobs, unsigned nTries=10, unsigned tryInterval=500000)
- static bool **WriteJobsToTruncatedFile** (const std::string &filename, const std::list< **Job** > &jobs, unsigned nTries=10, unsigned tryInterval=500000)
- static bool **WriteJobsToFile** (const std::string &filename, const std::list< **Job** > &jobs, unsigned nTries=10, unsigned tryInterval=500000)
- static bool **WriteJobsToFile** (const std::string &filename, const std::list< **Job** > &jobs, std::list< const **Job** ∗ > &newJobs, unsigned nTries=10, unsigned tryInterval=500000)
- static bool **RemoveJobsFromFile** (const std::string &filename, const std::list< **URL** > &jobids, unsigned nTries=10, unsigned tryInterval=500000)
- static bool **ReadJobIDsFromFile** (const std::string &filename, std::list< std::string > &jobids, unsigned nTries=10, unsigned tryInterval=500000)
- static bool **WriteJobIDToFile** (const **URL** &jobid, const std::string &filename, unsigned nTries=10, unsigned tryInterval=500000)
- static bool **WriteJobIDsToFile** (const std::list< **URL** > &jobids, const std::string &filename, unsigned nTries=10, unsigned tryInterval=500000)

6.137.1 Detailed Description

Job (p. 215). 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).

6.137.2 Constructor & Destructor Documentation

6.137.2.1 Arc::Job::Job ( )

Create a **Job** (p. 215) object.

Default constructor. Takes no arguments.

6.137.3 Member Function Documentation

6.137.3.1 Job& Arc::Job::operator= (XMLNode *job*)

Set **Job** (p. 215) attributes from a **XMLNode** (p. 465).

The attributes of the **Job** (p. 215) object is set to the values specified in the **XMLNode** (p. 465). The **XMLNode** (p. 465) should be a ComputingActivity type using the GLUE2 XML hierarchical rendering, see `http://forge.gridforum.org/sf/wiki/do/viewPage/projects.` for more information. Note that associations are not parsed.

Parameters

job	is a **XMLNode** (p. 465) of GLUE2 ComputingActivity type.

See also

ToXML (p. 218)

6.137.3.2 void Arc::Job::Print (bool *longlist*) const

DEPRECATED: Print the **Job** (p. 215) information to std::cout.

This method is DEPRECATED, use the SaveToStream method instead. Method to print the **Job** (p. 215) attributes to std::cout

Parameters

longlist	is boolean for long listing (more details).

See also

SaveToStream (p. 218)

6.137.3.3 static bool Arc::Job::ReadAllJobsFromFile (const std::string & *filename*, std::list< Job > & *jobs*, unsigned *nTries* = `10`, unsigned *tryInterval* = `500000`) `[static]`

Read all jobs from file.

This static method will read jobs (in XML format) from the specified file, and they will be stored in the referenced list of jobs. The XML element in the file representing a job should be named "Job", and have the same format as accepted by the **operator=(XMLNode)** (p. 216) method.

File locking: To avoid simultaneous use (writing and reading) of the file, reading will not be initiated before a lock on the file has been acquired. For this purpose the **FileLock** (p. 191) class is used. nTries specifies the maximal number of times the method will try to acquire a lock on the file, with an interval of tryInterval micro seconds between each attempt. If a lock is not acquired∗ this method returns false.

The method will also return false if the content of file is not in XML format. Otherwise it returns true.

Parameters

filename	is the filename of the job list to read jobs from.
jobs	is a reference to a list of **Job** (p. 215) objects, which will be filled with the jobs read from file (cleared before use).
nTries	specifies the maximal number of times the method will try to acquire a lock on file to read.
tryInterval	specifies the interval (in micro seconds) between each attempt to acquire a lock.

Returns

true in case of success, otherwise false.

See also

operator=(XMLNode) (p. 216)
WriteJobsToTruncatedFile (p. 221)
WriteJobsToFile (p. 221)
RemoveJobsFromFile (p. 218)
FileLock (p. 191)
XMLNode::ReadFromFile (p. 476)

6.137.3.4 static bool Arc::Job::ReadJobIDsFromFile (const std::string & *filename*, std::list< std::string > & *jobids*, unsigned *nTries* = `10`, unsigned *tryInterval* = `500000`) `[static]`

Read a list of **Job** (p. 215) IDs from a file, and append them to a list.

This static method will read job IDs from the given file, and append the strings to the string list given as parameter. File locking will be done as described for the ReadAllJobsFromFile method. It returns false if the file was not readable, true otherwise, even if there were no IDs in the file. The lines of the file will be trimmed, and lines starting with # will be ignored.

Parameters

filename	is the filename of the jobidfile
jobids	is a list of strings, to which the IDs read from the file will be appended
nTries	specifies the maximal number of times the method will try to acquire a lock on file to read.
tryInterval	specifies the interval (in micro seconds) between each attempt to acquire a lock.

Returns

true in case of success, otherwise false.

6.137.3.5 static bool Arc::Job::RemoveJobsFromFile (const std::string & *filename*, const std::list< URL > & *jobids*, unsigned *nTries* = `10`, unsigned *tryInterval* = `500000`) `[static]`

Truncate file and write jobs to it.

This static method will remove the jobs having IDFromEndpoint identical to any of those in the passed list jobids. File locking will be done as described for the ReadAllJobsFromFile method. The method will return false if reading from or writing jobs to the file fails. Otherwise it returns true.

Parameters

filename	is the filename of the job list to write jobs to.
jobids	is a list of **URL** (p. 387) objects which specifies which jobs from the file to remove.
nTries	specifies the maximal number of times the method will try to acquire a lock on file to read.
tryInterval	specifies the interval (in micro seconds) between each attempt to acquire a lock.

Returns

true in case of success, otherwise false.

See also

ReadAllJobsFromFile (p. 216)
WriteJobsToTruncatedFile (p. 221)
WriteJobsToFile (p. 221)
FileLock (p. 191)
XMLNode::ReadFromFile (p. 476)
XMLNode::SaveToFile (p. 476)

6.137.3.6 void Arc::Job::SaveToStream (std::ostream & *out*, bool *longlist*) const

Write job information to a std::ostream object.

This method will write job information to the passed std::ostream object. The longlist boolean specifies whether more (true) or less (false) information should be printed.

Parameters

out	is the std::ostream object to print the attributes to.
longlist	is a boolean for switching on long listing (more details).

6.137.3.7 void Arc::Job::ToXML (XMLNode *job*) const

Add job information to a **XMLNode** (p. 465).

Child nodes of GLUE ComputingActivity type containing job information of this object will be added to the passed **XMLNode** (p. 465).

Parameters

job	is the **XMLNode** (p. 465) to add job information to in form of GLUE2 ComputingActivity type child nodes.

See also

operator=

6.137.3.8 static bool Arc::Job::WriteJobIDsToFile (const std::list< URL > & *jobids*, const std::string & *filename*, unsigned *nTries* = `10`, unsigned *tryInterval* = `500000`) `[static]`

Append list of URLs to a file.

This static method will put the ID given as a string, and append it to the given file. File locking will be done as described for the ReadAllJobsFromFile method. It returns false if the file was not writable, true otherwise.

Parameters

jobid	is a list of **URL** (p. 387) objects to be written to file
filename	is the filename of file, where the **URL** (p. 387) objects will be appended to.
nTries	specifies the maximal number of times the method will try to acquire a lock on file to read.
tryInterval	specifies the interval (in micro seconds) between each attempt to acquire a lock.

Returns

true in case of success, otherwise false.

6.137.3.9 static bool Arc::Job::WriteJobIDToFile (const URL & *jobid*, const std::string & *filename*, unsigned *nTries* = `10`, unsigned *tryInterval* = `500000`) `[static]`

Append a jobID to a file.

This static method will put the ID represented by a **URL** (p. 387) object, and append it to the given file. File locking will be done as described for the ReadAllJobsFromFile method. It returns false if the file is not writable, true otherwise.

Parameters

jobid	is a jobID as a **URL** (p. 387) object
filename	is the filename of the jobidfile, where the jobID will be appended
nTries	specifies the maximal number of times the method will try to acquire a lock on file to read.

tryInterval	specifies the interval (in micro seconds) between each attempt to acquire a lock.

Returns

true in case of success, otherwise false.

6.137.3.10 static bool Arc::Job::WriteJobsToFile (const std::string & *filename*, const std::list< Job > & *jobs*, std::list< const Job * > & *newJobs*, unsigned *nTries* = `10`, unsigned *tryInterval* = `500000`) `[static]`

Write jobs to file.

This static method will write (appending) the passed list of jobs to the specified file. Jobs will be written in XML format as returned by the ToXML method, and each job will be contained in a element named "Job". The passed list of jobs must not contain two identical jobs (i.e. IDFromEndpoint identical), if that is the case false will be returned. If on the other hand a job in the list is identical to one in file, the one in file will be overwritten. A pointer (no new) to those jobs from the list which are not in the file will be added to newJobs list, thus these pointers goes out of scope when jobs list goes out of scope. File locking will be done as described for the ReadAllJobsFromFile method. The method will return false if writing jobs to the file fails. Otherwise it returns true.

Parameters

filename	is the filename of the job list to write jobs to.
jobs	is the list of **Job** (p. 215) objects which should be written to file.
newJobs	is a reference to a list of pointers to **Job** (p. 215) objects which are not duplicates (cleared before use).
nTries	specifies the maximal number of times the method will try to acquire a lock on file to read.
tryInterval	specifies the interval (in micro seconds) between each attempt to acquire a lock.

Returns

true in case of success, otherwise false.

See also

ToXML (p. 218)
ReadAllJobsFromFile (p. 216)
WriteJobsToTruncatedFile (p. 221)
RemoveJobsFromFile (p. 218)
FileLock (p. 191)
XMLNode::SaveToFile (p. 476)

6.137.3.11 static bool Arc::Job::WriteJobsToFile (const std::string & *filename*, const std::list< **Job** > & *jobs*, unsigned *nTries* = `10`, unsigned *tryInterval* = `500000`) `[static]`

Write jobs to file.

This method is in all respects identical to the **WriteJobsToFile(const std::string&, const std::list<Job>&, std::list<const Job*>&, unsigned, unsigned)** (p. 220) method, except for the information about new jobs which is disregarded.

See also

> **WriteJobsToFile(const std::string&, const std::list<Job>&, std::list<const Job*>&, unsigned, unsigned)** (p. 220)

6.137.3.12 static bool Arc::Job::WriteJobsToTruncatedFile (const std::string & *filename*, const std::list< **Job** > & *jobs*, unsigned *nTries* = `10`, unsigned *tryInterval* = `500000`) `[static]`

Truncate file and write jobs to it.

This static method will write the passed list of jobs to the specified file, but before writing the file will be truncated. Jobs will be written in XML format as returned by the ToXML method, and each job will be contained in a element named "Job". The passed list of jobs must not contain two identical jobs (i.e. IDFromEndpoint identical), if that is the case false will be returned. File locking will be done as described for the ReadAllJobsFromFile method. The method will return false if writing jobs to the file fails. Otherwise it returns true.

Parameters

filename	is the filename of the job list to write jobs to.
jobs	is the list of **Job** (p. 215) objects which should be written to file.
nTries	specifies the maximal number of times the method will try to acquire a lock on file to read.
tryInterval	specifies the interval (in micro seconds) between each attempt to acquire a lock.

Returns

> true in case of success, otherwise false.

See also

> **ToXML** (p. 218)
> **ReadAllJobsFromFile** (p. 216)
> **WriteJobsToFile** (p. 221)
> **RemoveJobsFromFile** (p. 218)
> **FileLock** (p. 191)
> **XMLNode::SaveToFile** (p. 476)

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

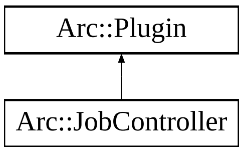
- Job.h

6.138 Arc::JobController Class Reference

Base class for the JobControllers.

`#include <JobController.h>`

Inheritance diagram for Arc::JobController:



Public Member Functions

- void **FillJobStore** (const **Job** &job)
- bool **Cat** (const std::list< std::string > &status, const std::string &whichfile)
- bool **Cat** (std::ostream &out, const std::list< std::string > &status, const std::string &whichfile)
- bool **PrintJobStatus** (const std::list< std::string > &status, const bool longlist)
- bool **SaveJobStatusToStream** (std::ostream &out, const std::list< std::string > &status, bool longlist)
- bool **Migrate** (**TargetGenerator** &targetGen, **Broker** ∗broker, const **UserConfig** &usercfg, const bool forcemigration, std::list< **URL** > &migratedJobIDs)

6.138.1 Detailed Description

Base class for the JobControllers. The **JobController** (p. 222) is the base class for middleware specialized derived classes. The **JobController** (p. 222) 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. 222) are private. The initialization of a (specialized) **JobController** (p. 222) object takes two steps. First the **JobController** (p. 222) specialization for the required grid flavour must be loaded by the **JobControllerLoader** (p. 225), which sees to that the **JobController** (p. 222) 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. 222) job pool (JobStore) which is the pool of jobs that the **JobController** (p. 222) can manage. Must be specialiced for each supported middleware flavour.

6.138.2 Member Function Documentation

6.138.2.1 bool Arc::JobController::Cat (const std::list< std::string > & *status*, const std::string & *whichfile*)

DEPRECATED: Catenate a log-file to standard out.

This method is DEPRECATED, use the **Cat(std::ostream&, const std::list<std::string>&, const std::string&)** (p. 223) instead.

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

Cat(std::ostream&, const std::list<std::string>&, const std::string&) (p. 223)
GetJobInformation
JobState (p. 232)

6.138.2.2 bool Arc::JobController::Cat (std::ostream & *out*, const std::list< std::string > & *status*, const std::string & *whichfile*)

Catenate a output log-file to a std::ostream object.

The method catenates one of the log-files standard out or error, or the job log file from the CE for each of the jobs contained in this object. A file can only be catenated if the location relative to the session directory are set in Job::StdOut, Job::StdErr and Job::LogDir respectively, and if supported so in the specialised ACC module. If the status parameter is non-empty only jobs having a job status specified in this list will considered. The whichfile parameter specifies what log-file to catenate. Possible values are "stdout", "stderr" and "joblog" respectively specifying standard out, error and job log file.

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

SaveJobStatusToStream (p. 224)
GetJobInformation
JobState (p. 232)

6.138.2.3 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. 371) with targets to migrate the job to.
broker	**Broker** (p. 69) 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.

6.138.2.4 bool Arc::JobController::PrintJobStatus (const std::list< std::string > & *status*, const bool *longlist*)

DEPRECATED: Print job status to std::cout.

This method is DEPRECATED, use the SaveJobStatusToStream instead.

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

SaveJobStatusToStream (p. 224)
GetJobInformation
JobState (p. 232)

6.138.2.5 bool Arc::JobController::SaveJobStatusToStream (std::ostream & *out*, const std::list< std::string > & *status*, bool *longlist*)

Print job status to a std::ostream object.

The job status is printed to a std::ostream object when calling this method. More specifically the **Job::SaveToStream** (p. 218) method is called on each of the **Job** (p. 215) 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. 232)) 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

out	a std::ostream object to direct job status information to.
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::SaveToStream (p. 218)
JobState (p. 232)

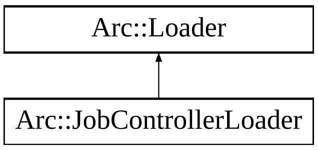
The documentation for this class was generated from the following file:

- JobController.h

6.139 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

6.139.1 Detailed Description

Class responsible for loading **JobController** (p. 222) plugins The **JobController** (p. 222) objects returned by a **JobControllerLoader** (p. 225) must not be used after the **JobControllerLoader** (p. 225) goes out of scope.

6.139.2 Constructor & Destructor Documentation

6.139.2.1 Arc::JobControllerLoader::JobControllerLoader ( )

Constructor Creates a new **JobControllerLoader** (p. 225).

6.139.2.2 Arc::JobControllerLoader::∼JobControllerLoader ( )

Destructor Calling the destructor destroys all JobControllers loaded by the **JobControllerLoader** (p. 225) instance.

6.139.3 Member Function Documentation

6.139.3.1 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().

6.139.3.2 JobController∗ Arc::JobControllerLoader::load (const std::string & *name,* const UserConfig & *usercfg*)

Load a new **JobController** (p. 222)

Parameters

name	The name of the **JobController** (p. 222) to load.
usercfg	The **UserConfig** (p. 399) object for the new **JobController** (p. 222).

Returns

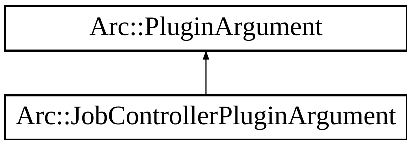
A pointer to the new **JobController** (p. 222) (NULL on error).

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

- JobController.h

6.140 Arc::JobControllerPluginArgument Class Reference

Inheritance diagram for Arc::JobControllerPluginArgument:



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

- JobController.h

6.141 Arc::JobDescription Class Reference

Public Member Functions

- **operator bool** () const
- bool **Parse** (const std::string &source, const std::string &language="", const std::string &dialect="")
- bool **Parse** (const **XMLNode** &xmlSource)
- std::string **UnParse** (const std::string &language="nordugrid:jsdl") const
- bool **UnParse** (std::string &product, std::string language, const std::string &dialect="") const
- const std::string & **GetSourceLanguage** () const
- void **Print** (bool longlist=false) const
- bool **SaveToStream** (std::ostream &out, const std::string &format) const

Static Public Member Functions

- static bool **Parse** (const std::string &source, std::list< **JobDescription** > &jobdescs, const std::string &language="", const std::string &dialect="")

Data Fields

- std::map< std::string, std::string > **OtherAttributes**

6.141.1 Member Function Documentation

6.141.1.1 const std::string& Arc::JobDescription::GetSourceLanguage ( ) const [inline]

Get input source language.

If this object was created by a **JobDescriptionParser** (p. 230), then this method returns a string which indicates the job description language of the parsed source. If not created by a JobDescripionParser the string returned is empty.

Returns

const std::string& source langauge of parsed input source.

6.141.1.2 Arc::JobDescription::operator bool ( ) const

DEPRECATED: Check whether **JobDescription** (p. 227) is valid.

The **JobDescription** (p. 227) class itself is not able to tell whether its objects are valid or not. Instead when parsing/outputting, **JobDescriptionParser** (p. 230) classes checks the validity. Thus the Parse and UnParse methods should be used for this purpose.

6.141.1.3 static bool Arc::JobDescription::Parse (const std::string & *source*, std::list< JobDescription > & *jobdescs*, const std::string & *language* = "", const std::string & *dialect* = "") [static]

Parse string into **JobDescription** (p. 227) objects.

The passed string will be tried parsed into the list of **JobDescription** (p. 227) objects. The available specialized JobDesciptionParser classes will be tried one by one, parsing the string, and if one succeeds the list of **JobDescription** (p. 227) objects is filled with the parsed contents and true is returned, otherwise false is returned. If no language specified, each **JobDescriptionParser** (p. 230) will try all its supported languages. On the other hand if a language is specified, only the **JobDescriptionParser** (p. 230) supporting that language will be tried. A dialect can also be specified, which only has an effect on the parsing if the **JobDescriptionParser** (p. 230) supports that dialect.

Parameters

source	
jobdescs	
language	
dialect	

Returns

true if the passed string can be parsed successfully by any of the available parsers.

6.141.1.4 bool Arc::JobDescription::Parse (const std::string & *source*, const std::string & *language* = "", const std::string & *dialect* = "")

DEPRECATED: Parse source string.

This method is deprecated, use the **Parse(const std::string&, std::list<JobDescription>&, const std::string&, const std::string&)** (p. 228) method instead.

6.141.1.5 bool Arc::JobDescription::Parse (const XMLNode & *xmlSource*)

DEPRECATED: Parse source string.

This method is deprecated, use the **Parse(const std::string&, std::list<JobDescription>&, const std::string&, const std::string&)** (p. 228) method instead.

6.141.1.6 void Arc::JobDescription::Print (bool *longlist* = `false`) const

DEPRECATED: Print all values to standard output.

This method is DEPRECATED, use the SaveToStream method instead.

Parameters

longlist	

See also

SaveToStream (p. 229)

6.141.1.7 bool Arc::JobDescription::SaveToStream (std::ostream & *out*, const std::string & *format*) const

Print job description to a std::ostream object.

The job description will be written to the passed std::ostream object out in the format indicated by the format parameter. The format parameter should specify the format of one of the job description languages supported by the library. Or by specifying the special "user" or "userlong" format the job description will be written as a attribute/value pair list with respectively less or more attributes.

The mote

Returns

true if writing the job description to the out object succeeds, otherwise false.

Parameters

out	a std::ostream reference specifying the ostream to write the job description to.
format	specifies the format the job description should written in.

6.141.1.8 std::string Arc::JobDescription::UnParse (const std::string & *language* = `"nordugrid:jsdl"`) const

DEPRECATED: Output contents in the specified language.

This method is deprecated, use the UnParse(std::string&, std::string, const std::string&) method instead.

6.141.1.9 bool Arc::JobDescription::UnParse (std::string & *product*, std::string *language*, const std::string & *dialect* = " ") const

Output contents in the specified language.

Parameters

product	
language	
dialect	

Returns

6.141.2 Field Documentation

6.141.2.1 std::map<std::string, std::string> Arc::JobDescription::OtherAttributes

Holds attributes not fitting into this class.

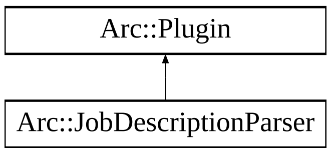
This member is used by **JobDescriptionParser** (p. 230) classes to store attribute/-value pairs not fitting into attributes stored in this class. The form of the attribute (the key in the map) should be as follows: <language>;<attribute-name> E.g.: "nor-dugrid:xrsl;hostname".

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

- JobDescription.h

6.142 Arc::JobDescriptionParser Class Reference

Inheritance diagram for Arc::JobDescriptionParser:



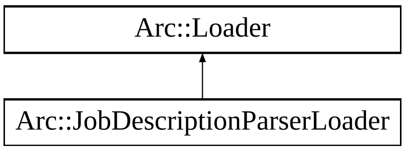
The documentation for this class was generated from the following file:

- JobDescriptionParser.h

6.143 Arc::JobDescriptionParserLoader Class Reference

```
#include <JobDescriptionParser.h>
```

Inheritance diagram for Arc::JobDescriptionParserLoader:



Data Structures

- class **iterator**

Public Member Functions

- **JobDescriptionParserLoader** ()
- ∼**JobDescriptionParserLoader** ()
- **JobDescriptionParser** ∗ **load** (const std::string &name)
- const std::list< **JobDescriptionParser** ∗ > & **GetJobDescriptionParsers** () const

6.143.1 Detailed Description

Class responsible for loading **JobDescriptionParser** (p. 230) plugins The **JobDescriptionParser** (p. 230) objects returned by a **JobDescriptionParserLoader** (p. 230) must not be used after the **JobDescriptionParserLoader** (p. 230) goes out of scope.

6.143.2 Constructor & Destructor Documentation

6.143.2.1 Arc::JobDescriptionParserLoader::JobDescriptionParserLoader ( )

Constructor Creates a new **JobDescriptionParserLoader** (p. 230).

6.143.2.2 Arc::JobDescriptionParserLoader::∼JobDescriptionParserLoader ( )

Destructor Calling the destructor destroys all **JobDescriptionParser** (p. 230) object loaded by the **JobDescriptionParserLoader** (p. 230) instance.

6.143.3 Member Function Documentation

6.143.3.1 const std::list<JobDescriptionParser∗>& Arc::JobDescriptionParserLoader::GetJobDescriptionParsers ( ) const [inline]

Retrieve the list of loaded **JobDescriptionParser** (p. 230) objects.

Returns

A reference to the list of **JobDescriptionParser** (p. 230) objects.

6.143.3.2 JobDescriptionParser∗ Arc::JobDescriptionParserLoader::load (const std::string & *name*)

Load a new **JobDescriptionParser** (p. 230)

Parameters

name	The name of the **JobDescriptionParser** (p. 230) to load.

Returns

A pointer to the new **JobDescriptionParser** (p. 230) (NULL on error).

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

- JobDescriptionParser.h

6.144 Arc::JobIdentificationType Class Reference

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

- JobDescription.h

6.145 Arc::JobState Class Reference

```
#include <JobState.h>
```

Public Member Functions

- bool **IsFinished** () const

6.145.1 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. 232):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.

6.145.2 Member Function Documentation

6.145.2.1 bool Arc::JobState::IsFinished ( ) const `[inline]`

Check if state is finished.

Returns

true is returned if the StateType is equal to FINISHED, KILLED, FAILED or DELETED, otherwise false is returned.

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

- JobState.h

6.146 Arc::JobSupervisor Class Reference

% **JobSupervisor** (p. 233) class

`#include <JobSupervisor.h>`

Public Member Functions

- **JobSupervisor** (const **UserConfig** &usercfg, const std::list< std::string > &jobs)
- **JobSupervisor** (const **UserConfig** &usercfg, const std::list< **Job** > &jobs)
- bool **Resubmit** (const std::list< std::string > &statusfilter, int destination, std::list< **Job** > &resubmittedJobs, std::list< **URL** > &notresubmitted)
- bool **Migrate** (bool forcemigration, std::list< **Job** > &migratedJobs, std::list< **URL** > &notmigrated)
- std::list< **URL** > **Cancel** (const std::list< **URL** > &jobids, std::list< **URL** > &notcancelled)
- std::list< **URL** > **Clean** (const std::list< **URL** > &jobids, std::list< **URL** > &notcleaned)
- const std::list< **JobController** ∗ > & **GetJobControllers** ()

6.146.1 Detailed Description

% **JobSupervisor** (p. 233) class The **JobSupervisor** (p. 233) class is tool for loading **JobController** (p. 222) plugins for managing Grid jobs.

6.146.2 Constructor & Destructor Documentation

6.146.2.1 Arc::JobSupervisor::JobSupervisor (const UserConfig & *usercfg*, const std::list< std::string > & *jobs*)

Create a **JobSupervisor** (p. 233) object.

Default constructor to create a **JobSupervisor** (p. 233). Automatically loads **JobController** (p. 222) plugins based upon the input jobids.

Parameters

usercfg	Reference to **UserConfig** (p. 399) object with information about user credentials and joblistfile.
jobs	List of jobs(jobid or job name) to be managed.

6.146.2.2 Arc::JobSupervisor::JobSupervisor (const UserConfig & *usercfg*, const std::list< Job > & *jobs*)

Create a **JobSupervisor** (p. 233).

The list of **Job** (p. 215) objects passed to the constructor will be managed by this **JobSupervisor** (p. 233), through the **JobController** (p. 222) class. It is important that the Flavour member of each **Job** (p. 215) object is set and correspond to the **JobController** (p. 222) plugin which are capable of managing that specific job. The **JobController** (p. 222) plugin will be loaded using the **JobControllerLoader** (p. 225) class, loading a plugin of type "HED:JobController" and name specified by the Flavour member, and the a reference to the **UserConfig** (p. 399) object usercfg will be passed to the plugin. Additionally a reference to the **UserConfig** (p. 399) object usercfg will be stored, thus usercfg must exist throughout the scope of the created object. If the Flavour member of a **Job** (p. 215) object is unset, a VERBOSE log message will be reported and that **Job** (p. 215) object will be ignored. If the **JobController** (p. 222) plugin for a given Flavour cannot be loaded, a WARNING log message will be reported and any **Job** (p. 215) object with that Flavour will be ignored. If loading of a specific plugin failed, that plugin will not be tried loaded for subsequent **Job** (p. 215) objects requiring that plugin. **Job** (p. 215) objects, for which the corresponding **JobController** (p. 222) plugin loaded successfully, will be added to that plugin using the **JobController::FillJobStore(const Job&)** (p. 222) method.

Parameters

usercfg	**UserConfig** (p. 399) object to pass to **JobController** (p. 222) plugins and to use in member methods.
jobs	List of **Job** (p. 215) objects which will be managed by the created object.

6.146.3 Member Function Documentation

6.146.3.1 std::list<URL> Arc::JobSupervisor::Cancel (const std::list< URL > & *jobids*, std::list< URL > & *notcancelled*)

Cancel jobs.

This method will request cancellation of jobs, identified by their IDFromEndpoint member, for which that **URL** (p. 387) is equal to any in the jobids list. Only jobs corresponding to a **Job** (p. 215) object managed by this **JobSupervisor** (p. 233) will be considered for cancellation. **Job** (p. 215) objects not in a valid state (see **JobState** (p. 232)) will not be considered, and the IDFromEndpoint URLs of those objects will be appended to the notcancelled **URL** (p. 387) list. For jobs not in a finished state (see **JobState::IsFinished** (p. 233)), the JobController::Cancel method will be called, passing the corresponding **Job** (p. 215) object, in order to cancel the job. If the JobController::Cancel call succeeds or if the job is in a finished state the IDFromEndpoint **URL** (p. 387) will be appended to the list to be returned. If the JobController::Cancel call fails the IDFromEndpoint **URL** (p. 387) is appended to the notkilled **URL** (p. 387) list.

Note: If there is any **URL** (p. 387) in the jobids list for which there is no corresponding **Job** (p. 215) object, then the size of the returned list plus the size of the notcancelled list will not equal that of the jobids list.

Parameters

jobids	List of Job::IDFromEndpoint **URL** (p. 387) objects for which a corresponding job, managed by this **JobSupervisor** (p. 233) should be cancelled.
notcancelled	List of Job::IDFromEndpoint **URL** (p. 387) objects for which the corresponding job were not cancelled.

Returns

The list of Job::IDFromEndpoint **URL** (p. 387) objects of successfully cancelled or finished jobs is returned.

6.146.3.2 std::list<URL> Arc::JobSupervisor::Clean (const std::list< URL > & *jobids*, std::list< URL > & *notcleaned*)

Clean jobs.

This method will request cleaning of jobs, identified by their IDFromEndpoint member, for which that **URL** (p. 387) is equal to any in the jobids list. Only jobs corresponding to a **Job** (p. 215) object managed by this **JobSupervisor** (p. 233) will be considered for cleaning. **Job** (p. 215) objects not in a valid state (see **JobState** (p. 232)) will not be considered, and the IDFromEndpoint URLs of those objects will be appended to the notcleaned **URL** (p. 387) list, otherwise the JobController::Clean method will be called, passing the corresponding **Job** (p. 215) object, in order to clean the job. If that method fails the IDFromEndpoint **URL** (p. 387) of the **Job** (p. 215) object will be appended to the notcleaned **URL** (p. 387) list, and if it succeeds the IDFromEndpoint **URL** (p. 387) will be appended to the list of **URL** (p. 387) objects to be returned.

Note: If there is any **URL** (p. 387) in the jobids list for which there is no corresponding **Job** (p. 215) object, then the size of the returned list plus the size of the notcleaned list will not equal that of the jobids list.

Parameters

jobids	List of Job::IDFromEndpoint **URL** (p. 387) objects for which a corresponding job, managed by this **JobSupervisor** (p. 233) should be cleaned.
notcleaned	List of Job::IDFromEndpoint **URL** (p. 387) objects for which the corresponding job were not cleaned.

Returns

The list of Job::IDFromEndpoint **URL** (p. 387) objects of successfully cleaned jobs is returned.

6.146.3.3 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().

6.146.3.4 bool Arc::JobSupervisor::Migrate (bool *forcemigration*, std::list< Job > & *migratedJobs*, std::list< URL > & *notmigrated*)

Migrate jobs.

Jobs managed by this **JobSupervisor** (p. 233) will be migrated when invoking this method, that is the job description of a job will be tried obtained, and if successful a job migration request will be sent, based on that job description.

Before identifying jobs to be migrated, the JobController::GetJobInformation method is called for each loaded **JobController** (p. 222) in order to retrieve the most up to date job information. Only jobs for which the State member of the **Job** (p. 215) object has the value JobState::QUEUEING, will be considered for migration. Furthermore the job description must be obtained (either locally or remote) and successfully parsed in order for a job to be migrated. If the job description cannot be obtained or parsed an ERROR log message is reported, and the IDFromEndpoint **URL** (p. 387) of the **Job** (p. 215) object is appended to the notmigrated list. If no jobs have been identified for migration, false will be returned in case ERRORs were reported, otherwise true is returned.

The execution services which can be targeted for migration are those specified in the **UserConfig** (p. 399) object of this class, as selected services. Before initiating any job migration request, resource discovery and broker∗ loading is carried out using the **TargetGenerator** (p. 371) and **Broker** (p. 69) classes, initialised by the **UserConfig** (p. 399) object of this class. If **Broker** (p. 69) loading fails, or no ExecutionTargets are found, an ERROR log message is reported and all IDFromEndpoint URLs for job considered for migration will be appended to the notmigrated list and then false will be returned.

When the above checks have been carried out successfully, the following is done for each job considered for migration. The ActivityOldId member of the Identification member in the job description will be set to that of the **Job** (p. 215) object, and the IDFromEndpoint **URL** (p. 387) will be appended to ActivityOldId member of the job description. After that the **Broker** (p. 69) object will be used to find a suitable **ExecutionTarget** (p. 177) object, and if found a migrate request will tried sent using the ExecutionTarget::Migrate method, passing the **UserConfig** (p. 399) object of this class. The passed forcemigration boolean indicates whether the migration request at the service side should ignore failures in cancelling the existing queuing job. If the request succeeds, the corresponding new **Job** (p. 215) object is appended to the migratedJobs list. If no suitable **ExecutionTarget** (p. 177) objects are found an ERROR log message is reported and the IDFromEndpoint **URL** (p. 387) of the **Job** (p. 215) object is appended to the notmigrated list. When all jobs have been processed, false is returned if any ERRORs were reported, otherwise true.

Parameters

forcemigration	indicates whether migration should succeed if service fails to cancel the existing queuing job.
migratedJobs	list of **Job** (p. 215) objects which migrated jobs will be appended to.
notmigrated	list of **URL** (p. 387) objects which the IDFromEndpoint **URL** (p. 387) will be appended to.

Returns

false if any error is encountered, otherwise true.

6.146.3.5 bool Arc::JobSupervisor::Resubmit (const std::list< std::string > & *statusfilter*, int *destination*, std::list< Job > & *resubmittedJobs*, std::list< URL > & *notresubmitted*)

Resubmit jobs.

Jobs managed by this **JobSupervisor** (p. 233) will be resubmitted when invoking this method, that is the job description of a job will be tried obtained, and if successful a new job will be submitted.

Before identifying jobs to be resubmitted, the JobController::GetJobInformation method is called for each loaded **JobController** (p. 222) in order to retrieve the most up to date job information. If an empty status-filter is specified, all jobs managed by this **JobSupervisor** (p. 233) will be considered for resubmission, except jobs in the undefined state (see **JobState** (p. 232)). If the status-filter is not empty, then only jobs with a general or specific state (see **JobState** (p. 232)) identical to any of the entries in the status-filter will be considered, except jobs in the undefined state. Jobs for which a job description cannot be obtained and successfully parsed will not be considered and an ERROR log message is reported, and the IDFromEndpoint **URL** (p. 387) is appended to the notresubmitted list. **Job** (p. 215) descriptions will be tried obtained either from

Job (p. 215) object itself, or fetching them remotely. Furthermore if a **Job** (p. 215) object has the LocalInputFiles object set, then the checksum of each of the local input files specified in that object (key) will be calculated and verified to match the checksum LocalInputFiles object (value). If checksums are not matching the job will be filtered, and an ERROR log message is reported and the IDFromEndpoint **URL** (p. 387) is appended to the notresubmitted list. If no job have been identified for resubmission, false will be returned if ERRORs were reported, otherwise true is returned.

The destination for jobs is partly determined by the destination parameter. If a value of 1 is specified a job will only be targeted to the execution service (ES) on which it reside. A value of 2 indicates that a job should not be targeted to the ES it currently reside. Specifying any other value will target any ES. The ESs which can be targeted are those specified in the **UserConfig** (p. 399) object of this class, as selected services. Before initiating any job submission, resource discovery and broker loading is carried out using the **TargetGenerator** (p. 371) and **Broker** (p. 69) classes, initialised by the **UserConfig** (p. 399) object of this class. If **Broker** (p. 69) loading fails, or no ExecutionTargets are found, an ERROR log message is reported and all IDFromEndpoint URLs for job considered for resubmission will be appended to the notresubmitted list and then false will be returned.

When the above checks have been carried out successfully, then the Broker::Submit method will be invoked for each considered for resubmission. If it fails the IDFromEndpoint **URL** (p. 387) for the job is appended to the notresubmitted list, and an ERROR is reported. If submission succeeds the new job represented by a **Job** (p. 215) object will be appended to the resubmittedJobs list - it will not be added to this **JobSupervisor** (p. 233). The method returns false if ERRORs were reported otherwise true is returned.

Parameters

statusfilter	list of job status used for filtering jobs.
destination	specifies how target destination should be determined (1 = same target, 2 = not same, any other = any target).
resubmitted-Jobs	list of **Job** (p. 215) objects which resubmitted jobs will be appended to.
notresubmit-ted	list of **URL** (p. 387) objects which the IDFromEndpoint **URL** (p. 387) will be appended to.

Returns

false if any error is encountered, otherwise true.

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

- JobSupervisor.h

6.147 Arc::LoadableModuleDesciption Class Reference

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

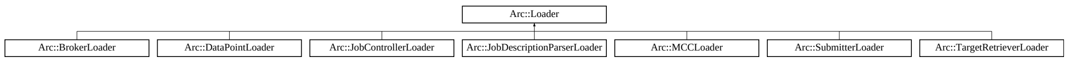
- ModuleManager.h

6.148 Arc::Loader Class Reference

Plugins loader.

`#include <Loader.h>`

Inheritance diagram for Arc::Loader:



Public Member Functions

- **Loader** (**XMLNode** cfg)
- ∼**Loader** ()

Protected Attributes

- **PluginsFactory** ∗ **factory_**

6.148.1 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.

6.148.2 Constructor & Destructor Documentation

6.148.2.1 Arc::Loader::Loader (**XMLNode** *cfg*)

Constructor that takes whole XML configuration and performs common configuration part

6.148.2.2 Arc::Loader::∼Loader ( )

Destructor destroys all components created by constructor

6.148.3 Field Documentation

6.148.3.1 **PluginsFactory**∗ **Arc::Loader::factory_** `[protected]`

Link to Factory responsible for loading and creation of **Plugin** (p. 305) and derived objects

Referenced by Arc::ChainContext::operator PluginsFactory ∗().

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

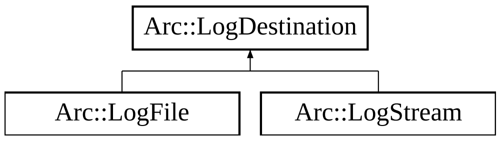
- Loader.h

6.149 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)

6.149.1 Detailed Description

A base class for log destinations. This class defines an interface for LogDestinations. **LogDestination** (p. 239) objects will typically contain synchronization mechanisms and should therefore never be copied.

6.149.2 Constructor & Destructor Documentation

6.149.2.1 Arc::LogDestination::LogDestination ( ) `[protected]`

Default constructor.

This destination will use the default locale.

6.149.2.2 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:

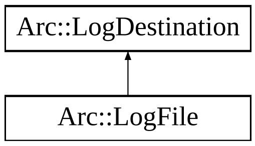
- Logger.h

6.150 Arc::LogFile Class Reference

A class for logging to files.

`#include <Logger.h>`

Inheritance diagram for Arc::LogFile:



Public Member Functions

- **LogFile** (const std::string &path)
- **LogFile** (const std::string &path, const std::string &locale)
- void **setMaxSize** (int newsize)
- void **setBackups** (int newbackup)
- void **setReopen** (bool newreopen)
- **operator bool** (void)
- bool **operator!** (void)
- virtual void **log** (const **LogMessage** &message)

6.150.1 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.

6.150.2 Constructor & Destructor Documentation

6.150.2.1 Arc::LogFile::LogFile (const std::string & *path*)

Creates a **LogFile** (p. 240) connected to a file.

Creates a **LogFile** (p. 240) 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. 240) 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.

6.150.2.2 Arc::LogFile::LogFile (const std::string & *path*, const std::string & *locale*)

Creates a **LogFile** (p. 240) connected to a file.

Creates a **LogFile** (p. 240) connected to the file located at specified path. The output will be localised to the specified locale.

6.150.3 Member Function Documentation

6.150.3.1 virtual void Arc::LogFile::log (const LogMessage & *message*) [virtual]

Writes a **LogMessage** (p. 247) to the file.

This method writes a **LogMessage** (p. 247) to the file that is connected to this **LogFile** (p. 240) object. If after writitng size of file exceeds one set by **setMaxSize()** (p. 242) file is moved to backup and new one is created.

Parameters

message	The **LogMessage** (p. 247) to write.

Implements **Arc::LogDestination** (p. 240).

6.150.3.2 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 **setMax-Size()** (p. 242) 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.

6.150.3.3 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. 247). To disable limit specify -1.

Parameters

newsize	Max size of log file.

6.150.3.4 void Arc::LogFile::setReopen (bool *newreopen*)

Set file reopen on every write.

Set file reopen on every write. If set to true file is opened before writing every log record and closed afterward.

Parameters

newreopen	If file to be reopened for every log record.

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

- Logger.h

6.151 Arc::Logger Class Reference

A logger class.

`#include <Logger.h>`

Public Member Functions

- **Logger** (**Logger** &parent, const std::string &subdomain)
- **Logger** (**Logger** &parent, const std::string &subdomain, **LogLevel** threshold)
- ∼**Logger** ()
- void **addDestination** (**LogDestination** &destination)
- void **addDestinations** (const std::list< **LogDestination** ∗ > &destinations)
- const std::list< **LogDestination** ∗ > & **getDestinations** (void) const
- void **removeDestinations** (void)
- void **setThreshold** (**LogLevel** threshold)
- **LogLevel getThreshold** () const
- void **setThreadContext** (void)
- void **msg** (**LogMessage** message)
- void **msg** (**LogLevel** level, const std::string &str)

Static Public Member Functions

- static **Logger** & **getRootLogger** ()
- static void **setThresholdForDomain** (**LogLevel** threshold, const std::list< std::string > &subdomains)
- static void **setThresholdForDomain** (**LogLevel** threshold, const std::string &domain)

6.151.1 Detailed Description

A logger class. This class defines a **Logger** (p. 243) to which LogMessages can be sent.

Every **Logger** (p. 243) (except for the rootLogger) has a parent **Logger** (p. 243). The domain of a **Logger** (p. 243) (a string that indicates the origin of LogMessages) is composed by adding a subdomain to the domain of its parent **Logger** (p. 243).

A **Logger** (p. 243) also has a threshold. Every **LogMessage** (p. 247) that have a level that is greater than or equal to the threshold is forwarded to any **LogDestination** (p. 239) connected to this **Logger** (p. 243) as well as to the parent **Logger** (p. 243).

Typical usage of the **Logger** (p. 243) class is to declare a global **Logger** (p. 243) object for each library/module/component to be used by all classes and methods there.

6.151.2 Constructor & Destructor Documentation

6.151.2.1 Arc::Logger::Logger (**Logger** & *parent*, const std::string & *subdomain*)

Creates a logger.

Creates a logger. The threshold is inherited from its parent **Logger** (p. 243).

Parameters

parent	The parent **Logger** (p. 243) of the new **Logger** (p. 243).
subdomain	The subdomain of the new logger.

6.151.2.2 Arc::Logger::Logger (**Logger** & *parent*, const std::string & *subdomain*, **LogLevel** *threshold*)

Creates a logger.

Creates a logger.

Parameters

parent	The parent **Logger** (p. 243) of the new **Logger** (p. 243).
subdomain	The subdomain of the new logger.
threshold	The threshold of the new logger.

6.151.2.3 Arc::Logger::∼Logger ( )

Destroys a logger.

Destructor

6.151.3 Member Function Documentation

6.151.3.1 void Arc::Logger::addDestination (LogDestination & *destination*)

Adds a **LogDestination** (p. 239).

Adds a **LogDestination** (p. 239) to which to forward LogMessages sent to this logger (if they pass the threshold). Since LogDestinatoins should not be copied, the new **LogDestination** (p. 239) is passed by reference and a pointer to it is kept for later use. It is therefore important that the **LogDestination** (p. 239) passed to this **Logger** (p. 243) exists at least as long as the **Logger** (p. 243) iteslf.

6.151.3.2 void Arc::Logger::addDestinations (const std::list< LogDestination * > & *destinations*)

Adds LogDestinations.

See **addDestination(LogDestination& destination)** (p. 244).

6.151.3.3 const std::list<LogDestination*>& Arc::Logger::getDestinations (void) const

Obtains current LogDestinations.

Returns list of pointers to **LogDestination** (p. 239) objects. Returned result refers directly to internal member of **Logger** (p. 243) intance. Hence it should not be used after this **Logger** (p. 243) is destroyed.

6.151.3.4 static Logger& Arc::Logger::getRootLogger ( ) `[static]`

The root **Logger** (p. 243).

This is the root **Logger** (p. 243). It is an ancestor of any other **Logger** (p. 243) and allways exists.

6.151.3.5 LogLevel Arc::Logger::getThreshold ( ) const

Returns the threshold.

Returns the threshold.

Returns

The threshold of this **Logger** (p. 243).

6.151.3.6 void Arc::Logger::msg (LogMessage *message*)

Sends a **LogMessage** (p. 247).

Sends a **LogMessage** (p. 247).

Parameters

The	**LogMessage** (p. 247) to send.

Referenced by msg(), and Arc::stringto().

6.151.3.7 void Arc::Logger::msg (LogLevel *level*, const std::string & *str*) [inline]

Logs a message text.

Logs a message text string at the specified LogLevel. This is a convenience method to save some typing. It simply creates a **LogMessage** (p. 247) and sends it to the other **msg()** (p. 245) method.

Parameters

level	The level of the message.
str	The message text.

References msg().

6.151.3.8 void Arc::Logger::setThreadContext (void)

Creates per-thread context.

Creates new context for this logger which becomes effective for operations initiated by this thread. All new threads started by this one will inherit new context. Context stores current threshold and pointers to destinations. Hence new context is identical to current one. One can modify new context using **setThreshold()** (p. 246), **removeDestinations()** (p. 243) and **addDestination()** (p. 244). All such operations will not affect old context.

6.151.3.9 void Arc::Logger::setThreshold (LogLevel *threshold*)

Sets the threshold.

This method sets the threshold of the **Logger** (p. 243). Any message sent to this **Logger** (p. 243) that has a level below this threshold will be discarded.

Parameters

The	threshold

6.151.3.10 static void Arc::Logger::setThresholdForDomain (LogLevel *threshold*, const std::string & *domain*) [static]

Sets the threshold for domain.

This method sets the default threshold of the domain. All new loggers created with specified domain will have specified threshold set by default. The domain is composed of all subdomains of all loggers in chain by merging them with '.' as separator.

Parameters

threshold	The threshold
domain	The domain of logger

6.151.3.11 static void Arc::Logger::setThresholdForDomain (LogLevel *threshold,* const std::list< std::string > & *subdomains*) [static]

Sets the threshold for domain.

This method sets the default threshold of the domain. All new loggers created with specified domain will have specified threshold set by default. The subdomains of all loggers in chain are matched against list of provided subdomains.

Parameters

threshold	The threshold
subdomains	The subdomains of all loggers in chain

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

- Logger.h

6.152 Arc::LoggerContext Class Reference

Container for logger configuration.

`#include <Logger.h>`

6.152.1 Detailed Description

Container for logger configuration.

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

- Logger.h

6.153 Arc::LoggerFormat Struct Reference

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

- Logger.h

6.154 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)

6.154.1 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.

6.154.2 Constructor & Destructor Documentation

6.154.2.1 Arc::LogMessage::LogMessage (LogLevel *level*, const IString & *message*)

Creates a **LogMessage** (p. 247) with the specified level and message text.

This constructor creates a **LogMessage** (p. 247) with the specified level and message text. The time is set automatically, the domain is set by the **Logger** (p. 243) to which the **LogMessage** (p. 247) 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. 247).
message	The message text.

6.154.2.2 Arc::LogMessage::LogMessage (**LogLevel** *level*, const **IString** & *message*, const std::string & *identifier*)

Creates a **LogMessage** (p. 247) with the specified attributes.

This constructor creates a **LogMessage** (p. 247) with the specified level, message text and identifier. The time is set automatically and the domain is set by the **Logger** (p. 243) to which the **LogMessage** (p. 247) is sent.

Parameters

level	The level of the **LogMessage** (p. 247).
message	The message text.
ident	The identifier of the **LogMessage** (p. 247).

6.154.3 Member Function Documentation

6.154.3.1 **LogLevel** Arc::LogMessage::getLevel ( ) const

Returns the level of the **LogMessage** (p. 247).

Returns the level of the **LogMessage** (p. 247).

Returns

The level of the **LogMessage** (p. 247).

6.154.3.2 void Arc::LogMessage::setIdentifier (std::string *identifier*) `[protected]`

Sets the identifier of the **LogMessage** (p. 247).

The purpose of this method is to allow subclasses (in case there are any) to set the identifier of a **LogMessage** (p. 247).

Parameters

The	identifier.

6.154.4 Friends And Related Function Documentation

6.154.4.1 friend class **Logger** `[friend]`

The **Logger** (p. 243) class is a friend.

The **Logger** (p. 243) class must have some privileges (e.g. ability to call the setDomain() method), therefore it is a friend.

6.154.4.2 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:

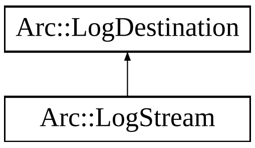
- Logger.h

6.155 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)

6.155.1 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. 249) object as long as the **Logger** (p. 243) to which it has been registered.

6.155.2 Constructor & Destructor Documentation

6.155.2.1 Arc::LogStream::LogStream (std::ostream & *destination*)

Creates a **LogStream** (p. 249) connected to an ostream.

Creates a **LogStream** (p. 249) connected to the specified ostream. In order not to break synchronization, it is important not to connect more than one **LogStream** (p. 249) object to a certain stream.

Parameters

destination	The ostream to which to erite LogMessages.

6.155.2.2 Arc::LogStream::LogStream (std::ostream & *destination*, const std::string & *locale*)

Creates a **LogStream** (p. 249) connected to an ostream.

Creates a **LogStream** (p. 249) connected to the specified ostream. The output will be localised to the specified locale.

6.155.3 Member Function Documentation

6.155.3.1 virtual void Arc::LogStream::log (const LogMessage & *message*) `[virtual]`

Writes a **LogMessage** (p. 247) to the stream.

This method writes a **LogMessage** (p. 247) to the ostream that is connected to this **LogStream** (p. 249) object. It is synchronized so that not more than one **LogMessage** (p. 247) can be written at a time.

Parameters

message	The **LogMessage** (p. 247) to write.

Implements **Arc::LogDestination** (p. 240).

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

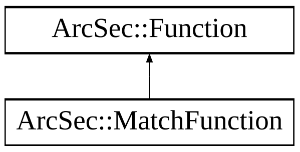
- Logger.h

6.156 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)

6.156.1 Detailed Description

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

6.156.2 Member Function Documentation

6.156.2.1 virtual AttributeValue∗ ArcSec::MatchFunction::evaluate (AttributeValue ∗ *arg0*, AttributeValue ∗ *arg1*, bool *check_id* = `true`) `[virtual]`

Evaluate two **AttributeValue** (p. 63) objects, and return one **AttributeValue** (p. 63) object

Implements **ArcSec::Function** (p. 195).

6.156.2.2 virtual std::list<AttributeValue∗> ArcSec::MatchFunction::evaluate (std::list< AttributeValue ∗ > *args*, bool *check_id* = `true`) `[virtual]`

Evaluate a list of **AttributeValue** (p. 63) objects, and return a list of Attribute objects

Implements **ArcSec::Function** (p. 195).

6.156.2.3 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:

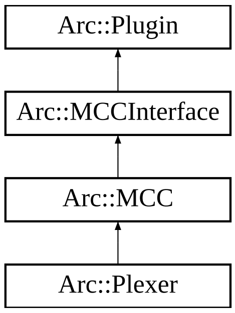
- MatchFunction.h

6.157 Arc::MCC Class Reference

Message (p. 262) Chain Component - base class for every **MCC** (p. 252) 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="") const

Protected Attributes

- std::map< std::string, **MCCInterface** ∗ > **next_**
- std::map< std::string, std::list< **ArcSec::SecHandler** ∗ > > **sechandlers_**

Static Protected Attributes

- static **Logger logger**

6.157.1 Detailed Description

Message (p. 262) Chain Component - base class for every **MCC** (p. 252) plugin. This is partialy virtual class which defines interface and common functionality for every **MCC** (p. 252) plugin needed for managing of component in a chain.

6.157.2 Constructor & Destructor Documentation

6.157.2.1 Arc::MCC::MCC (**Config** ∗) `[inline]`

Example contructor - **MCC** (p. 252) takes at least it's configuration subtree

6.157.3 Member Function Documentation

6.157.3.1 virtual void Arc::MCC::AddSecHandler (**Config** ∗ *cfg,* **ArcSec::SecHandler** ∗ *sechandler,* const std::string & *label* = "") `[virtual]`

Add security components/handlers to this **MCC** (p. 252). 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. 252) 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. 252) 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.

6.157.3.2 virtual void Arc::MCC::Next (**MCCInterface** ∗ *next,* const std::string & *label* = "") `[virtual]`

Add reference to next **MCC** (p. 252) in chain. This method is called by **Loader** (p. 238) for every potentially labeled link to next component which implements **MCCInterface** (p. 258). If next is NULL corresponding link is removed.

Reimplemented in **Arc::Plexer** (p. 304).

6.157.3.3 virtual **MCC_Status** Arc::MCC::process (**Message** & , **Message** &) `[inline, virtual]`

Dummy **Message** (p. 262) processing method. Just a placeholder.

Implements **Arc::MCCInterface** (p. 259).

Reimplemented in **Arc::Plexer** (p. 304).

6.157.3.4 bool Arc::MCC::ProcessSecHandlers (**Message** & *message,* const std::string & *label* = "") const `[protected]`

Executes security handlers of specified queue. Returns true if the message is authorized for further processing or if there are no security handlers which implement authorization functionality. This is a convenience method and has to be called by the implemention of the **MCC** (p. 252).

6.157.3.5 virtual void Arc::MCC::Unlink ( ) [virtual]

Removing all links. Useful for destroying chains.

6.157.4 Field Documentation

6.157.4.1 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. 305).

6.157.4.2 std::map<std::string, MCCInterface *> Arc::MCC::next_ [protected]

Set of labeled "next" components. Each implemented **MCC** (p. 252) must call **process()** (p. 254) method of corresponding **MCCInterface** (p. 258) from this set in own **process()** (p. 254) method.

6.157.4.3 std::map<std::string, std::list<ArcSec::SecHandler *> > Arc::MCC::sechandlers_ [protected]

Set of labeled authentication and authorization handlers. **MCC** (p. 252) 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

6.158 Arc::MCC_Status Class Reference

A class for communication of **MCC** (p. 252) 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

6.158.1 Detailed Description

A class for communication of **MCC** (p. 252) processing results. This class is used to communicate result status between MCCs. It contains a status kind, a string specifying the origin (**MCC** (p. 252)) of the status object and an explanation.

6.158.2 Constructor & Destructor Documentation

6.158.2.1 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. 255) object.

Parameters

kind	The StatusKind (default: STATUS_UNDEFINED)
origin	The origin **MCC** (p. 252) (default: "???")
explanation	An explanation (default: "No explanation.")

6.158.3 Member Function Documentation

6.158.3.1 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.

6.158.3.2 StatusKind Arc::MCC_Status::getKind ( ) const

Returns the status kind.

Returns the status kind of this object.

Returns

The status kind of this object.

6.158.3.3 const std::string& Arc::MCC_Status::getOrigin ( ) const

Returns the origin.

This method returns a string specifying the origin **MCC** (p. 252) of this object.

Returns

A string specifying the origin **MCC** (p. 252) of this object.

6.158.3.4 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!().

6.158.3.5 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().

6.158.3.6 Arc::MCC_Status::operator std::string ( ) const

Conversion to string.

This operator converts a **MCC_Status** (p. 255) object to a string.

6.158.3.7 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

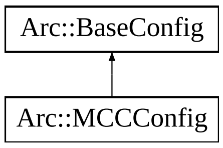
References isOk().

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

- MCC_Status.h

6.159 Arc::MCCConfig Class Reference

Inheritance diagram for Arc::MCCConfig:



Public Member Functions

- virtual **XMLNode MakeConfig** (**XMLNode** cfg) const

6.159.1 Member Function Documentation

6.159.1.1 virtual XMLNode Arc::MCCConfig::MakeConfig (XMLNode *cfg*) const [virtual]

Adds configuration part corresponding to stored information into common configuration tree supplied in 'cfg' argument. Returns reference to XML node representing configuration of **ModuleManager** (p. 271)

Reimplemented from **Arc::BaseConfig** (p. 68).

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

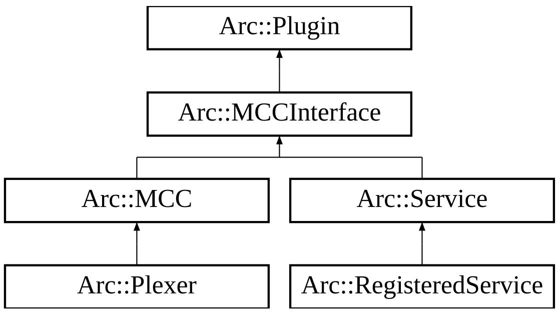
- MCC.h

6.160 Arc::MCCInterface Class Reference

Interface for communication between **MCC** (p. 252), **Service** (p. 341) and **Plexer** (p. 303) objects.

`#include <MCC.h>`

Inheritance diagram for Arc::MCCInterface:



Public Member Functions

- virtual **MCC_Status process** (**Message** &request, **Message** &response)=0

6.160.1 Detailed Description

Interface for communication between **MCC** (p. 252), **Service** (p. 341) and **Plexer** (p. 303) objects. The Interface consists of the method **process()** (p. 259) which is called by the previous **MCC** (p. 252) in the chain. For memory management policies please read the description of the **Message** (p. 262) class.

6.160.2 Member Function Documentation

6.160.2.1 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. 252) in chain when a request needs to be processed. This method must call similar method of next **MCC** (p. 252) in chain unless any failure happens. Result returned by call to next **MCC** (p. 252) should be processed and passed back to previous **MCC** (p. 252). In case of failure this method is expected to generate valid error response and return it back to previous **MCC** (p. 252) without calling the next one.

Parameters

request	The request that needs to be processed.
response	A **Message** (p. 262) object that will contain the response of the request when the method returns.

Returns

An object representing the status of the call.

Implemented in **Arc::MCC** (p. 254), and **Arc::Plexer** (p. 304).

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

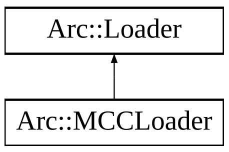
- MCC.h

6.161 Arc::MCCLoader Class Reference

Creator of **Message** (p. 262) Component Chains (**MCC** (p. 252)).

`#include <MCCLoader.h>`

Inheritance diagram for Arc::MCCLoader:



Public Member Functions

- **MCCLoader** (**Config** &cfg)
- ∼**MCCLoader** ()
- **MCC** ∗ **operator[]** (const std::string &id)

6.161.1 Detailed Description

Creator of **Message** (p. 262) Component Chains (**MCC** (p. 252)). 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. 252), **Service** (p. 341) and **Plexer** (p. 303). **MCC** (p. 252) and **Service** (p. 341) are loaded from dynamic libraries. For **Plexer** (p. 303) 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. 262) 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.

6.161.2 Constructor & Destructor Documentation

6.161.2.1 Arc::MCCLoader::MCCLoader (Config & *cfg*)

Constructor that takes whole XML configuration and creates component chains

6.161.2.2 Arc::MCCLoader::∼MCCLoader ( )

Destructor destroys all components created by constructor

6.161.3 Member Function Documentation

6.161.3.1 MCC∗ Arc::MCCLoader::operator[] (const std::string & *id*)

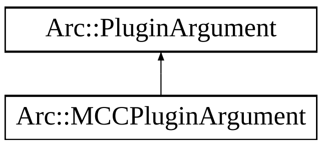
Access entry MCCs in chains. Those are components exposed for external access using 'entry' attribute

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

- MCCLoader.h

6.162 Arc::MCCPluginArgument Class Reference

Inheritance diagram for Arc::MCCPluginArgument:



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

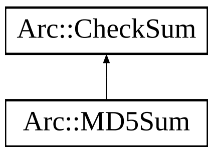
- MCC.h

6.163 Arc::MD5Sum Class Reference

Implementation of MD5 checksum.

`#include <CheckSum.h>`

Inheritance diagram for Arc::MD5Sum:



6.163.1 Detailed Description

Implementation of MD5 checksum.

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

- CheckSum.h

6.164 Arc::MemoryAllocationException Class Reference

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

- ByteArray.h

6.165 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)

6.165.1 Detailed Description

Object being passed through chain of MCCs. An instance of this class refers to objects with main content (**MessagePayload** (p. 271)), authentication/authorization information (**MessageAuth** (p. 268)) and common purpose attributes (**MessageAttributes** (p. 265)). **Message** (p. 262) class does not manage pointers to objects and their content. It only serves for grouping those objects. **Message** (p. 262) objects are supposed to be processed by MCCs and Services implementing **MCCInterface** (p. 258) method process(). All objects constituting content of **Message** (p. 262) 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. 262). b) Objects whose management is completely acquired by objects assigned to 'response' **Message** (p. 262).

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. 262) object).

4. It is allowed to change content of pointers of 'request' **Message** (p. 262). Calling process() method must not rely on that object to stay intact.

5. Called process() method should either fill 'response' **Message** (p. 262) with pointers to valid objects or to keep them intact. This makes it possible for calling process() to preload 'response' with valid error message.

6.165.2 Constructor & Destructor Documentation

6.165.2.1 Arc::Message::Message (void) `[inline]`

true if auth_ctx_ was created internally Dummy constructor

6.165.2.2 Arc::Message::Message (Message & *msg*) `[inline]`

Copy constructor. Ensures shallow copy.

6.165.2.3 Arc::Message::Message (long *msg_ptr_addr*)

Copy constructor. Used by language bindigs

6.165.2.4 Arc::Message::∼Message (void) `[inline]`

Destructor does not affect refered objects except those created internally

6.165.3 Member Function Documentation

6.165.3.1 MessageAttributes∗ Arc::Message::Attributes (void) [inline]

Returns a pointer to the current attributes object or creates it if no attributes object has been assigned.

6.165.3.2 MessageAuth∗ Arc::Message::Auth (void) [inline]

Returns a pointer to the current authentication/authorization object or creates it if no object has been assigned.

6.165.3.3 MessageAuthContext∗ Arc::Message::AuthContext (void) [inline]

Returns a pointer to the current auth∗ context object or creates it if no object has been assigned.

6.165.3.4 void Arc::Message::AuthContext (MessageAuthContext ∗ *auth_ctx*) [inline]

Assigns auth∗ context object

6.165.3.5 void Arc::Message::Context (MessageContext ∗ *ctx*) [inline]

Assigns message context object

6.165.3.6 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. 252) in a chain is connectionless like one implementing UDP protocol.

6.165.3.7 Message& Arc::Message::operator= (Message & *msg*) [inline]

Assignment. Ensures shallow copy.

6.165.3.8 MessagePayload∗ Arc::Message::Payload (void) [inline]

Returns pointer to current payload or NULL if no payload assigned.

6.165.3.9 MessagePayload∗ Arc::Message::Payload (MessagePayload ∗ *payload*) [inline]

Replaces payload with new one. Returns the old one.

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

- Message.h

6.166 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_**

6.166.1 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. 262) Chain Component (**MCC** (p. 252)) 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. 252) 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. 252). 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. 252) and used by the plexer for routing the message to the appropriate service.

6.166.2 Constructor & Destructor Documentation

6.166.2.1 Arc::MessageAttributes::MessageAttributes ( )

The default constructor.

This is the default constructor of the **MessageAttributes** (p. 265) class. It constructs an empty object that initially contains no attributes.

6.166.3 Member Function Documentation

6.166.3.1 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.

6.166.3.2 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.

6.166.3.3 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.

6.166.3.4 AttributeIterator Arc::MessageAttributes::getAll (const std::string & *key*) const

Access the value(s) of an attribute.

This method returns an **AttributeIterator** (p. 59) 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. 59) for access of the values of the attribute.

6.166.3.5 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.

6.166.3.6 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.

6.166.3.7 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.

6.166.4 Field Documentation

6.166.4.1 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:

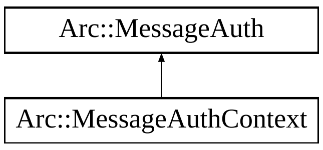
- MessageAttributes.h

6.167 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)

6.167.1 Detailed Description

Contains authencity information, authorization tokens and decisions. This class only supports string keys and **SecAttr** (p. 336) values.

6.167.2 Member Function Documentation

6.167.2.1 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. 269) tries to merge generated information to already existing like everything would be generated inside same **Export()** (p. 269) method. If does not represent valid node then new XML tree is created.

6.167.2.2 **MessageAuth**∗ Arc::MessageAuth::Filter (const std::list< std::string > & *selected_keys*, const std::list< std::string > & *rejected_keys*)

Creates new instance of **MessageAuth** (p. 268) with attributes filtered.

In new instance all attributes with keys listed in are removed. If is not empty only corresponding attributes are transferred to new instance. Created instance does not own refered attributes. Hence parent instance must not be deleted as long as this one is in use.

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

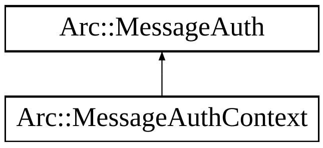
- MessageAuth.h

6.168 Arc::MessageAuthContext Class Reference

Handler for content of message auth∗ context.

`#include <Message.h>`

Inheritance diagram for Arc::MessageAuthContext:



6.168.1 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. 262) object usually by first **MCC** (p. 252) in a chain and is kept as long as connection persists.

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

- Message.h

6.169 Arc::MessageContext Class Reference

Handler for content of message context.

`#include <Message.h>`

Public Member Functions

- void **Add** (const std::string &name, **MessageContextElement** ∗element)

6.169.1 Detailed Description

Handler for content of message context. This class is a container for objects derived from **MessageContextElement** (p. 270). It gets associated with **Message** (p. 262) object usually by first **MCC** (p. 252) in a chain and is kept as long as connection persists.

6.169.2 Member Function Documentation

6.169.2.1 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:

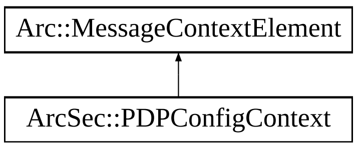
- Message.h

6.170 Arc::MessageContextElement Class Reference

Top class for elements contained in message context.

`#include <Message.h>`

Inheritance diagram for Arc::MessageContextElement:



6.170.1 Detailed Description

Top class for elements contained in message context. Objects of classes inherited with this one may be stored in **MessageContext** (p. 270) container.

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

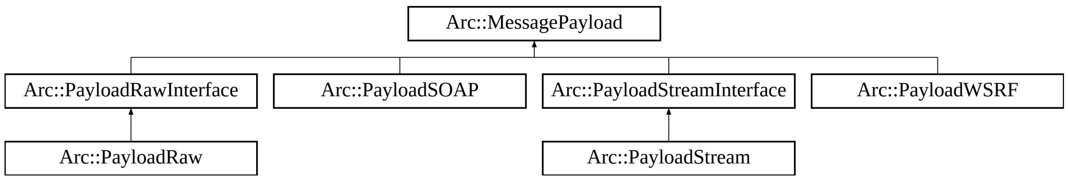
- Message.h

6.171 Arc::MessagePayload Class Reference

Base class for content of message passed through chain.

`#include <Message.h>`

Inheritance diagram for Arc::MessagePayload:



6.171.1 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

6.172 Arc::ModuleDesc Class Reference

Description of loadable module.

`#include <Plugin.h>`

6.172.1 Detailed Description

Description of loadable module. This class is used for reports

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

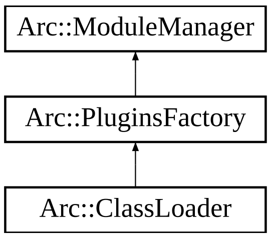
- Plugin.h

6.173 Arc::ModuleManager Class Reference

Manager of shared libraries.

```
#include <ModuleManager.h>
```

Inheritance diagram for Arc::ModuleManager:



Public Member Functions

- **ModuleManager** (**XMLNode** cfg)
- Glib::Module ∗ **load** (const std::string &name, bool probe=false)
- std::string **find** (const std::string &name)
- 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** (**XMLNode** 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 (XMLNode *cfg*)

Cache of handles of loaded modules Constructor. It is supposed to process correponding configuration subtree and tune module loading parameters accordingly.

6.173.3 Member Function Documentation

6.173.3.1 std::string Arc::ModuleManager::find (const std::string & *name*)

Finds loadable module by 'name' looking in same places as **load()** (p. 273) does, but does not load it.

6.173.3.2 std::string Arc::ModuleManager::findLocation (const std::string & *name*)

Finds shared library corresponding to module 'name' and returns path to it

6.173.3.3 Glib::Module∗ Arc::ModuleManager::load (const std::string & *name*, bool *probe* = `false`)

Finds module 'name' in cache or loads corresponding loadable module

6.173.3.4 bool Arc::ModuleManager::makePersistent (const std::string & *name*)

Make sure this module is never unloaded. Even if **unload()** (p. 273) is called.

6.173.3.5 bool Arc::ModuleManager::makePersistent (Glib::Module ∗ *module*)

Make sure this module is never unloaded. Even if **unload()** (p. 273) is called.

6.173.3.6 Glib::Module∗ Arc::ModuleManager::reload (Glib::Module ∗ *module*)

Reload module previously loaded in probe mode. New module is loaded with all symbols resolved and old module handler is unloaded. In case of error old module is not unloaded.

6.173.3.7 void Arc::ModuleManager::setCfg (XMLNode *cfg*)

Input the configuration subtree, and trigger the module loading (do almost the same as the Constructor); It is function desgined for **ClassLoader** (p. 77) to adopt the singleton pattern

6.173.3.8 void Arc::ModuleManager::unload (const std::string & *name*)

Unload module by its name

6.173.3.9 void Arc::ModuleManager::unload (Glib::Module ∗ *module*)

Unload module by its identifier

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

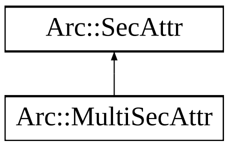
- ModuleManager.h

6.174 Arc::MultiSecAttr Class Reference

Container of multiple **SecAttr** (p. 336) 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. 336) 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. 337).

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. 337).

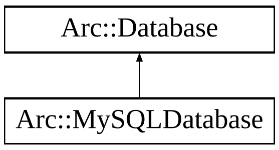
The documentation for this class was generated from the following file:

- 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. 112).

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. 112).

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. 112).

6.175.2.4 virtual bool Arc::MySQLDatabase::isconnected ( ) const [inline, virtual]

Get the connection status

Implements **Arc::Database** (p. 112).

6.175.2.5 virtual bool Arc::MySQLDatabase::shutdown ( ) [virtual]

Ask database server to shutdown

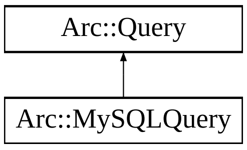
Implements **Arc::Database** (p. 113).

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. 317).

6.176.1.2 virtual bool Arc::MySQLQuery::get_array (std::string & *sqlstr,* QueryArrayResult & *result,* std::vector< std::string > & *arguments*) [virtual]

Query (p. 316) 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. 317).

6.176.1.3 virtual int Arc::MySQLQuery::get_num_colums ( ) [virtual]

Get the colum number in the query result

Implements **Arc::Query** (p. 317).

6.176.1.4 virtual int Arc::MySQLQuery::get_num_rows ( ) [virtual]

Get the row number in the query result

Implements **Arc::Query** (p. 318).

6.176.1.5 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. 318).

6.176.1.6 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. 318).

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. 318).

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

- MysqlWrapper.h

6.177 Arc::NotificationType Class Reference

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

- JobDescription.h

6.178 Arc::NS Class Reference

Public Member Functions

- **NS** (void)
- **NS** (const char ∗prefix, const char ∗uri)
- **NS** (const char ∗nslist[ ][2])

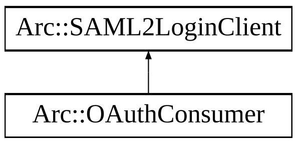
The documentation for this class was generated from the following file:

- XMLNode.h

6.179 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 pass-word="")

6.179.1 Detailed Description

The OAuth functionality depends on the availability of the liboauth C-bindings library

6.179.2 Constructor & Destructor Documentation

6.179.2.1 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 hard-coded value on the SAML2 SP side. This is expected to change in the future.

6.179.3 Member Function Documentation

6.179.3.1 MCC_Status Arc::OAuthConsumer::approveCSR (const std::string *approve_page*) [virtual]

Unsupported placeholder function until Confusa supports OAuth.

Implements **Arc::SAML2LoginClient** (p. 329).

6.179.3.2 MCC_Status Arc::OAuthConsumer::parseDN (std::string * *dn*) [virtual]

Unsupported placeholder function until Confusa supports OAuth.

Implements **Arc::SAML2LoginClient** (p. 329).

6.179.3.3 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. 330).

6.179.3.4 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. 329).

6.179.3.5 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.

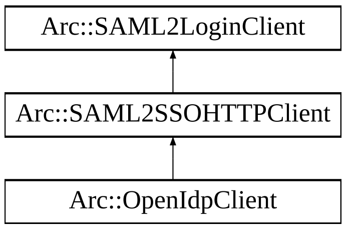
Implements **Arc::SAML2LoginClient** (p. 329).

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

- OAuthConsumer.h

6.180 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** ()

6.180.1 Member Function Documentation

6.180.1.1 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. 331).

6.180.1.2 MCC_Status Arc::OpenIdpClient::processIdP2Confusa ( ) `[protected, virtual]`

Redirects the user back from identity provider to the Confusa SP

Implements **Arc::SAML2SSOHTTPClient** (p. 331).

6.180.1.3 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. 332).

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

- OpenIdpClient.h

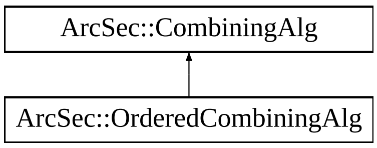
6.181 Arc::OptionParser Class Reference

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

- OptionParser.h

6.182 ArcSec::OrderedCombiningAlg Class Reference

Inheritance diagram for ArcSec::OrderedCombiningAlg:



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

- OrderedAlg.h

6.183 passwd Struct Reference

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

- win32.h

6.184 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

6.184.1 Detailed Description

Class to iterate through elements of path.

6.184.2 Constructor & Destructor Documentation

6.184.2.1 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.

6.184.3 Member Function Documentation

6.184.3.1 Arc::PathIterator::operator bool ( ) const

Return false when iterator moved outside path elements

6.184.3.2 std::string Arc::PathIterator::operator* ( ) const

Returns part of initial path from first till and including current

6.184.3.3 PathIterator& Arc::PathIterator::operator++ ( )

Advances iterator to point at next path element

6.184.3.4 PathIterator& Arc::PathIterator::operator-- ( )

Moves iterator to element before current

6.184.3.5 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:

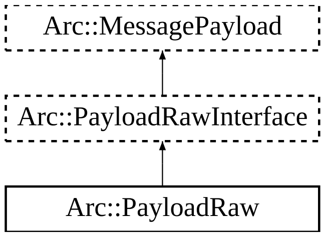
- **URL.h**

6.185 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)

6.185.1 Detailed Description

Raw byte multi-buffer. This is implementation of **PayloadRawInterface** (p. 286). Buffers are memory blocks logically placed one after another.

6.185.2 Constructor & Destructor Documentation

6.185.2.1 Arc::PayloadRaw::PayloadRaw (void) `[inline]`

List of handled buffers. Constructor. Created object contains no buffers.

6.185.2.2 virtual Arc::PayloadRaw::∼PayloadRaw (void) `[virtual]`

Destructor. Frees allocated buffers.

6.185.3 Member Function Documentation

6.185.3.1 virtual char∗ Arc::PayloadRaw::Buffer (unsigned int *num* = `0`) `[virtual]`

Returns pointer to num'th buffer

Implements **Arc::PayloadRawInterface** (p. 287).

6.185.3.2 virtual Size_t Arc::PayloadRaw::BufferPos (unsigned int *num* = 0) const [virtual]

Returns position of num'th buffer

Implements **Arc::PayloadRawInterface** (p. 287).

6.185.3.3 virtual Size_t Arc::PayloadRaw::BufferSize (unsigned int *num* = 0) const [virtual]

Returns length of num'th buffer

Implements **Arc::PayloadRawInterface** (p. 288).

6.185.3.4 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. 288).

6.185.3.5 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. 288).

6.185.3.6 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. 288).

6.185.3.7 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. 288).

6.185.3.8 virtual Size_t Arc::PayloadRaw::Size (void) const [virtual]

Returns logical size of whole structure.

Implements **Arc::PayloadRawInterface** (p. 288).

6.185.3.9 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. 289).

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

- PayloadRaw.h

6.186 Arc::PayloadRawBuf Struct Reference

Data Fields

- int **size**
- int **length**
- bool **allocated**

6.186.1 Field Documentation

6.186.1.1 bool Arc::PayloadRawBuf::allocated

size of used memory - size of buffer

6.186.1.2 int Arc::PayloadRawBuf::length

size of allocated memory

6.186.1.3 int Arc::PayloadRawBuf::size

pointer to buffer in memory

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

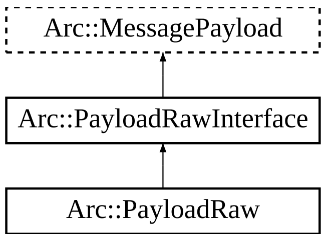
- PayloadRaw.h

6.187 Arc::PayloadRawInterface Class Reference

Random Access Payload for **Message** (p. 262) 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

6.187.1 Detailed Description

Random Access Payload for **Message** (p. 262) objects. This class is a virtual interface for managing **Message** (p. 262) 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.

6.187.2 Member Function Documentation

6.187.2.1 virtual char∗ Arc::PayloadRawInterface::Buffer (unsigned int *num*) `[pure virtual]`

Returns pointer to num'th buffer

Implemented in **Arc::PayloadRaw** (p. 284).

6.187.2.2 virtual Size_t Arc::PayloadRawInterface::BufferPos (unsigned int *num*) const `[pure virtual]`

Returns position of num'th buffer

Implemented in **Arc::PayloadRaw** (p. 285).

6.187.2.3 virtual Size_t Arc::PayloadRawInterface::BufferSize (unsigned int *num*) const `[pure virtual]`

Returns length of num'th buffer

Implemented in **Arc::PayloadRaw** (p. 285).

6.187.2.4 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. 285).

6.187.2.5 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. 285).

6.187.2.6 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. 285).

6.187.2.7 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. 285).

6.187.2.8 virtual Size_t Arc::PayloadRawInterface::Size (void) const `[pure virtual]`

Returns logical size of whole structure.

Implemented in **Arc::PayloadRaw** (p. 285).

6.187.2.9 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. 286).

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

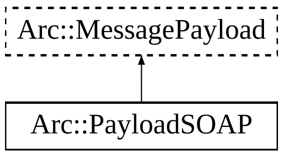
- PayloadRaw.h

6.188 Arc::PayloadSOAP Class Reference

Payload of **Message** (p. 262) 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)

6.188.1 Detailed Description

Payload of **Message** (p. 262) with SOAP content. This class combines **MessagePayload** (p. 271) with SOAPEnvelope to make it possible to pass SOAP messages through **MCC** (p. 252) chain.

6.188.2 Constructor & Destructor Documentation

6.188.2.1 Arc::PayloadSOAP::PayloadSOAP (const NS & *ns*, bool *fault* = `false`)

Constructor - creates new **Message** (p. 262) payload

6.188.2.2 Arc::PayloadSOAP::PayloadSOAP (const SOAPEnvelope & *soap*)

Constructor - creates **Message** (p. 262) payload from SOAP document. Provided SOAP document is copied to new object.

6.188.2.3 Arc::PayloadSOAP::PayloadSOAP (const MessagePayload & *source*)

Constructor - creates SOAP message from payload. **PayloadRawInterface** (p. 286) and derived classes are supported.

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

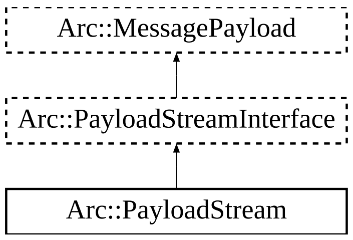
- PayloadSOAP.h

6.189 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_**

6.189.1 Detailed Description

POSIX handle as Payload. This is an implemetation of **PayloadStreamInterface** (p. 293) for generic POSIX handle.

6.189.2 Constructor & Destructor Documentation

6.189.2.1 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.

6.189.2.2 virtual Arc::PayloadStream::∼PayloadStream (void) `[inline, virtual]`

Destructor.

6.189.3 Member Function Documentation

6.189.3.1 virtual bool Arc::PayloadStream::Get (char ∗ *buf,* int & *size*) `[virtual]`

Extracts information from stream up to 'size' bytes. 'size' contains number of read bytes on exit. Returns true in case of success.

Implements **Arc::PayloadStreamInterface** (p. 294).

6.189.3.2 virtual bool Arc::PayloadStream::Get (std::string & *buf*) `[virtual]`

Read as many as possible (sane amount) of bytes into buf.

Implements **Arc::PayloadStreamInterface** (p. 294).

6.189.3.3 virtual std::string Arc::PayloadStream::Get (void) `[inline, virtual]`

Read as many as possible (sane amount) of bytes.

Implements **Arc::PayloadStreamInterface** (p. 295).

References Get().

Referenced by Get().

6.189.3.4 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. 293) if instance is meant to provide access to only part of underlying obejct.

Implements **Arc::PayloadStreamInterface** (p. 295).

6.189.3.5 virtual Arc::PayloadStream::operator bool (void) [inline, virtual]

Returns true if stream is valid.

Implements **Arc::PayloadStreamInterface** (p. 295).

References handle_.

6.189.3.6 virtual bool Arc::PayloadStream::operator! (void) [inline, virtual]

Returns true if stream is invalid.

Implements **Arc::PayloadStreamInterface** (p. 295).

References handle_.

6.189.3.7 virtual Size_t Arc::PayloadStream::Pos (void) const [inline, virtual]

Returns current position in stream if supported.

Implements **Arc::PayloadStreamInterface** (p. 295).

6.189.3.8 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. 295).

6.189.3.9 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. 295).

References Put().

Referenced by Put().

6.189.3.10 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. 296).

References Put().

Referenced by Put().

6.189.3.11 virtual Size_t Arc::PayloadStream::Size (void) const `[inline, virtual]`

Returns size of underlying object if supported.

Implements **Arc::PayloadStreamInterface** (p. 296).

6.189.3.12 virtual int Arc::PayloadStream::Timeout (void) const `[inline, virtual]`

Query (p. 316) current timeout for **Get()** (p. 291) and **Put()** (p. 292) operations.

Implements **Arc::PayloadStreamInterface** (p. 296).

6.189.3.13 virtual void Arc::PayloadStream::Timeout (int *to*) `[inline, virtual]`

Set current timeout for **Get()** (p. 291) and **Put()** (p. 292) operations.

Implements **Arc::PayloadStreamInterface** (p. 296).

6.189.4 Field Documentation

6.189.4.1 int Arc::PayloadStream::handle_ `[protected]`

Timeout for read/write operations

Referenced by operator bool(), and operator!().

6.189.4.2 bool Arc::PayloadStream::seekable_ `[protected]`

Handle for operations

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

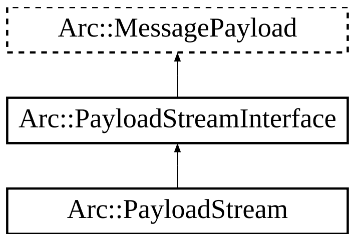
- PayloadStream.h

6.190 Arc::PayloadStreamInterface Class Reference

Stream-like Payload for **Message** (p. 262) 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

6.190.1 Detailed Description

Stream-like Payload for **Message** (p. 262) object. This class is a virtual interface for managing stream-like source and destination. It's supposed to be passed through **MCC** (p. 252) chain as payload of **Message** (p. 262). It must be treated by MCCs and Services as dynamic payload. This class is purely virtual.

6.190.2 Member Function Documentation

6.190.2.1 virtual bool Arc::PayloadStreamInterface::Get (char ∗ *buf,* int & *size*) `[pure virtual]`

Extracts information from stream up to 'size' bytes. 'size' contains number of read bytes on exit. Returns true in case of success.

Implemented in **Arc::PayloadStream** (p. 291).

6.190.2.2 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. 291).

6.190.2.3 virtual std::string Arc::PayloadStreamInterface::Get (void) `[pure virtual]`

Read as many as possible (sane amount) of bytes.

Implemented in **Arc::PayloadStream** (p. 291).

6.190.2.4 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. 296) if instance is meant to provide access to only part of underlying obejct.

Implemented in **Arc::PayloadStream** (p. 292).

6.190.2.5 virtual Arc::PayloadStreamInterface::operator bool (void) `[pure virtual]`

Returns true if stream is valid.

Implemented in **Arc::PayloadStream** (p. 292).

6.190.2.6 virtual bool Arc::PayloadStreamInterface::operator! (void) `[pure virtual]`

Returns true if stream is invalid.

Implemented in **Arc::PayloadStream** (p. 292).

6.190.2.7 virtual Size_t Arc::PayloadStreamInterface::Pos (void) const `[pure virtual]`

Returns current position in stream if supported.

Implemented in **Arc::PayloadStream** (p. 292).

6.190.2.8 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. 292).

6.190.2.9 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. 292).

6.190.2.10 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. 292).

6.190.2.11 virtual Size_t Arc::PayloadStreamInterface::Size (void) const `[pure virtual]`

Returns size of underlying object if supported.

Implemented in **Arc::PayloadStream** (p. 293).

6.190.2.12 virtual int Arc::PayloadStreamInterface::Timeout (void) const `[pure virtual]`

Query (p. 316) current timeout for **Get()** (p. 295) and **Put()** (p. 295) operations.

Implemented in **Arc::PayloadStream** (p. 293).

6.190.2.13 virtual void Arc::PayloadStreamInterface::Timeout (int *to*) `[pure virtual]`

Set current timeout for **Get()** (p. 295) and **Put()** (p. 295) operations.

Implemented in **Arc::PayloadStream** (p. 293).

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

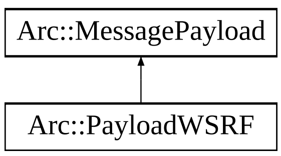
- PayloadStream.h

6.191 Arc::PayloadWSRF Class Reference

This class combines **MessagePayload** (p. 271) with **WSRF** (p. 441).

`#include <PayloadWSRF.h>`

Inheritance diagram for Arc::PayloadWSRF:



Public Member Functions

- **PayloadWSRF** (const SOAPEnvelope &soap)

- **PayloadWSRF** (**WSRF** &wsrp)
- **PayloadWSRF** (const **MessagePayload** &source)

6.191.1 Detailed Description

This class combines **MessagePayload** (p. 271) with **WSRF** (p. 441). It's intention is to make it possible to pass **WSRF** (p. 441) messages through **MCC** (p. 252) chain as one more Payload type.

6.191.2 Constructor & Destructor Documentation

6.191.2.1 Arc::PayloadWSRF::PayloadWSRF (const SOAPEnvelope & *soap*)

Constructor - creates **Message** (p. 262) payload from SOAP message. Returns invalid **WSRF** (p. 441) if SOAP does not represent WS-ResourceProperties

6.191.2.2 Arc::PayloadWSRF::PayloadWSRF (WSRF & *wsrp*)

Constructor - creates **Message** (p. 262) payload with acquired **WSRF** (p. 441) message. **WSRF** (p. 441) message will be destroyed by destructor of this object.

6.191.2.3 Arc::PayloadWSRF::PayloadWSRF (const MessagePayload & *source*)

Constructor - creates **WSRF** (p. 441) message from payload. All classes derived from SOAPEnvelope are supported.

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

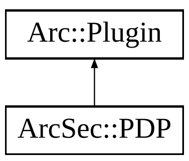
- PayloadWSRF.h

6.192 ArcSec::PDP Class Reference

Base class for **Policy** (p. 310) Decision Point plugins.

`#include <PDP.h>`

Inheritance diagram for ArcSec::PDP:



6.192.1 Detailed Description

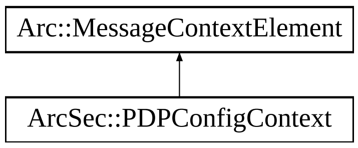
Base class for **Policy** (p. 310) Decision Point plugins. This virtual class defines method isPermitted() which processes security related information/attributes in Message and makes security decision - permit (true) or deny (false). Configuration of **PDP** (p. 297) is consumed during creation of instance through XML subtree fed to constructor.

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

- PDP.h

6.193 ArcSec::PDPConfigContext Class Reference

Inheritance diagram for ArcSec::PDPConfigContext:

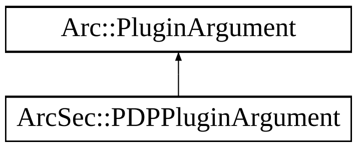


The documentation for this class was generated from the following file:

- PDP.h

6.194 ArcSec::PDPPluginArgument Class Reference

Inheritance diagram for ArcSec::PDPPluginArgument:



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

- PDP.h

6.195 Arc::Period Class Reference

Public Member Functions

- **Period** ()

- **Period** (time_t)
- **Period** (time_t seconds, uint32_t nanoseconds)
- **Period** (const std::string &, PeriodBase base=PeriodSeconds)
- **Period** & **operator=** (time_t)
- **Period** & **operator=** (const **Period** &)
- void **SetPeriod** (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

6.195.1 Constructor & Destructor Documentation

6.195.1.1 Arc::Period::Period ( )

Default constructor. The period is set to 0 length.

6.195.1.2 Arc::Period::Period (time_t)

Constructor that takes a time_t variable and stores it.

6.195.1.3 Arc::Period::Period (time_t *seconds*, uint32_t *nanoseconds*)

Constructor that takes seconds and nanoseconds and stores them.

6.195.1.4 Arc::Period::Period (const std::string & , PeriodBase *base* = `PeriodSeconds`)

Constructor that tries to convert a string.

6.195.2 Member Function Documentation

6.195.2.1 time_t Arc::Period::GetPeriod ( ) const

gets the period

6.195.2.2 const sigc::slot<const char∗>∗ Arc::Period::istr ( ) const

For use with **IString** (p. 214)

6.195.2.3 Arc::Period::operator std::string ( ) const

Returns a string representation of the period.

6.195.2.4 bool Arc::Period::operator!= (const Period &) const

Comparing two **Period** (p. 298) objects.

6.195.2.5 bool Arc::Period::operator< (const Period &) const

Comparing two **Period** (p. 298) objects.

6.195.2.6 bool Arc::Period::operator<= (const Period &) const

Comparing two **Period** (p. 298) objects.

6.195.2.7 Period& Arc::Period::operator= (time_t)

Assignment operator from a time_t.

6.195.2.8 Period& Arc::Period::operator= (const Period &)

Assignment operator from a **Period** (p. 298).

6.195.2.9 bool Arc::Period::operator== (const Period &) const

Comparing two **Period** (p. 298) objects.

6.195.2.10 bool Arc::Period::operator> (const Period &) const

Comparing two **Period** (p. 298) objects.

6.195.2.11 bool Arc::Period::operator>= (const Period &) const

Comparing two **Period** (p. 298) objects.

6.195.2.12 void Arc::Period::SetPeriod (time_t)

sets the period

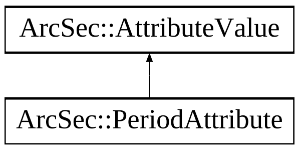
The documentation for this class was generated from the following file:

- DateTime.h

6.196 ArcSec::PeriodAttribute Class Reference

`#include <DateTimeAttribute.h>`

Inheritance diagram for ArcSec::PeriodAttribute:



Public Member Functions

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

6.196.1 Detailed Description

Formate: datetime"/"duration datetime"/"datetime duration"/"datetime

6.196.2 Member Function Documentation

6.196.2.1 virtual std::string ArcSec::PeriodAttribute::encode ( ) `[virtual]`

encode the value in a string format

Implements **ArcSec::AttributeValue** (p. 64).

6.196.2.2 virtual bool ArcSec::PeriodAttribute::equal (AttributeValue ∗ *value*, bool *check_id* = `true`) `[virtual]`

Evluate whether "this" equale to the parameter value

Implements **ArcSec::AttributeValue** (p. 64).

6.196.2.3 virtual std::string ArcSec::PeriodAttribute::getId ( ) `[inline, virtual]`

Get the AttributeId of the <Attribute>

Implements **ArcSec::AttributeValue** (p. 64).

6.196.2.4 virtual std::string ArcSec::PeriodAttribute::getType ( ) [inline, virtual]

Get the DataType of the <Attribute>

Implements **ArcSec::AttributeValue** (p. 65).

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

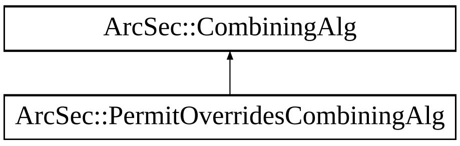
- DateTimeAttribute.h

6.197 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

6.197.1 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".

6.197.2 Member Function Documentation

6.197.2.1 virtual Result ArcSec::PermitOverridesCombiningAlg::combine (EvaluationCtx ∗ *ctx*, std::list< Policy ∗ > *policies*) [virtual]

If there is one policy which return positive evaluation result, then omit the other policies and return DECISION_PERMIT

Parameters

ctx	This object contains request information which will be used to evaluated against policy.
policies	This is a container which contains policy objects.

Returns

The combined result according to the algorithm.

Implements **ArcSec::CombiningAlg** (p. 85).

6.197.2.2 virtual const std::string& ArcSec::PermitOverridesCombiningAlg::getalgId (void) const `[inline, virtual]`

Get the identifier

Implements **ArcSec::CombiningAlg** (p. 85).

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

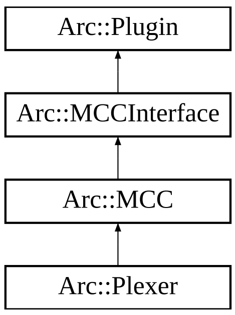
- PermitOverridesAlg.h

6.198 Arc::Plexer Class Reference

The **Plexer** (p. 303) 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**

6.198.1 Detailed Description

The **Plexer** (p. 303) class, used for routing messages to services. This is the **Plexer** (p. 303) class. Its purpose is to route incoming messages to appropriate Services and **MCC** (p. 252) chains.

6.198.2 Constructor & Destructor Documentation

6.198.2.1 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.

6.198.2.2 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.

6.198.3 Member Function Documentation

6.198.3.1 virtual void Arc::Plexer::Next (MCCInterface ∗ *next*, const std::string & *label*) `[virtual]`

Add reference to next **MCC** (p. 252) in chain.

This method is called by **Loader** (p. 238) for every potentially labeled link to next component which implements **MCCInterface** (p. 258). If next is set NULL corresponding link is removed.

Reimplemented from **Arc::MCC** (p. 254).

6.198.3.2 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. 254).

6.198.4 Field Documentation

6.198.4.1 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. 255).

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

- Plexer.h

6.199 Arc::PlexerEntry Class Reference

A pair of label (regex) and pointer to **MCC** (p. 252).

`#include <Plexer.h>`

6.199.1 Detailed Description

A pair of label (regex) and pointer to **MCC** (p. 252). 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:

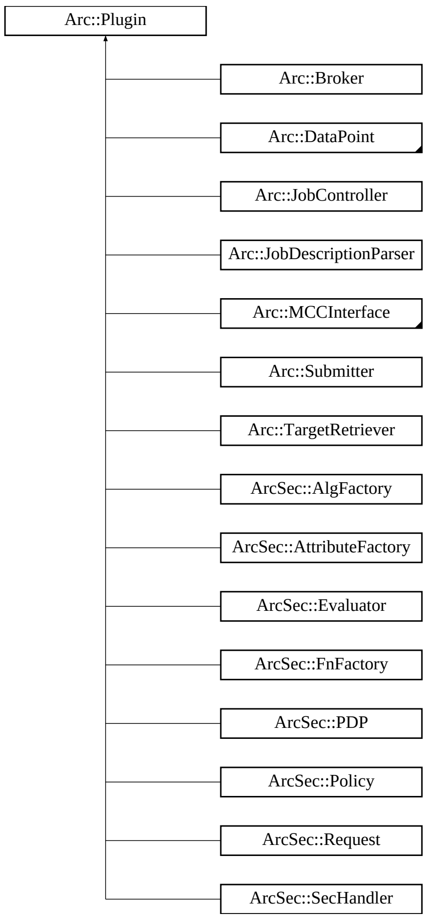
- Plexer.h

6.200 Arc::Plugin Class Reference

Base class for loadable ARC components.

`#include <Plugin.h>`

Inheritance diagram for Arc::Plugin:



6.200.1 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:

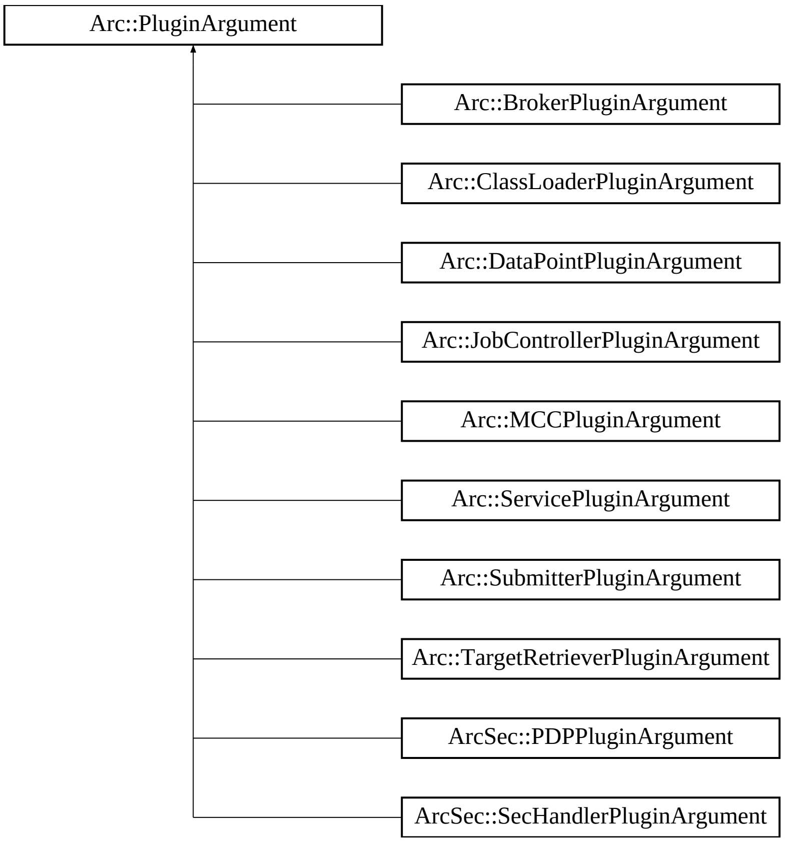
- Plugin.h

6.201 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)

6.201.1 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.

6.201.2 Member Function Documentation

6.201.2.1 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.

6.201.2.2 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

6.202 Arc::PluginDesc Class Reference

Description of plugin.

```
#include <Plugin.h>
```

6.202.1 Detailed Description

Description of plugin. This class is used for reports

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

- Plugin.h

6.203 Arc::PluginDescriptor Struct Reference

Description of ARC lodable component.

```
#include <Plugin.h>
```

6.203.1 Detailed Description

Description of ARC lodable component.

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

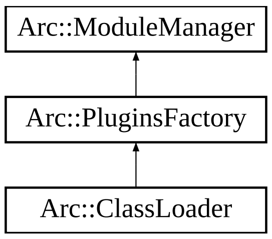
- Plugin.h

6.204 Arc::PluginsFactory Class Reference

Generic ARC plugins loader.

```
#include <Plugin.h>
```

Inheritance diagram for Arc::PluginsFactory:



Public Member Functions

- **PluginsFactory** (**XMLNode** cfg)
- void **TryLoad** (bool v=true)
- bool **load** (const std::string &name)
- bool **scan** (const std::string &name, **ModuleDesc** &desc)
- void **report** (std::list< **ModuleDesc** > &descs)

Static Public Member Functions

- static void **FilterByKind** (const std::string &kind, std::list< **ModuleDesc** > &descs)

6.204.1 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.

6.204.2 Constructor & Destructor Documentation

6.204.2.1 Arc::PluginsFactory::PluginsFactory (**XMLNode** *cfg*)

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

6.204.3 Member Function Documentation

6.204.3.1 static void Arc::PluginsFactory::FilterByKind (const std::string & *kind*, std::list< **ModuleDesc** > & *descs*) `[static]`

Filter list of modules by kind.

6.204.3.2 bool Arc::PluginsFactory::load (const std::string & *name*)

These methods load module named lib'name' and check if it contains ARC plugin(s) of specified 'kind' and 'name'. If there are no specified plugins or module does not contain any ARC plugins it is unloaded. All loaded plugins are also registered in internal list of this instance of **PluginsFactory** (p. 308) class. Returns true if any plugin was loaded.

6.204.3.3 void Arc::PluginsFactory::report (std::list< ModuleDesc > & *descs*)

Provides information about currently loaded modules and plugins.

6.204.3.4 bool Arc::PluginsFactory::scan (const std::string & *name*, ModuleDesc & *desc*)

Collect information about plugins stored in module(s) with specified names. Returns true if any of specified modules has plugins.

6.204.3.5 void Arc::PluginsFactory::TryLoad (bool *v* = `true`) `[inline]`

These methods load module 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. 308) class. If search is set to false then no attempt is made to find plugins in loadable modules. Only plugins already loaded with previous calls to get_instance() and load() are checked. Returns created instance or NULL if failed. Specifies if loadable module may be loaded while looking for analyzing its content. If set to false only ∗.apd files are checked. Modules without corresponding ∗.apd will be ignored. Default is true;

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

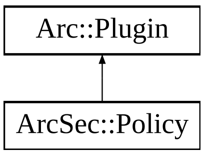
- Plugin.h

6.205 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

6.205.1 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.

6.205.2 Constructor & Destructor Documentation

6.205.2.1 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.

6.205.2.2 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. 174) which includes the factory objects for combining algorithm and function

6.205.3 Member Function Documentation

6.205.3.1 virtual void ArcSec::Policy::addPolicy (Policy ∗ *pl*) `[inline, virtual]`

Add a policy element to into "this" object

6.205.3.2 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>

6.205.3.3 virtual std::string ArcSec::Policy::getEffect ( ) const `[pure virtual]`

Get the "Effect" attribute

6.205.3.4 virtual const char∗ ArcSec::Policy::getEvalName ( ) const `[pure virtual]`

Get the name of **Evaluator** (p. 171) which can evaluate this policy

6.205.3.5 virtual EvalResult& ArcSec::Policy::getEvalResult ( ) `[pure virtual]`

Get eveluation result

6.205.3.6 virtual const char∗ ArcSec::Policy::getName ( ) const `[pure virtual]`

Get the name of this policy

6.205.3.7 virtual void ArcSec::Policy::make_policy ( ) `[inline, virtual]`

Parse XMLNode, and construct the low-level Rule object

6.205.3.8 virtual void ArcSec::Policy::setEvalResult (EvalResult & *res*) `[pure virtual]`

Set eveluation result

6.205.3.9 virtual void ArcSec::Policy::setEvaluatorContext (**EvaluatorContext** ∗) `[inline, virtual]`

Set **Evaluator** (p. 171) Context for the usage in creating low-level policy object

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

- Policy.h

6.206 ArcSec::PolicyStore::PolicyElement Class Reference

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

- PolicyStore.h

6.207 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)

6.207.1 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.).

6.207.2 Member Function Documentation

6.207.2.1 virtual **Policy**∗ ArcSec::PolicyParser::parsePolicy (const **Source** & *source*, std::string *policyclassname*, **EvaluatorContext** ∗ *ctx*) `[virtual]`

Parse policy

Parameters

source	location of the policy
policyclass-name	name of the policy for ClassLoader
ctx	**EvaluatorContext** (p. 174) which includes the ∗∗Factory

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

- PolicyParser.h

6.208 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)

6.208.1 Detailed Description

Storage place for policy objects.

6.208.2 Constructor & Destructor Documentation

6.208.2.1 ArcSec::PolicyStore::PolicyStore (const std::string & *alg,* const std::string & *policyclassname,* EvaluatorContext ∗ *ctx*)

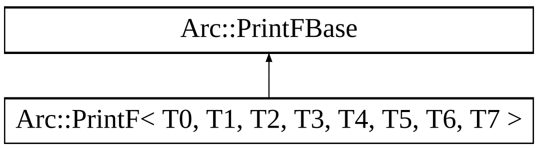
Creates policy store with specified combing algorithm (alg - not used yet), policy name (policyclassname) and context (ctx)

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

- PolicyStore.h

6.209 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 >:



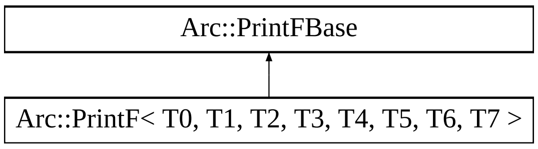
template<class T0 = int, class T1 = int, class T2 = int, class T3 = int, class T4 = int, class T5 = int, class T6 = int, class T7 = int> class Arc::PrintF< T0, T1, T2, T3, T4, T5, T6, T7 >

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

- IString.h

6.210 Arc::PrintFBase Class Reference

Inheritance diagram for Arc::PrintFBase:

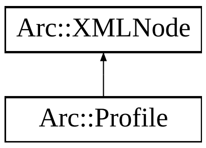


The documentation for this class was generated from the following file:

- IString.h

6.211 Arc::Profile Class Reference

Inheritance diagram for Arc::Profile:



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

- Profile.h

6.212 ArcCredential::PROXYCERTINFO_st Struct Reference

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

- Proxycertinfo.h

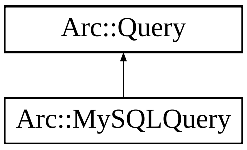
6.213 ArcCredential::PROXYPOLICY_st Struct Reference

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

- Proxycertinfo.h

6.214 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

6.214.1 Constructor & Destructor Documentation

6.214.1.1 Arc::Query::Query ( ) `[inline]`

Default constructor

6.214.1.2 Arc::Query::Query (Database ∗ *db*) `[inline]`

Constructor

Parameters

db	The database object which will be used by **Query** (p. 316) class to get the database connection

6.214.1.3 virtual Arc::Query::∼Query ( ) `[inline, virtual]`

Deconstructor

6.214.2 Member Function Documentation

6.214.2.1 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. 277).

6.214.2.2 virtual bool Arc::Query::get_array (std::string & *sqlstr,* QueryArrayResult & *result,* std::vector< std::string > & *arguments*) `[pure virtual]`

Query (p. 316) 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. 277).

6.214.2.3 virtual int Arc::Query::get_num_colums ( ) `[pure virtual]`

Get the colum number in the query result

Implemented in **Arc::MySQLQuery** (p. 277).

6.214.2.4 virtual int Arc::Query::get_num_rows ( ) [pure virtual]

Get the row number in the query result

Implemented in **Arc::MySQLQuery** (p. 277).

6.214.2.5 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. 278).

6.214.2.6 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. 278).

6.214.2.7 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. 278).

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

- DBInterface.h

6.215 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

6.216 Arc::Register_Info_Type Struct Reference

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

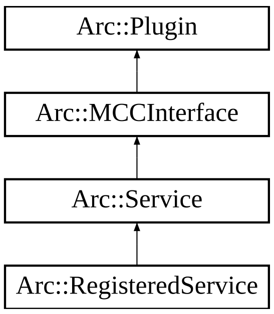
- InfoRegister.h

6.217 Arc::RegisteredService Class Reference

RegisteredService (p. 319) - extension of **Service** (p. 341) performing self-registration.

```
#include <RegisteredService.h>
```

Inheritance diagram for Arc::RegisteredService:



Public Member Functions

- **RegisteredService** (**Config** ∗)

6.217.1 Detailed Description

RegisteredService (p. 319) - extension of **Service** (p. 341) performing self-registration.

6.217.2 Constructor & Destructor Documentation

6.217.2.1 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

6.218 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

6.218.1 Detailed Description

A regular expression class. This class is a wrapper around the functions provided in regex.h.

6.218.2 Member Function Documentation

6.218.2.1 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 first entry in matched is the string that matched the regex, and the following entries are parenthesised elements of the regex

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

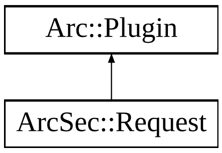
- ArcRegex.h

6.219 ArcSec::Request Class Reference

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

`#include <Request.h>`

Inheritance diagram for ArcSec::Request:



Public Member Functions

- virtual ReqItemList **getRequestItems** () const
- virtual void **setRequestItems** (ReqItemList)
- virtual void **addRequestItem** (**Attrs** &, **Attrs** &, **Attrs** &, **Attrs** &)
- virtual void **setAttributeFactory** (**AttributeFactory** ∗attributefactory)=0
- virtual void **make_request** ()=0
- virtual const char ∗ **getEvalName** () const =0
- virtual const char ∗ **getName** () const =0
- **Request** ()
- **Request** (const **Source** &)

6.219.1 Detailed Description

Base class/Interface for request, includes a container for RequestItems and some operations. A **Request** (p. 321) object can has a few <subjects, actions, objects> tuples, i.e. **RequestItem** (p. 323) The **Request** (p. 321) 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. 321) name="arc.request" /> <......> </pdp:PDPConfig> </Service>

There can be different types of subclass which inherit **Request** (p. 321), such like XACMLRequest, ArcRequest, GACLRequest

6.219.2 Constructor & Destructor Documentation

6.219.2.1 ArcSec::Request::Request ( ) `[inline]`

Default constructor

6.219.2.2 ArcSec::Request::Request (const **Source** &) `[inline]`

Constructor: Parse request information from a xml stucture in memory

6.219.3 Member Function Documentation

6.219.3.1 virtual void ArcSec::Request::addRequestItem (**Attrs** &, **Attrs** &, **Attrs** &, **Attrs** &) `[inline, virtual]`

Add request tuple from non-XMLNode

6.219.3.2 virtual const char∗ ArcSec::Request::getEvalName ( ) const `[pure virtual]`

Get the name of corresponding evaulator

6.219.3.3 virtual const char∗ ArcSec::Request::getName ( ) const `[pure virtual]`

Get the name of this request

6.219.3.4 virtual ReqItemList ArcSec::Request::getRequestItems ( ) const `[inline, virtual]`

Get all the **RequestItem** (p. 323) inside **RequestItem** (p. 323) container

6.219.3.5 virtual void ArcSec::Request::make_request ( ) `[pure virtual]`

Create the objects included in **Request** (p. 321) according to the node attached to the **Request** (p. 321) object

6.219.3.6 virtual void ArcSec::Request::setAttributeFactory (**AttributeFactory** ∗ *attributefactory*) `[pure virtual]`

Set the attribute factory for the usage of **Request** (p. 321)

6.219.3.7 virtual void ArcSec::Request::setRequestItems (ReqItemList) [inline, virtual]

Set the content of the container

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

- Request.h

6.220 ArcSec::RequestAttribute Class Reference

Wrapper which includes **AttributeValue** (p. 63) 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** &)

6.220.1 Detailed Description

Wrapper which includes **AttributeValue** (p. 63) object which is generated according to date type of one spefic node in Request.xml.

6.220.2 Constructor & Destructor Documentation

6.220.2.1 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>

6.220.3 Member Function Documentation

6.220.3.1 RequestAttribute& ArcSec::RequestAttribute::duplicate (RequestAttribute &)

Duplicate the parameter into "this"

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

- RequestAttribute.h

6.221 ArcSec::RequestItem Class Reference

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

`#include <RequestItem.h>`

Public Member Functions

- **RequestItem** (**Arc::XMLNode** &, **AttributeFactory** ∗)

6.221.1 Detailed Description

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

6.221.2 Constructor & Destructor Documentation

6.221.2.1 ArcSec::RequestItem::RequestItem (**Arc::XMLNode** & , **AttributeFactory** ∗) `[inline]`

Constructor

Parameters

node	The XMLNode structure of the request item
attributefactory	The **AttributeFactory** (p. 58) which will be used to generate **RequestAttribute** (p. 323)

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

- RequestItem.h

6.222 ArcSec::RequestTuple Class Reference

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

- EvaluationCtx.h

6.223 Arc::ResourceSlotType Class Reference

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

- JobDescription.h

6.224 Arc::ResourcesType Class Reference

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

- JobDescription.h

6.225 ArcSec::Response Class Reference

Container for the evaluation results.

```
#include <Response.h>
```

6.225.1 Detailed Description

Container for the evaluation results.

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

- Response.h

6.226 ArcSec::ResponseItem Class Reference

Evaluation result concerning one **RequestTuple** (p. 324).

```
#include <Response.h>
```

6.226.1 Detailed Description

Evaluation result concerning one **RequestTuple** (p. 324). Include the **RequestTuple** (p. 324), 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

6.227 ArcSec::ResponseList Class Reference

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

- Response.h

6.228 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)

Static Public Member Functions

- static void **AfterFork** (void)

6.228.1 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.

6.228.2 Constructor & Destructor Documentation

6.228.2.1 Arc::Run::Run (const std::string & *cmdline*)

Constructor preapres object to run cmdline

6.228.2.2 Arc::Run::Run (const std::list< std::string > & *argv*)

Constructor preapres object to run executable and arguments specified in argv

6.228.2.3 Arc::Run::∼Run (void)

Destructor kills running executable and releases associated resources

6.228.3 Member Function Documentation

6.228.3.1 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.

6.228.3.2 static void Arc::Run::AfterFork (void) `[static]`

Call this method after fork() in child cporocess. It will reinitialize internal structures for new environment. Do not call it in any other case than defined.

6.228.3.3 void Arc::Run::AssignStderr (std::string & *str*)

Associate stderr handle of executable with string. This method must be called before **Start()** (p. 329). str object must be valid as long as this object exists.

6.228.3.4 void Arc::Run::AssignStdin (std::string & *str*)

Associate stdin handle of executable with string. This method must be called before **Start()** (p. 329). str object must be valid as long as this object exists.

6.228.3.5 void Arc::Run::AssignStdout (std::string & *str*)

Associate stdout handle of executable with string. This method must be called before **Start()** (p. 329). str object must be valid as long as this object exists.

6.228.3.6 void Arc::Run::AssignWorkingDirectory (std::string & *wd*) `[inline]`

Assign working direcotry of the running process

6.228.3.7 void Arc::Run::CloseStderr (void)

Closes pipe associated with stderr handle

6.228.3.8 void Arc::Run::CloseStdin (void)

Closes pipe associated with stdin handle

6.228.3.9 void Arc::Run::CloseStdout (void)

Closes pipe associated with stdout handle

6.228.3.10 void Arc::Run::KeepStderr (bool *keep* = `true`)

Keep stderr same as parent's if keep = true

6.228.3.11 void Arc::Run::KeepStdin (bool *keep* = `true`)

Keep stdin same as parent's if keep = true

6.228.3.12 void Arc::Run::KeepStdout (bool *keep* = `true`)

Keep stdout same as parent's if keep = true

6.228.3.13 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

6.228.3.14 Arc::Run::operator bool (void) `[inline]`

Returns true if object is valid

6.228.3.15 bool Arc::Run::operator! (void) `[inline]`

Returns true if object is invalid

6.228.3.16 int Arc::Run::ReadStderr (int *timeout*, char * *buf*, int *size*)

Read from stderr handle of running executable. Parameter timeout specifies upper limit for which method will block in milliseconds. Negative means infinite. This method may be used while stderr is directed to string. But result is unpredictable. Returns number of read bytes.

6.228.3.17 int Arc::Run::ReadStdout (int *timeout*, char ∗ *buf*, int *size*)

Read from stdout handle of running executable. Parameter timeout specifies upper limit for which method will block in milliseconds. Negative means infinite. This method may be used while stdout is directed to string. But result is unpredictable. Returns number of read bytes.

6.228.3.18 int Arc::Run::Result (void) [inline]

Returns exit code of execution.

6.228.3.19 bool Arc::Run::Running (void)

Return true if execution is going on.

6.228.3.20 bool Arc::Run::Start (void)

Starts running executable. This method may be called only once.

6.228.3.21 bool Arc::Run::Wait (int *timeout*)

Wait till execution finished or till timeout seconds expires. Returns true if execution is complete.

6.228.3.22 bool Arc::Run::Wait (void)

Wait till execution finished

6.228.3.23 int Arc::Run::WriteStdin (int *timeout*, const char ∗ *buf*, int *size*)

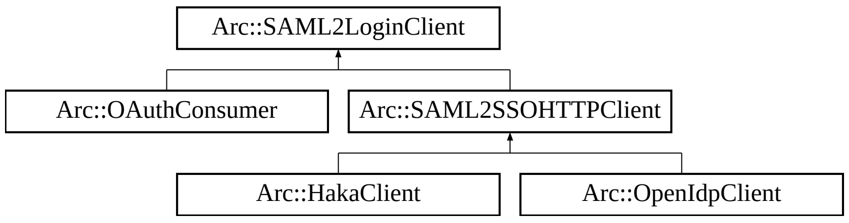
Write to stdin handle of running executable. Parameter timeout specifies upper limit for which method will block in milliseconds. Negative means infinite. This method may be used while stdin is directed to string. But result is unpredictable. Returns number of written bytes.

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

- Run.h

6.229 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** ()

6.229.1 Constructor & Destructor Documentation

6.229.1.1 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.

6.229.2 Member Function Documentation

6.229.2.1 MCC_Status Arc::SAML2LoginClient::findSimpleSAMLInstallation ( )

find the location of the simplesamlphp installation on the SP side Will be stored in (∗sso_pages)[SimpleSAML]

6.229.2.2 virtual MCC_Status Arc::SAML2LoginClient::processLogin (const std::string *username* = "", const std::string *password* = "") [pure virtual]

Base interface for all login procedures

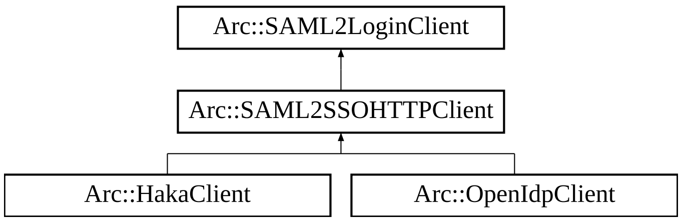
Implemented in **Arc::OAuthConsumer** (p. 280), and **Arc::SAML2SSOHTTPClient** (p. 332).

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

- SAML2LoginClient.h

6.230 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

6.230.1 Member Function Documentation

6.230.1.1 MCC_Status Arc::SAML2SSOHTTPClient::approveCSR (const std::string *approve_page*) `[virtual]`

Simulate click on the approve cert signing request link

Implements **Arc::SAML2LoginClient** (p. 329).

6.230.1.2 MCC_Status Arc::SAML2SSOHTTPClient::parseDN (std::string ∗ *dn*) `[virtual]`

Parse the used DN from the Confusa about_you page

Implements **Arc::SAML2LoginClient** (p. 329).

6.230.1.3 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. 197), and **Arc::OpenIdpClient** (p. 281).

6.230.1.4 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. 197), and **Arc::OpenIdpClient** (p. 281).

6.230.1.5 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. 197), and **Arc::OpenIdpClient** (p. 281).

6.230.1.6 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. 330).

6.230.1.7 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. 329).

6.230.1.8 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. 329).

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

- SAML2LoginClient.h

6.231 Arc::SAMLToken Class Reference

Class for manipulating SAML Token **Profile** (p. 315).

```
#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)

6.231.1 Detailed Description

Class for manipulating SAML Token **Profile** (p. 315). This class is for generating/-consuming SAML Token profile. See WS-Security SAML Token **Profile** (p. 315) 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. 315) 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. 315) 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).

6.231.2 Member Enumeration Documentation

6.231.2.1 enum Arc::SAMLToken::SAMLVersion

Since the specfication SAMLVersion is for distinguishing two types of saml version. It is used as the parameter of constructor.

6.231.3 Constructor & Destructor Documentation

6.231.3.1 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. 332) object will be used for authentication.

Parameters

soap	The SOAP message which contains the **SAMLToken** (p. 332) in the soap header

6.231.3.2 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.

6.231.3.3 Arc::SAMLToken::∼SAMLToken (void)

Deconstructor. Nothing to be done except finalizing the xmlsec library.

6.231.4 Member Function Documentation

6.231.4.1 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. 334) 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

6.231.4.2 bool Arc::SAMLToken::Authenticate (void)

Check signature by using the cert information in soap message

6.231.4.3 Arc::SAMLToken::operator bool (void)

Returns true of constructor succeeded

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

- SAMLToken.h

6.232 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

6.233 Arc::ScalableTime< int > Class Template Reference

template<> class Arc::ScalableTime< int >

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

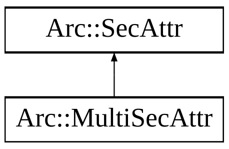
- JobDescription.h

6.234 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**

6.234.1 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.

6.234.2 Member Function Documentation

6.234.2.1 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.

6.234.2.2 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. 274).

6.234.2.3 virtual bool Arc::SecAttr::Import (SecAttrFormat *format,* const std::string & *val*) `[virtual]`

Fills internal structure from external object of specified format. Returns false if failed to do. The usage pattern for this method is not defined and it is provided only to make class symmetric. Hence it's implementation is not required yet.

6.234.2.4 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. 274).

6.234.2.5 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.

6.234.2.6 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

6.235 Arc::SecAttrFormat Class Reference

Export/import format.

`#include <SecAttr.h>`

6.235.1 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:

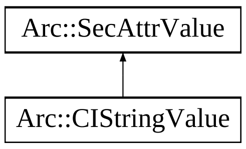
- SecAttr.h

6.236 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** ()

6.236.1 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.

6.236.2 Member Function Documentation

6.236.2.1 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. 77).

6.236.2.2 bool Arc::SecAttrValue::operator!= (SecAttrValue & *b*)

This is a convenience function to allow the usage of "not equal" conditions and need not be overridden.

6.236.2.3 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:

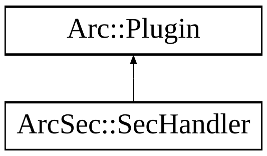
- SecAttrValue.h

6.237 ArcSec::SecHandler Class Reference

Base class for simple security handling plugins.

`#include <SecHandler.h>`

Inheritance diagram for ArcSec::SecHandler:



6.237.1 Detailed Description

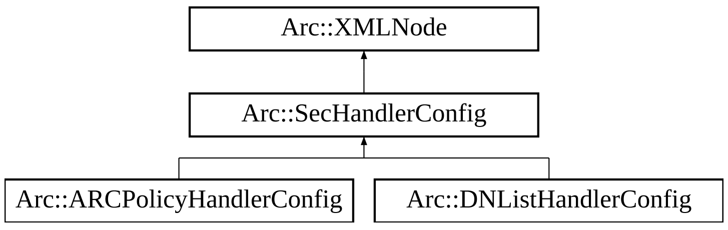
Base class for simple security handling plugins. This virtual class defines method Handle() which processes security related information/attributes in Message and optionally makes security decision. Instances of such classes are normally arranged in chains abd are called on incoming and outgoing messages 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. 339) 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

6.238 Arc::SecHandlerConfig Class Reference

Inheritance diagram for Arc::SecHandlerConfig:



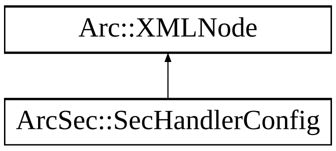
The documentation for this class was generated from the following file:

- ClientInterface.h

6.239 ArcSec::SecHandlerConfig Class Reference

`#include <SecHandler.h>`

Inheritance diagram for ArcSec::SecHandlerConfig:



6.239.1 Detailed Description

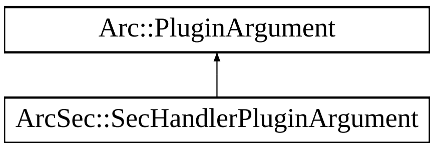
Helper class to create **Security** (p. 341) Handler configuration

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

- SecHandler.h

6.240 ArcSec::SecHandlerPluginArgument Class Reference

Inheritance diagram for ArcSec::SecHandlerPluginArgument:



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

- SecHandler.h

6.241 ArcSec::Security Class Reference

Common stuff used by security related slasses.

`#include <Security.h>`

6.241.1 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:

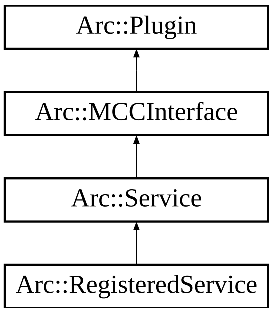
- Security.h

6.242 Arc::Service Class Reference

Service (p. 341) - last component in a **Message** (p. 262) 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="") const

Protected Attributes

- std::map< std::string, std::list< **ArcSec::SecHandler** ∗ > > **sechandlers_**

Static Protected Attributes

- static **Logger** **logger**

6.242.1 Detailed Description

Service (p. 341) - last component in a **Message** (p. 262) Chain. This class which defines interface and common functionality for every **Service** (p. 341) plugin. Interface is made of method **process()** (p. 259) which is called by **Plexer** (p. 303) or **MCC** (p. 252) class. There is one **Service** (p. 341) object created for every service description processed by **Loader** (p. 238) class objects. Classes derived from **Service** (p. 341) class must implement **process()** (p. 259) method of **MCCInterface** (p. 258). It is up to developer how internal state of service is stored and communicated to other services and external utilites. **Service** (p. 341) 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. 252) it must accept and generate messages with **PayloadSOAP** (p. 289) payload. Method **process()** (p. 259) of class derived from **Service** (p. 341) 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 .

6.242.2 Constructor & Destructor Documentation

6.242.2.1 Arc::Service::Service (Config *)

Example contructor - Server takes at least it's configuration subtree

6.242.3 Member Function Documentation

6.242.3.1 virtual void Arc::Service::AddSecHandler (Config * *cfg*, ArcSec::SecHandler * *sechandler*, const std::string & *label* = "") `[virtual]`

Add security components/handlers to this **MCC** (p. 252). For more information please see description of **MCC::AddSecHandler** (p. 254)

6.242.3.2 virtual std::string Arc::Service::getID ( ) `[inline, virtual]`

Service (p. 341) may implement own service identitifer gathering method. This method return identifier of service which is used for registering it Information Services.

6.242.3.3 bool Arc::Service::ProcessSecHandlers (Message & *message*, const std::string & *label* = "") const `[protected]`

Executes security handlers of specified queue. For more information please see description of **MCC::ProcessSecHandlers** (p. 254)

6.242.3.4 virtual bool Arc::Service::RegistrationCollector (XMLNode & *doc*) [virtual]

Service (p. 341) 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.

6.242.4 Field Documentation

6.242.4.1 Logger Arc::Service::logger [static, protected]

Logger (p. 243) object used to print messages generated by this class.

6.242.4.2 std::map<std::string,std::list<ArcSec::SecHandler*> > Arc::Service::sechandlers_ [protected]

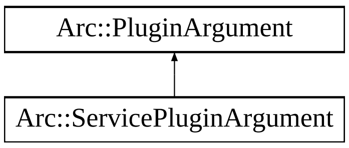
Set of labeled authentication and authorization handlers. **MCC** (p. 252) calls sequence of handlers at specific point depending on associated identifier. in most aces those are "in" and "out" for incoming and outgoing messages correspondingly.

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

- Service.h

6.243 Arc::ServicePluginArgument Class Reference

Inheritance diagram for Arc::ServicePluginArgument:



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

- Service.h

6.244 Arc::SharedMutex Class Reference

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

- Thread.h

6.245 Arc::SimpleCondition Class Reference

Simple triggered condition.

`#include <Thread.h>`

Public Member Functions

- void **lock** (void)
- void **unlock** (void)
- void **signal** (void)
- void **signal_nonblock** (void)
- void **broadcast** (void)
- void **wait** (void)
- void **wait_nonblock** (void)
- bool **wait** (int t)
- void **reset** (void)

6.245.1 Detailed Description

Simple triggered condition. Provides condition and semaphor objects in one element.

6.245.2 Member Function Documentation

6.245.2.1 void Arc::SimpleCondition::broadcast (void) `[inline]`

Signal about condition to all waiting threads

6.245.2.2 void Arc::SimpleCondition::lock (void) `[inline]`

Acquire semaphor

6.245.2.3 void Arc::SimpleCondition::reset (void) `[inline]`

Reset object to initial state

6.245.2.4 void Arc::SimpleCondition::signal (void) `[inline]`

Signal about condition

6.245.2.5 void Arc::SimpleCondition::signal_nonblock (void) `[inline]`

Signal about condition without using semaphor

6.245.2.6 void Arc::SimpleCondition::unlock (void) `[inline]`

Release semaphor

6.245.2.7 bool Arc::SimpleCondition::wait (int *t*) `[inline]`

Wait for condition no longer than t milliseconds

6.245.2.8 void Arc::SimpleCondition::wait (void) `[inline]`

Wait for condition

6.245.2.9 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

6.246 Arc::SimpleCounter Class Reference

Public Member Functions

- bool **wait** (int t)

6.246.1 Member Function Documentation

6.246.1.1 bool Arc::SimpleCounter::wait (int *t*) `[inline]`

Wait for condition no longer than t milliseconds

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

- Thread.h

6.247 Arc::SOAPMessage Class Reference

Message (p. 262) 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)

6.247.1 Detailed Description

Message (p. 262) restricted to SOAP payload. This is a special **Message** (p. 262) 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. 262) but can carry only SOAP content.

6.247.2 Constructor & Destructor Documentation

6.247.2.1 Arc::SOAPMessage::SOAPMessage (void) `[inline]`

Dummy constructor

6.247.2.2 Arc::SOAPMessage::SOAPMessage (long *msg_ptr_addr*)

Copy constructor. Used by language bindigs

6.247.2.3 Arc::SOAPMessage::SOAPMessage (Message & *msg*)

Copy constructor. Ensures shallow copy.

6.247.2.4 Arc::SOAPMessage::~SOAPMessage (void)

Destructor does not affect refered objects

6.247.3 Member Function Documentation

6.247.3.1 MessageAttributes* Arc::SOAPMessage::Attributes (void) `[inline]`

Returns a pointer to the current attributes object or NULL if no attributes object has been assigned.

6.247.3.2 SOAPEnvelope∗ Arc::SOAPMessage::Payload (void)

Returns pointer to current payload or NULL if no payload assigned.

6.247.3.3 void Arc::SOAPMessage::Payload (SOAPEnvelope ∗ *new_payload*)

Replace payload with a COPY of new one

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

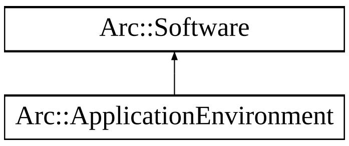
- SOAPMessage.h

6.248 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)

6.248.1 Detailed Description

Used to represent software (names and version) and comparison. The **Software** (p. 347) 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. 356)) are fulfilled, by using the comparability of the class.

Internally the **Software** (p. 347) 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.

6.248.2 Member Typedef Documentation

6.248.2.1 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. 353),

operator!= (p. 352),
operator> (p. 354),
operator< (p. 353),
operator>= (p. 354),
operator<= (p. 353),
ComparisonOperatorEnum (p. 349).

6.248.3 Member Enumeration Documentation

6.248.3.1 enum Arc::Software::ComparisonOperatorEnum

Comparison operator enum.

The **ComparisonOperatorEnum** (p. 349) enumeration is a 1-1 correspondance between the defined comparison method operators (**Software::ComparisonOperator** (p. 349)), and can be used in circumstances where method pointers are not supported.

Enumerator:

NOTEQUAL	see **operator!=** (p. 352)
EQUAL	see **operator==** (p. 353)
GREATERTHAN	see **operator>** (p. 354)
LESSTHAN	see **operator<** (p. 353)
GREATERTHANOREQUAL	see **operator>=** (p. 354)
LESSTHANOREQUAL	see **operator<=** (p. 353)

6.248.4 Constructor & Destructor Documentation

6.248.4.1 Arc::Software::Software ( ) `[inline]`

Dummy constructor.

This constructor creates a empty object.

6.248.4.2 Arc::Software::Software (const std::string & *name_version*)

Create a **Software** (p. 347) object.

Create a **Software** (p. 347) 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.

6.248.4.3 Arc::Software::Software (const std::string & *name*, const std::string & *version*)

Create a **Software** (p. 347) object.

Create a **Software** (p. 347) 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.

6.248.4.4 Arc::Software::Software (const std::string & *family*, const std::string & *name*, const std::string & *version*)

Create a **Software** (p. 347) object.

Create a **Software** (p. 347) 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.

6.248.5 Member Function Documentation

6.248.5.1 static ComparisonOperator Arc::Software::convert (const ComparisonOperatorEnum & *co*) `[static]`

Convert a **ComparisonOperatorEnum** (p. 349) value to a comparison method pointer.

The passed **ComparisonOperatorEnum** (p. 349) will be converted to a comparison method pointer defined by the **Software::ComparisonOperator** (p. 349) 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. 349) value.

Returns

A method pointer to a comparison method is returned.

6.248.5.2 bool Arc::Software::empty ( ) const `[inline]`

Indicates whether the object is empty.

Returns

`true` if the name of this object is empty, otherwise `false`.

6.248.5.3 const std::string& Arc::Software::getFamily ( ) const [inline]

Get family.

Returns

The family the represented software belongs to is returned.

6.248.5.4 const std::string& Arc::Software::getName ( ) const [inline]

Get name.

Returns

The name of the represented software is returned.

6.248.5.5 const std::string& Arc::Software::getVersion ( ) const [inline]

Get version.

Returns

The version of the represented software is returned.

6.248.5.6 Arc::Software::operator std::string (void) const [inline]

Cast to string.

This casting operator behaves exactly as **operator()()** (p. 352) does. The cast is used like (std::string) <software-object>.

See also

operator()() (p. 352).

References operator()().

6.248.5.7 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. 347) objects are of different versions but share the same name and family. So it should not be used to test if two **Software** (p. 347) objects differ in either name, version or family. Two **Software** (p. 347) 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. 347) object.

Returns

`true` when the two objects are inequal, otherwise `false`.

6.248.5.8 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().

6.248.5.9 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. 354)) 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. 354).

6.248.5.10 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. 353)) or if the LHS is greater than the RHS (**operator>()** (p. 354)).

Parameters

sw	is the RHS object.

Returns

`true` if the LHS is less than or equal the RHS, otherwise `false`.

See also

operator==() (p. 353),
operator<() (p. 353).

6.248.5.11 bool Arc::Software::operator== (const Software & *sw*) const `[inline]`

Equality operator.

Two **Software** (p. 347) 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. 350) **ComparisonOperatorEnum** (p. 349) value.

Parameters

sw	is the RHS **Software** (p. 347) object.

Returns

`true` when the two objects equals, otherwise `false`.

6.248.5.12 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`.

6.248.5.13 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. 353)) or if the LHS is greater than the RHS (**operator>()** (p. 354)).

Parameters

sw	is the RHS object.

Returns

`true` if the LHS is greated than or equal the RHS, otherwise `false`.

See also

operator==() (p. 353),
operator>() (p. 354).

6.248.5.14 static std::string Arc::Software::toString (ComparisonOperator *co*) `[static]`

Convert **Software::ComparisonOperator** (p. 349) 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. 349).

Returns

The string representation of the passed **Software::ComparisonOperator** (p. 349) is returned.

6.248.6 Friends And Related Function Documentation

6.248.6.1 std::ostream& operator<< (std::ostream & *out*, const Software & *sw*) [friend]

Write **Software** (p. 347) string representation to a std::ostream.

Write the string representation of a **Software** (p. 347) object to a std::ostream.

Parameters

out	is a std::ostream to write the string representation of the **Software** (p. 347) object to.
sw	is the **Software** (p. 347) object to write to the std::ostream.

Returns

The passed std::ostream *out* is returned.

6.248.7 Field Documentation

6.248.7.1 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

6.249 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

6.249.1 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. 347) object and either a **Software::ComparisonOperator** (p. 349) method pointer or equally a **Software::ComparisonOperatorEnum** (p. 349) enum value. A **SoftwareRequirement** (p. 356) 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. 347) object or a list of either **Software** (p. 347) or **ApplicationEnvironment** (p. 56) objects, by using the method **isSatisfied()** (p. 361). This class also contain a number of methods (**selectSoftware()** (p. 363)) to select **Software** (p. 347) objects which are satisfying the requirements, and in this way resolving requirements.

6.249.2 Constructor & Destructor Documentation

6.249.2.1 Arc::SoftwareRequirement::SoftwareRequirement (bool *requiresAll* = `false`) `[inline]`

Create a empty **SoftwareRequirement** (p. 356) object.

The created **SoftwareRequirement** (p. 356) 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.

6.249.2.2 Arc::SoftwareRequirement::SoftwareRequirement (const Software & *sw*, Software::ComparisonOperator *swComOp* = `&Software::operator==`, bool *requiresAll* = `false`)

Create a **SoftwareRequirement** (p. 356) object.

The created **SoftwareRequirement** (p. 356) object will contain one requirement specified by the **Software** (p. 347) object *sw*, and the **Software::ComparisonOperator** (p. 349) *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. 358) instead.

Parameters

sw	is the **Software** (p. 347) object of the requirement to add.
swComOp	is the **Software::ComparisonOperator** (p. 349) 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.

6.249.2.3 Arc::SoftwareRequirement::SoftwareRequirement (const Software & *sw*, Software::ComparisonOperatorEnum *co*, bool *requiresAll* = `false`)

Create a **SoftwareRequirement** (p. 356) object.

The created **SoftwareRequirement** (p. 356) object will contain one requirement specified by the **Software** (p. 347) object *sw*, and the **Software::ComparisonOperatorEnum** (p. 349) *co*.

Parameters

sw	is the **Software** (p. 347) object of the requirement to add.
co	is the **Software::ComparisonOperatorEnum** (p. 349) 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.

6.249.2.4 Arc::SoftwareRequirement::SoftwareRequirement (const SoftwareRequirement & *sr*) `[inline]`

Copy constructor.

Create a **SoftwareRequirement** (p. 356) object from another **SoftwareRequirement** (p. 356) object.

Parameters

sr	is the **SoftwareRequirement** (p. 356) object to make a copy of.

6.249.3 Member Function Documentation

6.249.3.1 void Arc::SoftwareRequirement::add (const Software & *sw*, Software::ComparisonOperator *swComOp* = `&Software::operator==`)

Add a **Software** (p. 347) 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. 359) instead.

Parameters

sw	is the **Software** (p. 347) object to add as part of a requirement.
swComOp	is the **Software::ComparisonOperator** (p. 349) method pointer to add as part of a requirement, the default operator will be **Software::operator==()** (p. 353).

6.249.3.2 void Arc::SoftwareRequirement::add (const Software & *sw*, Software::ComparisonOperatorEnum *co*)

Add a **Software** (p. 347) 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. 347) object to add as part of a requirement.
co	is the **Software::ComparisonOperatorEnum** (p. 349) value to add as part of a requirement, the default enum will be **Software::EQUAL** (p. 350).

6.249.3.3 void Arc::SoftwareRequirement::clear ( ) `[inline]`

Clear the object.

The requirements in this object will be cleared when invoking this method.

6.249.3.4 bool Arc::SoftwareRequirement::empty ( ) const `[inline]`

Test if the object is empty.

Returns

`true` if this object do no contain any requirements, otherwise `false`.

6.249.3.5 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. 349),
getSoftwareList (p. 359).

6.249.3.6 const std::list<Software>& Arc::SoftwareRequirement::getSoftwareList ( ) const `[inline]`

Get list of **Software** (p. 347) objects.

Returns

The list of internally stored **Software** (p. 347) objects is returned.

See also

Software (p. 347),
getComparisonOperatorList (p. 359).

6.249.3.7 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. 364).

6.249.3.8 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. 353)).

If specified that all requirements has to be satisfied, then for this object to be resolved each requirement must have a **Software** (p. 347) object with a unique family/name composition, i.e. no other requirements have a **Software** (p. 347) object with the same family/name composition, and each requirement must use the equal operator (**Software::operator==** (p. 353)).

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`.

6.249.3.9 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. 361)method does.

Parameters

swList	is the list of **ApplicationEnvironment** (p. 56) 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. 361), **isSatisfied(const std::list<Software>&) const** (p. 361), **selectSoftware(const std::list<ApplicationEnvironment>&)** (p. 363), **isResolved() const** (p. 360).

6.249.3.10 bool Arc::SoftwareRequirement::isSatisfied (const Software & *sw*) const `[inline]`

Test if requirements are satisfied.

Returns `true` if the requirements are satisfied by the specified **Software** (p. 347) *sw*, otherwise `false` is returned.

Parameters

sw	is the **Software** (p. 347) which should satisfy the requirements.

Returns

`true` if requirements are satisfied, otherwise `false`.

See also

isSatisfied(const std::list<Software>&) const (p. 361),
isSatisfied(const std::list<ApplicationEnvironment>&) const (p. 360),
selectSoftware(const Software&) (p. 363),
isResolved() const (p. 360).

References isSatisfied().

Referenced by isSatisfied().

6.249.3.11 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. 347) object.

Parameters

swList	is the list of **Software** (p. 347) 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. 361),
isSatisfied(const std::list<ApplicationEnvironment>&) const (p. 360),
selectSoftware(const std::list<Software>&) (p. 362),
isResolved() const (p. 360).

6.249.3.12 SoftwareRequirement& Arc::SoftwareRequirement::operator= (const SoftwareRequirement & *sr*)

Assignment operator.

Set this object equal to that of the passed **SoftwareRequirement** (p. 356) object *sr*.

Parameters

sr	is the **SoftwareRequirement** (p. 356) object to set object equal to.

6.249.3.13 bool Arc::SoftwareRequirement::selectSoftware (const std::list< Software > & *swList*)

Select software.

If the passed list of **Software** (p. 347) objects *swList* do not satisfy the requirements `false` is returned and this object is not modified. If however the list of **Software** (p. 347) objects *swList* do satisfy the requirements `true` is returned and the **Software** (p. 347) objects satisfying the requirements will replace these with the equality operator (**Software::operator==** (p. 353)) 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. 347) object and it will replace all these requirements.

Parameters

swList	is a list of **Software** (p. 347) objects used to satisfy requirements.

Returns

`true` if requirements are satisfied, otherwise `false`.

See also

selectSoftware(const Software&) (p. 363),
selectSoftware(const std::list<ApplicationEnvironment>&) (p. 363),
isSatisfied(const std::list<Software>&) const (p. 361),
isResolved() const (p. 360).

6.249.3.14 bool Arc::SoftwareRequirement::selectSoftware (const Software & *sw*) `[inline]`

Select software.

If the passed **Software** (p. 347) *sw* do not satisfy the requirements `false` is returned and this object is not modified. If however the **Software** (p. 347) object *sw* do satisfy the requirements `true` is returned and the requirements are set to equal the *sw* **Software** (p. 347) object.

Parameters

sw	is the **Software** (p. 347) object used to satisfy requirements.

Returns

`true` if requirements are satisfied, otherwise `false`.

See also

selectSoftware(const std::list<Software>&) (p. 362),
selectSoftware(const std::list<ApplicationEnvironment>&) (p. 363),
isSatisfied(const Software&) const (p. 361),

isResolved() const (p. 360).

References selectSoftware().

Referenced by selectSoftware().

6.249.3.15 bool Arc::SoftwareRequirement::selectSoftware (const std::list< ApplicationEnvironment > & *swList*)

Select software.

This method behaves exactly as the **selectSoftware(const std::list<Software>&)** (p. 362) method does.

Parameters

swList	is a list of **ApplicationEnvironment** (p. 56) objects used to satisfy requirements.

Returns

`true` if requirements are satisfied, otherwise `false`.

See also

selectSoftware(const Software&) (p. 363),
selectSoftware(const std::list<Software>&) (p. 362),
isSatisfied(const std::list<ApplicationEnvironment>&) const (p. 360),
isResolved() const (p. 360).

6.249.3.16 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. 360).

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

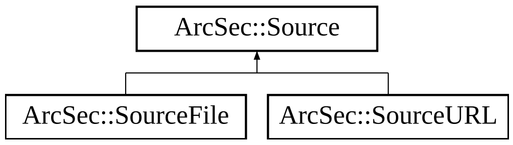
- Software.h

6.250 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)

6.250.1 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.

6.250.2 Constructor & Destructor Documentation

6.250.2.1 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.

6.250.2.2 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:

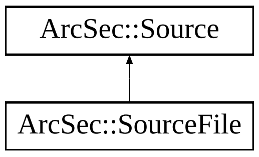
- Source.h

6.251 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)

6.251.1 Detailed Description

Convenience class for obtaining XML document from file.

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

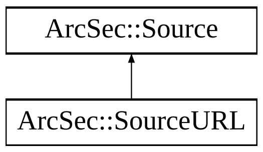
- Source.h

6.252 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)

6.252.1 Detailed Description

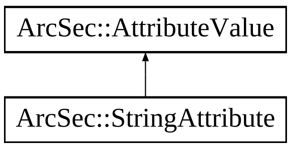
Convenience class for obtaining XML document from remote URL.

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

- Source.h

6.253 ArcSec::StringAttribute Class Reference

Inheritance diagram for ArcSec::StringAttribute:



Public Member Functions

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

6.253.1 Member Function Documentation

6.253.1.1 virtual std::string ArcSec::StringAttribute::encode ( ) `[inline, virtual]`

encode the value in a string format

Implements **ArcSec::AttributeValue** (p. 64).

6.253.1.2 virtual bool ArcSec::StringAttribute::equal (AttributeValue ∗ *value*, bool *check_id* = `true`) `[virtual]`

Evluate whether "this" equale to the parameter value

Implements **ArcSec::AttributeValue** (p. 64).

6.253.1.3 virtual std::string ArcSec::StringAttribute::getId ( ) `[inline, virtual]`

Get the AttributeId of the <Attribute>

Implements **ArcSec::AttributeValue** (p. 64).

6.253.1.4 virtual std::string ArcSec::StringAttribute::getType ( ) `[inline, virtual]`

Get the DataType of the <Attribute>

Implements **ArcSec::AttributeValue** (p. 65).

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

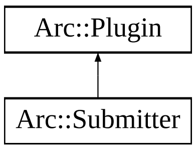
- StringAttribute.h

6.254 Arc::Submitter Class Reference

Base class for the Submitters.

`#include <Submitter.h>`

Inheritance diagram for Arc::Submitter:



Public Member Functions

- virtual bool **GetTestJob** (const int &testid, **JobDescription** &jobdescription)
- **URL Submit** (const **JobDescription** &jobdesc, const **ExecutionTarget** &et)
- **URL Migrate** (const **URL** &jobid, const **JobDescription** &jobdesc, const **ExecutionTarget** &et, bool forcemigration)

Protected Attributes

- const **ExecutionTarget** ∗ **target**

6.254.1 Detailed Description

Base class for the Submitters. **Submitter** (p. 367) is the base class for Grid middleware specialized **Submitter** (p. 367) objects. The class submits job(s) to the computing resource it represents and uploads (needed by the job) local input files.

6.254.2 Member Function Documentation

6.254.2.1 virtual bool Arc::Submitter::GetTestJob (const int & *testid,* JobDescription & *jobdescription*) [inline, virtual]

This virtual method can be ovverriden by plugins which should be capable of getting test job descriptions for the specified flavour. This method should return with the **Job-Description** (p. 227) or NULL if ther is no test description defined with the requested id.

6.254.2.2 URL Arc::Submitter::Migrate (const URL & *jobid,* const JobDescription & *jobdesc,* const ExecutionTarget & *et,* bool *forcemigration*)

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. 387) jobid, and is represented by the **JobDescription** (p. 227) 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. 387) of the migrated job. In case migration fails an empty **URL** (p. 387) should be returned.

6.254.2.3 URL Arc::Submitter::Submit (const JobDescription & *jobdesc,* const ExecutionTarget & *et*)

This virtual method should be overridden by plugins which should be capable of submitting jobs, defined in the **JobDescription** (p. 227) jobdesc, to the **ExecutionTarget** (p. 177) et. The protected convenience method AddJob can be used to save job information. This method should return the **URL** (p. 387) of the submitted job. In case submission fails an empty **URL** (p. 387) should be returned.

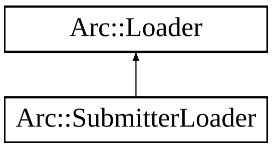
The documentation for this class was generated from the following file:

- Submitter.h

6.255 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

6.255.1 Detailed Description

Class responsible for loading **Submitter** (p. 367) plugins The **Submitter** (p. 367) objects returned by a **SubmitterLoader** (p. 369) must not be used after the **SubmitterLoader** (p. 369) goes out of scope.

6.255.2 Constructor & Destructor Documentation

6.255.2.1 Arc::SubmitterLoader::SubmitterLoader ( )

Constructor Creates a new **SubmitterLoader** (p. 369).

6.255.2.2 Arc::SubmitterLoader::∼SubmitterLoader ( )

Destructor Calling the destructor destroys all Submitters loaded by the **SubmitterLoader** (p. 369) instance.

6.255.3 Member Function Documentation

6.255.3.1 const std::list<Submitter∗>& Arc::SubmitterLoader::GetSubmitters ( ) const [inline]

Retrieve the list of loaded Submitters.

Returns

A reference to the list of Submitters.

6.255.3.2 Submitter∗ Arc::SubmitterLoader::load (const std::string & *name*, const UserConfig & *usercfg*)

Load a new **Submitter** (p. 367)

Parameters

name	The name of the **Submitter** (p. 367) to load.
usercfg	The **UserConfig** (p. 399) object for the new **Submitter** (p. 367).

Returns

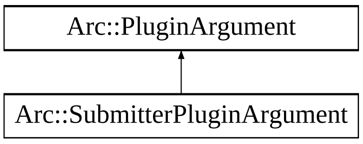
A pointer to the new **Submitter** (p. 367) (NULL on error).

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

- Submitter.h

6.256 Arc::SubmitterPluginArgument Class Reference

Inheritance diagram for Arc::SubmitterPluginArgument:



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

- Submitter.h

6.257 Arc::TargetGenerator Class Reference

Target generation class

```
#include <TargetGenerator.h>
```

Public Member Functions

- **TargetGenerator** (const **UserConfig** &usercfg, unsigned int startRetrieval=0)
- void **GetTargets** (int targetType, int detailLevel)
- void **RetrieveExecutionTargets** ()
- void **RetrieveJobs** ()
- const std::list< **ExecutionTarget** > & **GetExecutionTargets** () const
- std::list< **ExecutionTarget** > & **ModifyFoundTargets** ()
- const std::list< **ExecutionTarget** > & **FoundTargets** () const
- const std::list< **XMLNode** $*$ > & **FoundJobs** () const
- const std::list< **Job** > & **GetJobs** () const
- bool **AddService** (const std::string Flavour, const **URL** &url)
- bool **AddIndexServer** (const std::string Flavour, const **URL** &url)
- void **AddTarget** (const **ExecutionTarget** &target)
- void **AddJob** (const **XMLNode** &job)
- void **AddJob** (const **Job** &job)

- void **PrintTargetInfo** (bool longlist) const
- void **SaveTargetInfoToStream** (std::ostream &out, bool longlist) const
- **SimpleCounter** & **ServiceCounter** (void)

6.257.1 Detailed Description

Target generation class The **TargetGenerator** (p. 371) class is the umbrella class for resource discovery and information retrieval (index servers and execution services). It can also be used to discover user Grid jobs and detailed information. The **TargetGenerator** (p. 371) loads **TargetRetriever** (p. 376) plugins (which implements the actual information retrieval) from **URL** (p. 387) objects found in the **UserConfig** (p. 399) object passed to its constructor using the custom **TargetRetrieverLoader** (p. 378).

6.257.2 Constructor & Destructor Documentation

6.257.2.1 Arc::TargetGenerator::TargetGenerator (const UserConfig & *usercfg*, unsigned int *startRetrieval* = 0)

Create a **TargetGenerator** (p. 371) object.

Default constructor to create a TargeGenerator. The constructor reads the computing and index service **URL** (p. 387) objects from the passed **UserConfig** (p. 399) object using the **UserConfig** (p. 399):GetSelectedServices method. From each **URL** (p. 387) a matching specialized **TargetRetriever** (p. 376) plugin is loaded using the **TargetRetrieverLoader** (p. 378). If the second parameter, startRetrieval, is specified, and matches bitwise either a value of 1, 2 or both, retrieval of execution services, jobs or both will be initiated.

Parameters

usercfg	is a reference to a **UserConfig** (p. 399) object from which endpoints to execution and/or index services will be used. The object also hold information about user credentials.
startRetrival	specifies whether retrival should be started directly. It will be parsed bitwise. A value of 1 will start execution service retrieval (RetrieveExecutionTargets), 2 jobs (RetrieveJobs), and 3 both, while 0 will not start retrieval at all. If not specified, default is 0.

6.257.3 Member Function Documentation

6.257.3.1 bool Arc::TargetGenerator::AddIndexServer (const std::string *Flavour*, 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. 387) against the servers returned by **UserConfig::GetRejectedServices** (p. 413) and only allows to add the service if not specifically rejected.

Parameters

flavour	The flavour if the the index server.
url	**URL** (p. 387) pointing to the index server.

6.257.3.2 void Arc::TargetGenerator::AddJob (const XMLNode & *job*)

DEPRECATED: Add a new **Job** (p. 215) to this object.

This method is DEPRECATED, use the **AddJob(const Job&)** (p. 372) method instead. Method to add a new **Job** (p. 215) (usually discovered by a **TargetRetriever** (p. 376)) to the internal list of jobs in a thread secure way.

Parameters

job	**XMLNode** (p. 465) describing the job.

6.257.3.3 void Arc::TargetGenerator::AddJob (const Job & *job*)

Add a new **Job** (p. 215) to this object.

Method to add a new **Job** (p. 215) (usually discovered by a **TargetRetriever** (p. 376)) to the internal list of jobs in a thread secure way.

Parameters

job	**Job** (p. 215) describing the job.

See also

AddJob(const Job&) (p. 372)

6.257.3.4 bool Arc::TargetGenerator::AddService (const std::string *Flavour,* 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. 387) against the services returned by **UserConfig::GetRejectedServices** (p. 413) and only allows to add the service if not specifically rejected.

Parameters

flavour	The flavour if the the computing service.
url	**URL** (p. 387) pointing to the information system of the computing service.

6.257.3.5 void Arc::TargetGenerator::AddTarget (const ExecutionTarget & *target*)

Add a new **ExecutionTarget** (p. 177) to the foundTargets list.

Method to add a new **ExecutionTarget** (p. 177) (usually discovered by a **TargetRetriever** (p. 376)) to the list of foundTargets in a thread secure way.

Parameters

target	**ExecutionTarget** (p. 177) to be added.

6.257.3.6 const std::list<XMLNode∗>& Arc::TargetGenerator::FoundJobs ( ) const

DEPRECATED: Return jobs found by GetTargets.

This method is DEPRECATED, use the GetFoundJobs method instead. Method to return the list of jobs found by a call to the GetJobs method.

Returns

A list of jobs in XML format is returned.

6.257.3.7 const std::list<ExecutionTarget>& Arc::TargetGenerator::FoundTargets ( ) const [inline]

DEPRECATED: Return targets found by GetTargets.

This method is DEPRECATED, use the **FoundTargets()** (p. 373) instead. Method to return the list of **ExecutionTarget** (p. 177) objects (currently only supported Target type) found by the GetTarget method.

6.257.3.8 const std::list<ExecutionTarget>& Arc::TargetGenerator::GetExecutionTargets ( ) const [inline]

Return targets fetched by RetrieveExecutionTargets method.

Method to return a const list of **ExecutionTarget** (p. 177) objects retrieved by the RetrieveExecutionTargets method.

See also

RetrieveExecutionTargets (p. 375)
GetExecutionTargets (p. 374)

6.257.3.9 const std::list<Job>& Arc::TargetGenerator::GetJobs ( ) const [inline]

Return jobs retrieved by RetrieveJobs method.

Method to return the list of jobs found by a call to the GetJobs method.

Returns

A list of the discovered jobs as **Job** (p. 215) objects is returned

See also

RetrieveJobs (p. 375)

6.257.3.10 void Arc::TargetGenerator::GetTargets (int *targetType*, int *detailLevel*)

DEPRECATED: Find available targets.

This method is DEPRECATED, use the **RetrieveExecutionTargets()** (p. 375) or **RetrieveJobs()** (p. 375) method instead. Method to prepare a list of chosen Targets with a specified detail level. Current implementation supports finding computing elements (**ExecutionTarget** (p. 177)) with full detail level and jobs with limited detail level.

Parameters

targetType	0 = **ExecutionTarget** (p. 177), 1 = Grid jobs
detailLevel	

See also

RetrieveExecutionsTargets()
RetrieveJobs() (p. 375)

6.257.3.11 std::list<ExecutionTarget>& Arc::TargetGenerator::ModifyFoundTargets ( )

DEPRECATED: Return targets found by GetTargets.

This method is DEPRECATED, use the **FoundTargets()** (p. 373) instead. Method to return the list of **ExecutionTarget** (p. 177) objects (currently only supported Target type) found by the GetTarget method.

6.257.3.12 void Arc::TargetGenerator::PrintTargetInfo (bool *longlist*) const

DEPRECATED: Prints target information.

This method is DEPRECATED, use the SaveTargetInfoToStream method instead. Method to print information of the found targets to std::cout.

Parameters

longlist	false for minimal information, true for detailed information

See also

SaveTargetInfoToStream (p. 376)

6.257.3.13 void Arc::TargetGenerator::RetrieveExecutionTargets ( )

Retrieve available execution services.

The endpoints specified in the **UserConfig** (p. 399) object passed to this object will be used to retrieve information about execution services (**ExecutionTarget** (p. 177) objects). The discovery and information retrieval of targets is carried out in parallel threads to speed up the process. If a endpoint is a index service each execution service registered will be queried.

See also

RetrieveJobs (p. 375)
GetExecutionTargets (p. 374)

6.257.3.14 void Arc::TargetGenerator::RetrieveJobs ( )

Retrieve job information from execution services.

The endpoints specified in the **UserConfig** (p. 399) object passed to this object will be used to retrieve job information from these endpoints. Only jobs owned by the user which is identified by the credentials specified in the passed **UserConfig** (p. 399) object will be considered (exception being services which has no user authentification). If a endpoint is a index service, each execution service registered will be queried, and searched for job information.

See also

RetrieveExecutionTargets (p. 375)

6.257.3.15 void Arc::TargetGenerator::SaveTargetInfoToStream (std::ostream & *out*, bool *longlist*) const

Prints target information.

Method to print information of the found targets to std::cout.

Parameters

out	is a std::ostream object which to direct target informetion to.
longlist	false for minimal information, true for detailed information

6.257.3.16 SimpleCounter& Arc::TargetGenerator::ServiceCounter (void)

Returns reference to worker counter.

This method returns reference to counter which keeps amount of started worker threads communicating with services asynchronously. The counter must be incremented for every thread started and decremented when thread exits. Main thread will then wait till counters drops to zero.

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

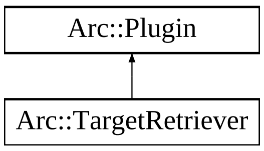
- TargetGenerator.h

6.258 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)
- virtual void **GetExecutionTargets** (**TargetGenerator** &mom)=0
- virtual void **GetJobs** (**TargetGenerator** &mom)=0

6.258.1 Detailed Description

TargetRetriever base class The **TargetRetriever** (p. 376) 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. 371).

6.258.2 Constructor & Destructor Documentation

6.258.2.1 Arc::TargetRetriever::TargetRetriever (const UserConfig & *usercfg*, const URL & *url*, ServiceType *st*, const std::string & *flavour*) `[protected]`

TargetRetriever (p. 376) constructor.

Default constructor to create a TargeGenerator. The constructor reads the computing and index service **URL** (p. 387) objects from the

Parameters

usercfg	
url	
st	
flavour	

6.258.3 Member Function Documentation

6.258.3.1 virtual void Arc::TargetRetriever::GetExecutionTargets (TargetGenerator & *mom*) `[protected, 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. 371) which has loaded the **TargetRetriever** (p. 376)
detailLevel	is the required level of details (1 = All details, 2 = Limited details)

6.258.3.2 virtual void Arc::TargetRetriever::GetJobs (TargetGenerator & *mom*) `[protected, 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. 371) which has loaded the **TargetRetriever** (p. 376)
detailLevel	is the required level of details (1 = All details, 2 = Limited details)

6.258.3.3 virtual void Arc::TargetRetriever::GetTargets (TargetGenerator & *mom*, int *targetType*, int *detailLevel*) `[pure virtual]`

DEPRECATED: Method for collecting targets.

This method is DEPRECATED, the GetExecutionTargets and GetJobs methods replaces it.

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. 371) which has loaded the **TargetRetriever** (p. 376)
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)

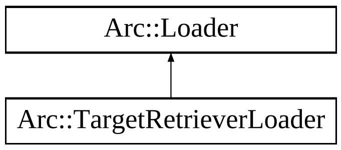
The documentation for this class was generated from the following file:

- TargetRetriever.h

6.259 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 std::string &service, const ServiceType &st)
- const std::list< **TargetRetriever** ∗ > & **GetTargetRetrievers** () const

6.259.1 Detailed Description

Class responsible for loading **TargetRetriever** (p. 376) plugins The **TargetRetriever** (p. 376) objects returned by a **TargetRetrieverLoader** (p. 378) must not be used after the **TargetRetrieverLoader** (p. 378) goes out of scope.

6.259.2 Constructor & Destructor Documentation

6.259.2.1 Arc::TargetRetrieverLoader::TargetRetrieverLoader ( )

Constructor Creates a new **TargetRetrieverLoader** (p. 378).

6.259.2.2 Arc::TargetRetrieverLoader::∼TargetRetrieverLoader ( )

Destructor Calling the destructor destroys all TargetRetrievers loaded by the **TargetRetrieverLoader** (p. 378) instance.

6.259.3 Member Function Documentation

6.259.3.1 const std::list<TargetRetriever∗>& Arc::TargetRetrieverLoader::GetTargetRetrievers ( ) const [inline]

Retrieve the list of loaded TargetRetrievers.

Returns

A reference to the list of TargetRetrievers.

6.259.3.2 TargetRetriever∗ Arc::TargetRetrieverLoader::load (const std::string & *name*, const UserConfig & *usercfg*, const std::string & *service*, const ServiceType & *st*)

Load a new **TargetRetriever** (p. 376)

Parameters

name	The name of the **TargetRetriever** (p. 376) to load.
usercfg	The **UserConfig** (p. 399) object for the new **TargetRetriever** (p. 376).
service	The **URL** (p. 387) used to contact the target.
st	specifies service type of the target.

Returns

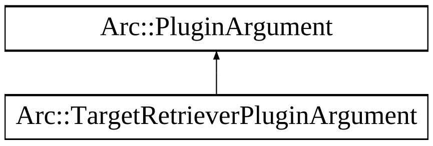
A pointer to the new **TargetRetriever** (p. 376) (NULL on error).

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

- TargetRetriever.h

6.260 Arc::TargetRetrieverPluginArgument Class Reference

Inheritance diagram for Arc::TargetRetrieverPluginArgument:



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

- TargetRetriever.h

6.261 Arc::ThreadDataItem Class Reference

Base class for per-thread object.

`#include <Thread.h>`

Public Member Functions

- **ThreadDataItem** (void)
- **ThreadDataItem** (std::string &key)
- **ThreadDataItem** (const std::string &key)
- void **Attach** (std::string &key)
- void **Attach** (const std::string &key)
- virtual void **Dup** (void)

Static Public Member Functions

- static **ThreadDataItem** ∗ **Get** (const std::string &key)

6.261.1 Detailed Description

Base class for per-thread object. Classes inherited from this one are attached to current thread under specified key and destroyed only when thread ends or object is replaced by another one with same key.

6.261.2 Constructor & Destructor Documentation

6.261.2.1 Arc::ThreadDataItem::ThreadDataItem (void)

Dummy constructor which does nothing. To make object usable one of Attach(...) methods must be used.

6.261.2.2 Arc::ThreadDataItem::ThreadDataItem (std::string & *key*)

Creates instance and attaches it to current thread under key. If supplied key is empty random one is generated and stored in key variable.

6.261.2.3 Arc::ThreadDataItem::ThreadDataItem (const std::string & *key*)

Creates instance and attaches it to current thread under key.

6.261.3 Member Function Documentation

6.261.3.1 void Arc::ThreadDataItem::Attach (std::string & *key*)

Attaches object to current thread under key. If supplied key is empty random one is generated and stored in key variable. This method must be used only if object was created using dummy constructor.

6.261.3.2 void Arc::ThreadDataItem::Attach (const std::string & *key*)

Attaches object to current thread under key. This method must be used only if object was created using dummy constructor.

6.261.3.3 virtual void Arc::ThreadDataItem::Dup (void) `[virtual]`

Creates copy of object. This method is called when new thread is created from current thread. It is called in new thread, so new object - if created - gets attached to new thread. If object is not meant to be inherited by new threads then this method should do nothing.

6.261.3.4 static ThreadDataItem∗ Arc::ThreadDataItem::Get (const std::string & *key*) `[static]`

Retrieves object attached to thread under key. Returns if no such obejct.

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

- Thread.h

6.262 Arc::ThreadInitializer Class Reference

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

- Thread.h

6.263 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)

6.263.1 Detailed Description

This class is a set of conditions, mutexes, etc. conveniently exposed to monitor running child threads and to wait till they exit. There are no protections against race conditions. So use it carefully.

6.263.2 Member Function Documentation

6.263.2.1 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.

6.263.2.2 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

6.264 Arc::Time Class Reference

A class for storing and manipulating times.

`#include <DateTime.h>`

Public Member Functions

- **Time** ()
- **Time** (time_t)
- **Time** (time_t time, uint32_t nanosec)
- **Time** (const std::string &)
- **Time** & **operator=** (time_t)
- **Time** & **operator=** (const **Time** &)
- **Time** & **operator=** (const char ∗)
- **Time** & **operator=** (const std::string &)
- void **SetTime** (time_t)
- void **SetTime** (time_t time, uint32_t nanosec)
- time_t **GetTime** () const
- **operator std::string** () const
- std::string **str** (const **TimeFormat** &=time_format) const
- bool **operator<** (const **Time** &) const
- bool **operator>** (const **Time** &) const
- bool **operator<=** (const **Time** &) const

- bool **operator>=** (const **Time** &) const
- bool **operator==** (const **Time** &) const
- bool **operator!=** (const **Time** &) const
- **Time operator+** (const **Period** &) const
- **Time operator-** (const **Period** &) const
- **Period operator-** (const **Time** &) const

Static Public Member Functions

- static void **SetFormat** (const **TimeFormat** &)
- static **TimeFormat GetFormat** ()

6.264.1 Detailed Description

A class for storing and manipulating times.

6.264.2 Constructor & Destructor Documentation

6.264.2.1 Arc::Time::Time ( )

Default constructor. The time is put equal the current time.

6.264.2.2 Arc::Time::Time (time_t)

Constructor that takes a time_t variable and stores it.

6.264.2.3 Arc::Time::Time (time_t *time,* uint32_t *nanosec*)

Constructor that takes a fine grained time variables and stores them.

6.264.2.4 Arc::Time::Time (const std::string &)

Constructor that tries to convert a string into a time_t.

6.264.3 Member Function Documentation

6.264.3.1 static TimeFormat Arc::Time::GetFormat ( ) `[static]`

Gets the default format for time strings.

6.264.3.2 time_t Arc::Time::GetTime ( ) const

gets the time

6.264.3.3 Arc::Time::operator std::string ( ) const

Returns a string representation of the time, using the default format.

6.264.3.4 bool Arc::Time::operator!= (const Time &) const

Comparing two **Time** (p. 383) objects.

6.264.3.5 Time Arc::Time::operator+ (const Period &) const

Adding **Time** (p. 383) object with **Period** (p. 298) object.

6.264.3.6 Time Arc::Time::operator- (const Period &) const

Subtracting **Period** (p. 298) object from **Time** (p. 383) object.

6.264.3.7 Period Arc::Time::operator- (const Time &) const

Subtracting **Time** (p. 383) object from the other **Time** (p. 383) object.

6.264.3.8 bool Arc::Time::operator< (const Time &) const

Comparing two **Time** (p. 383) objects.

6.264.3.9 bool Arc::Time::operator<= (const Time &) const

Comparing two **Time** (p. 383) objects.

6.264.3.10 Time& Arc::Time::operator= (const char *)

Assignment operator from a char pointer.

6.264.3.11 Time& Arc::Time::operator= (const std::string &)

Assignment operator from a string.

6.264.3.12 Time& Arc::Time::operator= (const Time &)

Assignment operator from a **Time** (p. 383).

6.264.3.13 Time& Arc::Time::operator= (time_t)

Assignment operator from a time_t.

6.264.3.14 bool Arc::Time::operator== (const Time &) const

Comparing two **Time** (p. 383) objects.

6.264.3.15 bool Arc::Time::operator> (const Time &) const

Comparing two **Time** (p. 383) objects.

6.264.3.16 bool Arc::Time::operator>= (const Time &) const

Comparing two **Time** (p. 383) objects.

6.264.3.17 static void Arc::Time::SetFormat (const TimeFormat &) [static]

Sets the default format for time strings.

6.264.3.18 void Arc::Time::SetTime (time_t)

sets the time

6.264.3.19 void Arc::Time::SetTime (time_t *time*, uint32_t *nanosec*)

sets the fine grained time

6.264.3.20 std::string Arc::Time::str (const TimeFormat & = time_format) const

Returns a string representation of the time, using the specified format.

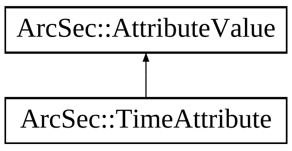
The documentation for this class was generated from the following file:

- DateTime.h

6.265 ArcSec::TimeAttribute Class Reference

`#include <DateTimeAttribute.h>`

Inheritance diagram for ArcSec::TimeAttribute:



Public Member Functions

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

6.265.1 Detailed Description

Format: HHMMSSZ HH:MM:SS HH:MM:SS+HH:MM HH:MM:SSZ

6.265.2 Member Function Documentation

6.265.2.1 virtual std::string ArcSec::TimeAttribute::encode ( ) `[virtual]`

encode the value in a string format

Implements **ArcSec::AttributeValue** (p. 64).

6.265.2.2 virtual bool ArcSec::TimeAttribute::equal (AttributeValue ∗ *value*, bool *check_id* = `true`) `[virtual]`

Evluate whether "this" equale to the parameter value

Implements **ArcSec::AttributeValue** (p. 64).

6.265.2.3 virtual std::string ArcSec::TimeAttribute::getId ( ) `[inline, virtual]`

Get the AttributeId of the <Attribute>

Implements **ArcSec::AttributeValue** (p. 64).

6.265.2.4 virtual std::string ArcSec::TimeAttribute::getType ( ) `[inline, virtual]`

Get the DataType of the <Attribute>

Implements **ArcSec::AttributeValue** (p. 65).

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

- DateTimeAttribute.h

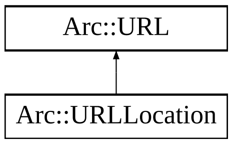
6.266 Arc::TimedMutex Class Reference

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

- Thread.h

6.267 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)
- bool **IsSecureProtocol** () const
- const std::string & **Username** () const
- const std::string & **Passwd** () const
- const std::string & **Host** () const
- void **ChangeHost** (const std::string &newhost)
- int **Port** () const
- void **ChangePort** (int newport)
- const std::string & **Path** () const
- std::string **FullPath** () const
- 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)
- void **AddMetaDataOption** (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 **plainstr** () const
- virtual std::string **fullstr** () const
- virtual std::string **ConnectionURL** () const
- bool **operator<** (const **URL** &url) const
- bool **operator==** (const **URL** &url) const
- **operator bool** () const
- bool **StringMatches** (const std::string &str) const
- std::map< std::string, std::string > **ParseOptions** (const std::string &optstring, char separator)

Static Public Member Functions

- static std::string **OptionString** (const std::map< std::string, std::string > &options, char separator)

Static Protected Member Functions

- static std::string **BaseDN2Path** (const std::string &)
- static std::string **Path2BaseDN** (const std::string &)

Protected Attributes

- std::string **protocol**
- std::string **username**
- std::string **passwd**
- std::string **host**
- bool **ip6addr**
- int **port**
- std::string **path**
- std::map< std::string, std::string > **httpoptions**
- std::map< std::string, std::string > **metadataoptions**
- std::list< std::string > **ldapattributes**
- **Scope ldapscope**
- std::string **ldapfilter**

- std::map< std::string, std::string > **urloptions**
- std::list< **URLLocation** > **locations**
- std::map< std::string, std::string > **commonlocoptions**
- bool **valid**

Friends

- std::ostream & **operator**<< (std::ostream &out, const **URL** &u)

6.267.1 Member Enumeration Documentation

6.267.1.1 enum Arc::URL::Scope

Scope for LDAP URLs

6.267.2 Constructor & Destructor Documentation

6.267.2.1 Arc::URL::URL ( )

Empty constructor. Necessary when the class is part of another class and the like.

6.267.2.2 Arc::URL::URL (const std::string & *url*)

Constructs a new **URL** (p. 387) from a string representation.

6.267.2.3 virtual Arc::URL::∼URL ( ) `[virtual]`

URL (p. 387) Destructor

6.267.3 Member Function Documentation

6.267.3.1 void Arc::URL::AddLDAPAttribute (const std::string & *attribute*)

Adds an LDAP attribute.

6.267.3.2 void Arc::URL::AddMetaDataOption (const std::string & *option,* const std::string & *value,* bool *overwrite* = `true`)

Adds a metadata option

6.267.3.3 void Arc::URL::AddOption (const std::string & *option,* const std::string & *value,* bool *overwrite* = `true`)

Adds a **URL** (p. 387) option.

6.267.3.4 static std::string Arc::URL::BaseDN2Path (const std::string &) [static, protected]

a private method that converts an ldap basedn to a path.

6.267.3.5 void Arc::URL::ChangeHost (const std::string & *newhost*)

Changes the hostname of the **URL** (p. 387).

6.267.3.6 void Arc::URL::ChangeLDAPFilter (const std::string & *newfilter*)

Changes the LDAP filter.

6.267.3.7 void Arc::URL::ChangeLDAPScope (const Scope *newscope*)

Changes the LDAP scope.

6.267.3.8 void Arc::URL::ChangePath (const std::string & *newpath*)

Changes the path of the **URL** (p. 387).

6.267.3.9 void Arc::URL::ChangePort (int *newport*)

Changes the port of the **URL** (p. 387).

6.267.3.10 void Arc::URL::ChangeProtocol (const std::string & *newprot*)

Changes the protocol of the **URL** (p. 387).

6.267.3.11 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.

6.267.3.12 const std::map<std::string, std::string>& Arc::URL::CommonLocOptions ( ) const

Returns the common location options if any.

6.267.3.13 virtual std::string Arc::URL::ConnectionURL ( ) const [virtual]

Returns a string representation with protocol, host and port only

6.267.3.14 std::string Arc::URL::FullPath ( ) const

Returns the path of the **URL** (p. 387) with all options attached.

6.267.3.15 virtual std::string Arc::URL::fullstr ( ) const [virtual]

Returns a string representation including options and locations

Reimplemented in **Arc::URLLocation** (p. 398).

6.267.3.16 const std::string& Arc::URL::Host ( ) const

Returns the hostname of the **URL** (p. 387).

6.267.3.17 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.

6.267.3.18 const std::map<std::string, std::string>& Arc::URL::HTTPOptions ( ) const

Returns HTTP options if any.

6.267.3.19 bool Arc::URL::IsSecureProtocol ( ) const

Indicates whether the protocol is secure or not.

6.267.3.20 const std::list<std::string>& Arc::URL::LDAPAttributes ( ) const

Returns the LDAP attributes if any.

6.267.3.21 const std::string& Arc::URL::LDAPFilter ( ) const

Returns the LDAP filter.

6.267.3.22 Scope Arc::URL::LDAPScope ( ) const

Returns the LDAP scope.

6.267.3.23 const std::list<URLLocation>& Arc::URL::Locations ( ) const

Returns the locations if any.

6.267.3.24 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.

6.267.3.25 const std::map<std::string, std::string>& Arc::URL::MetaDataOptions ( ) const

Returns metadata options if any.

6.267.3.26 Arc::URL::operator bool ( ) const

Check if instance holds valid **URL** (p. 387)

6.267.3.27 bool Arc::URL::operator< (const URL & *url*) const

Compares one **URL** (p. 387) to another

6.267.3.28 bool Arc::URL::operator== (const URL & *url*) const

Is one **URL** (p. 387) equal to another?

6.267.3.29 const std::string& Arc::URL::Option (const std::string & *option*, const std::string & *undefined* = "") const

Returns the value of a **URL** (p. 387) option.

Parameters

option	The option whose value is returned.
undefined	This value is returned if the **URL** (p. 387) option is not defined.

6.267.3.30 const std::map<std::string, std::string>& Arc::URL::Options ( ) const

Returns **URL** (p. 387) options if any.

6.267.3.31 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

6.267.3.32 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

6.267.3.33 const std::string& Arc::URL::Passwd ( ) const

Returns the password of the **URL** (p. 387).

6.267.3.34 const std::string& Arc::URL::Path ( ) const

Returns the path of the **URL** (p. 387).

6.267.3.35 static std::string Arc::URL::Path2BaseDN (const std::string &) [static, protected]

a private method that converts an ldap path to a basedn.

6.267.3.36 virtual std::string Arc::URL::plainstr ( ) const [virtual]

Returns a string representation of the **URL** (p. 387) without any options

6.267.3.37 int Arc::URL::Port ( ) const

Returns the port of the **URL** (p. 387).

6.267.3.38 const std::string& Arc::URL::Protocol ( ) const

Returns the protocol of the **URL** (p. 387).

6.267.3.39 virtual std::string Arc::URL::str ( ) const [virtual]

Returns a string representation of the **URL** (p. 387) including meta-options.

Reimplemented in **Arc::URLLocation** (p. 398).

6.267.3.40 const std::string& Arc::URL::Username ( ) const

Returns the username of the **URL** (p. 387).

6.267.4 Friends And Related Function Documentation

6.267.4.1 std::ostream& operator<< (std::ostream & *out*, const URL & *u*) `[friend]`

Overloaded operator << to print a **URL** (p. 387).

6.267.5 Field Documentation

6.267.5.1 std::map<std::string, std::string> Arc::URL::commonlocoptions `[protected]`

common location options for index server URLs.

6.267.5.2 std::string Arc::URL::host `[protected]`

hostname of the url.

6.267.5.3 std::map<std::string, std::string> Arc::URL::httpoptions `[protected]`

HTTP options of the url.

6.267.5.4 bool Arc::URL::ip6addr `[protected]`

if host is IPv6 numerical address notation.

6.267.5.5 std::list<std::string> Arc::URL::ldapattributes `[protected]`

LDAP attributes of the url.

6.267.5.6 std::string Arc::URL::ldapfilter `[protected]`

LDAP filter of the url.

6.267.5.7 Scope Arc::URL::ldapscope `[protected]`

LDAP scope of the url.

6.267.5.8 std::list<URLLocation> Arc::URL::locations [protected]

locations for index server URLs.

6.267.5.9 std::map<std::string, std::string> Arc::URL::metadataoptions [protected]

Meta data options

6.267.5.10 std::string Arc::URL::passwd [protected]

password of the url.

6.267.5.11 std::string Arc::URL::path [protected]

the url path.

6.267.5.12 int Arc::URL::port [protected]

portnumber of the url.

6.267.5.13 std::string Arc::URL::protocol [protected]

the url protocol.

6.267.5.14 std::map<std::string, std::string> Arc::URL::urloptions [protected]

options of the url.

6.267.5.15 std::string Arc::URL::username [protected]

username of the url.

6.267.5.16 bool Arc::URL::valid [protected]

flag to describe validity of **URL** (p. 387)

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

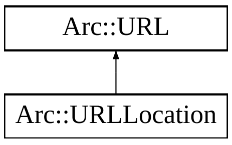
- **URL.h**

6.268 Arc::URLLocation Class Reference

Class to hold a resolved **URL** (p. 387) 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**

6.268.1 Detailed Description

Class to hold a resolved **URL** (p. 387) location. It is specific to file indexing service registrations.

6.268.2 Constructor & Destructor Documentation

6.268.2.1 Arc::URLLocation::URLLocation (const std::string & *url* = "")

Creates a **URLLocation** (p. 396) from a string representaion.

6.268.2.2 Arc::URLLocation::URLLocation (const std::string & *url*, const std::string & *name*)

Creates a **URLLocation** (p. 396) from a string representaion and a name.

6.268.2.3 Arc::URLLocation::URLLocation (const URL & *url*)

Creates a **URLLocation** (p. 396) from a **URL** (p. 387).

6.268.2.4 Arc::URLLocation::URLLocation (const URL & *url,* const std::string & *name*)

Creates a **URLLocation** (p. 396) from a **URL** (p. 387) and a name.

6.268.2.5 Arc::URLLocation::URLLocation (const std::map< std::string, std::string > & *options,* const std::string & *name*)

Creates a **URLLocation** (p. 396) from options and a name.

6.268.2.6 virtual Arc::URLLocation::~URLLocation ( ) [virtual]

URLLocation (p. 396) destructor.

6.268.3 Member Function Documentation

6.268.3.1 virtual std::string Arc::URLLocation::fullstr ( ) const [virtual]

Returns a string representation including options and locations

Reimplemented from **Arc::URL** (p. 391).

6.268.3.2 const std::string& Arc::URLLocation::Name ( ) const

Returns the **URLLocation** (p. 396) name.

6.268.3.3 virtual std::string Arc::URLLocation::str ( ) const [virtual]

Returns a string representation of the **URLLocation** (p. 396).

Reimplemented from **Arc::URL** (p. 394).

6.268.4 Field Documentation

6.268.4.1 std::string Arc::URLLocation::name [protected]

the **URLLocation** (p. 396) name as registered in the indexing service.

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

- **URL.h**

6.269 Arc::URLMap Class Reference

Data Structures

- class **map_entry**

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

- URLMap.h

6.270 Arc::User Class Reference

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

- User.h

6.271 Arc::UserConfig Class Reference

User configuration class

```
#include <UserConfig.h>
```

Public Member Functions

- **UserConfig** (**initializeCredentialsType** initializeCredentials=**initializeCredentialsType**())
- **UserConfig** (const std::string &conffile, **initializeCredentialsType** initialize-Credentials=**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)
- bool **SaveToFile** (const std::string &filename) const
- void **ApplyToConfig** (**BaseConfig** &ccfg) const
- **operator bool** () const
- bool **operator!** () const
- bool **JobListFile** (const std::string &path)
- const std::string & **JobListFile** () const
- bool **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 std::list< std::string > & **GetSelectedServices** (ServiceType st) const
- const std::list< std::string > & **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 **JobDownloadDirectory** (const std::string &newDownloadDirectory)
- const std::string & **JobDownloadDirectory** () const
- bool **IdPName** (const std::string &name)
- const std::string & **IdPName** () const

- bool **OverlayFile** (const std::string &path)
- const std::string & **OverlayFile** () const
- bool **UtilsDirPath** (const std::string &dir)
- const std::string & **UtilsDirPath** () const
- void **SetUser** (const **User** &u)
- const **User** & **GetUser** () const

Static Public Attributes

- static const std::string **ARCUSERDIRECTORY**
- static const std::string **SYSCONFIG**
- static const std::string **SYSCONFIGARCLOC**
- static const std::string **DEFAULTCONFIG**
- static const std::string **EXAMPLECONFIG**
- static const int **DEFAULT_TIMEOUT** = 20
- static const std::string **DEFAULT_BROKER**

6.271.1 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. 411)
- keypath / **KeyPath(const std::string&)** (p. 418)
- proxypath / **ProxyPath(const std::string&)** (p. 424)
- cacertificatesdirectory / **CACertificatesDirectory(const std::string&)** (p. 409)
- cacertificatepath / **CACertificatePath(const std::string&)** (p. 408)
- timeout / **Timeout(int)** (p. 426)
- joblist / **JobListFile(const std::string&)** (p. 417)
- defaultservices / **AddServices(const std::list<std::string>&, const std::list<std::string>&, ServiceType)** (p. 405)
- rejectservices / **AddServices(const std::list<std::string>&, const std::list<std::string>&, ServiceType)** (p. 405)
- verbosity / **Verbosity(const std::string&)** (p. 429)
- brokername / **Broker(const std::string&)** (p. 407) or **Broker(const std::string&, const std::string&)** (p. 408)

- brokerarguments / **Broker(const std::string&)** (p. 407) or **Broker(const std::string&, const std::string&)** (p. 408)
- bartender / Bartender(const std::list<URL>&)
- vomsserverpath / **VOMSServerPath(const std::string&)** (p. 430)
- username / **UserName(const std::string&)** (p. 427)
- password / **Password(const std::string&)** (p. 423)
- keypassword / **KeyPassword(const std::string&)** (p. 418)
- keysize / **KeySize(int)** (p. 420)
- certificatelifetime / **CertificateLifeTime(const Period&)** (p. 410)
- slcs / **SLCS(const URL&)** (p. 425)
- storedirectory / **StoreDirectory(const std::string&)** (p. 426)
- jobdownloaddirectory / **JobDownloadDirectory(const std::string&)** (p. 416)
- idpname / **IdPName(const std::string&)** (p. 414)

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. 208) 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. 404) and **AddServices(const std::list<std::string>&, const std::list<std::string>&, ServiceType)** (p. 405) methods.

The **UserConfig** (p. 399) class also provides a method **InitializeCredentials()** (p. 415) for locating user credentials by searching in different standard locations. The **CredentialsFound()** (p. 413) method can be used to test if locating the credentials succeeded.

6.271.2 Constructor & Destructor Documentation

6.271.2.1 Arc::UserConfig::UserConfig (**initializeCredentialsType** *initializeCredentials* = **initializeCredentialsType** `()`)

Create a **UserConfig** (p. 399) object.

The **UserConfig** (p. 399) 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. 415) method. The object is only non-valid if initialization of credentials fails which can be checked with the **operator bool()** (p. 422) method.

Parameters

initialize-Credentials	is a optional boolean indicating if the **InitializeCredentials()** (p. 415) method should be invoked, the default is `true`.

See also

InitializeCredentials() (p. 415)
operator bool() (p. 422)

6.271.2.2 Arc::UserConfig::UserConfig (const std::string & *conffile*, initializeCredentialsType *initializeCredentials* = initializeCredentialsType `()`, bool *loadSysConfig* = `true`)

Create a **UserConfig** (p. 399) object.

The **UserConfig** (p. 399) 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. 420) 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 **Initialize-Credentials()** (p. 415) 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.
initialize-Credentials	is a boolean indicating if credentials should be initialized, the default is `true`.
loadSysCon-fig	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. 420)
InitializeCredentials() (p. 415)
operator bool() (p. 422)
SYSCONFIG (p. 431)
EXAMPLECONFIG (p. 431)

6.271.2.3 Arc::UserConfig::UserConfig (const std::string & *conffile*, const std::string & *jfile*, initializeCredentialsType *initializeCredentials* = initializeCredentialsType `()`, bool *loadSysConfig* = `true`)

Create a **UserConfig** (p. 399) object.

The **UserConfig** (p. 399) 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. 417). 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.
initialize-Credentials	is a boolean indicating if credentials should be initialized, the default is `true`.
loadSysCon-fig	is a boolean indicating if the system configuration file should be loaded aswell, the default is `true`.

See also

JobListFile(const std::string&) (p. 417)
LoadConfigurationFile(const std::string&, bool) (p. 420)
InitializeCredentials() (p. 415)
operator bool() (p. 422)

6.271.2.4 Arc::UserConfig::UserConfig (const long int & *ptraddr*)

Language binding constructor.

The passed long int should be a pointer address to a **UserConfig** (p. 399) object, and this address is then casted into this **UserConfig** (p. 399) object.

Parameters

ptraddr	is an memory address to a **UserConfig** (p. 399) object.

6.271.3 Member Function Documentation

6.271.3.1 void Arc::UserConfig::AddBartender (const URL & *url*) `[inline]`

Set bartenders, used to contact Chelonia.

Takes as input a Bartender **URL** (p. 387) and adds this to the list of bartenders.

Parameters

url	is a **URL** (p. 387) to be added to the list of bartenders.

See also

Bartender(const std::list<URL>&)

Bartender() const (p. 407)

6.271.3.2 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. 387) <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 like-wise 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. 413)
GetRejectedServices() (p. 413)
ClearSelectedServices() (p. 412)
ClearRejectedServices() (p. 412)
LoadConfigurationFile() (p. 420)

6.271.3.3 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. 404) 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 like-wise 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. 404)
GetSelectedServices() (p. 413)
GetRejectedServices() (p. 413)
ClearSelectedServices() (p. 412)
ClearRejectedServices() (p. 412)
LoadConfigurationFile() (p. 420)

6.271.3.4 void Arc::UserConfig::ApplyToConfig (BaseConfig & *ccfg*) const

Apply credentials to **BaseConfig** (p. 67).

This methods sets the **BaseConfig** (p. 67) credentials to the credentials contained in this object. It also passes user defined configuration overlay if any.

See also

InitializeCredentials() (p. 415)
CredentialsFound() (p. 413)
BaseConfig (p. 67)

Parameters

ccfg	a **BaseConfig** (p. 67) object which will configured with the credentials of this object.

6.271.3.5 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. 387) object to be set as bartenders.

Returns

This method always returns `true`.

See also

AddBartender(const URL&) (p. 404)
Bartender() const (p. 407)

6.271.3.6 const std::vector<URL>& Arc::UserConfig::Bartender ( ) const [inline]

Get bartenders.

Returns a list of Bartender URLs

Returns

The list of bartender **URL** (p. 387) objects is returned.

See also

Bartender(const std::list<URL>&)
AddBartender(const URL&) (p. 404)

6.271.3.7 bool Arc::UserConfig::Broker (const std::string & *name*)

Set broker to use in target matching.

The string passed to this method should be in the format:

$$< name > [:< argument >]$$

where the <name> is the name of the broker and cannot contain any ':', and the optional <argument> should contain arguments which should be passed to the broker.

Two attributes are associated with this setter method 'brokername' and 'brokerarguments'.

Parameters

name	the broker name and argument specified in the format given above.

Returns

This method allways returns `true`.

See also

Broker (p. 69)
Broker(const std::string&, const std::string&) (p. 408)
Broker() const (p. 407)
DEFAULT_BROKER (p. 430)

6.271.3.8 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. 407)
Broker(const std::string&, const std::string&) (p. 408)
DEFAULT_BROKER (p. 430)

6.271.3.9 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. 407) 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. 69)

Broker(const std::string&) (p. 407)
Broker() const (p. 407)
DEFAULT_BROKER (p. 430)

6.271.3.10 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. 409)

6.271.3.11 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. 408)

6.271.3.12 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. 415) 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. 415)
CredentialsFound() const (p. 413)
CACertificatesDirectory() const (p. 409)

6.271.3.13 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. 415)
CredentialsFound() const (p. 413)
CACertificatesDirectory(const std::string&) (p. 409)

6.271.3.14 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. 341).

The attribute associated with this setter method is 'certificatelifetime'.

Parameters

newCertificateLifeTime	is the life time of a certificate, as a **Period** (p. 298) object.

Returns

This method always returns `true`.

See also

CertificateLifeTime() const (p. 410)

6.271.3.15 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. 341).

Returns

The certificate life time is returned as a **Period** (p. 298) object.

See also

CertificateLifeTime(const Period&) (p. 410)

6.271.3.16 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. 418) method. Note that the **InitializeCredentials()** (p. 415) 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. 415)
CredentialsFound() const (p. 413)
CertificatePath() const (p. 411)
KeyPath(const std::string&) (p. 418)

6.271.3.17 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. 415)
CredentialsFound() const (p. 413)
CertificatePath(const std::string&) (p. 411)
KeyPath() const (p. 419)

6.271.3.18 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. 412)
ClearSelectedServices(ServiceType) (p. 412)
AddServices(const std::list<std::string>&, ServiceType) (p. 404)
AddServices(const std::list<std::string>&, const std::list<std::string>&, ServiceType) (p. 405)
GetRejectedServices() (p. 413)

6.271.3.19 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. 411)
ClearSelectedServices() (p. 412)
AddServices(const std::list<std::string>&, ServiceType) (p. 404)
AddServices(const std::list<std::string>&, const std::list<std::string>&, ServiceType) (p. 405)
GetRejectedServices() (p. 413)

6.271.3.20 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. 412)
ClearRejectedServices(ServiceType) (p. 411)
AddServices(const std::list<std::string>&, ServiceType) (p. 404)
AddServices(const std::list<std::string>&, const std::list<std::string>&, ServiceType) (p. 405)
GetSelectedServices() (p. 413)

6.271.3.21 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. 412)
ClearRejectedServices() (p. 412)
AddServices(const std::list<std::string>&, ServiceType) (p. 404)
AddServices(const std::list<std::string>&, const std::list<std::string>&, ServiceType) (p. 405)
GetSelectedServices() (p. 413)

6.271.3.22 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. 415)

6.271.3.23 const std::list<std::string>& 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. 404)
AddServices(const std::list<std::string>&, const std::list<std::string>&, ServiceType) (p. 405)
GetSelectedServices(ServiceType)
ClearRejectedServices() (p. 412)

6.271.3.24 const std::list<std::string>& 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. 404)
AddServices(const std::list<std::string>&, const std::list<std::string>&, ServiceType) (p. 405)
GetRejectedServices(ServiceType) const (p. 413)
ClearSelectedServices() (p. 412)

6.271.3.25 const User& Arc::UserConfig::GetUser ( ) const `[inline]`

Get **User** (p. 398) for filesystem access.

Returns

The user identity to use for file system access

See also

SetUser(const User&) (p. 425)

6.271.3.26 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. 341).

The attribute associated with this setter method is 'idpname'.

Parameters

name	is the new IdP name.

Returns

This method always returns `true`.

See also

6.271.3.27 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. 414)

6.271.3.28 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. 424)
- Key/certificate path specified in either configuration file passed to the constructor or explicitly set using the setter methods **KeyPath(const std::string&)** (p. 418) and **CertificatePath(const std::string&)** (p. 411)
- 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. 409).
- 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. 57), with 'etc' and 'certificates' separated by the directory delimeter.
- Path created by concatenating the output of **ArcLocation::Get()** (p. 57), with 'etc', 'grid-security' and 'certificates' separated by the directory delimeter.
- Path created by concatenating the output of **ArcLocation::Get()** (p. 57), 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. 413)
ProxyPath(const std::string&) (p. 424)
KeyPath(const std::string&) (p. 418)
CertificatePath(const std::string&) (p. 411)
CACertificatesDirectory(const std::string&) (p. 409)

6.271.3.29 bool Arc::UserConfig::JobDownloadDirectory (const std::string & *newDownloadDirectory*) `[inline]`

Set download directory.

Sets directory which will be used to download the job directory using arcget command.

The attribute associated with this setter method is 'jobdownloaddirectory'.

Parameters

newDownloadDirectory	is the path to the download directory.

Returns

This method always returns `true`.

See also

6.271.3.30 const std::string& Arc::UserConfig::JobDownloadDirectory ( ) const `[inline]`

Get download directory.

returns directory which will be used to download the job directory using arcget command.

The attribute associated with the method is 'jobdownloaddirectory'.

Returns

This method returns the job download directory.

See also

6.271.3.31 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. 417)

6.271.3.32 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. 417)

6.271.3.33 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. 341).

The attribute associated with this setter method is 'keypassword'.

Parameters

newKey-Password	is the new password to the key.

Returns

This method always returns `true`.

See also

KeyPassword() const (p. 418)
KeyPath(const std::string&) (p. 418)
KeySize(int) (p. 420)

6.271.3.34 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. 341).

Returns

The key password is returned.

See also

KeyPassword(const std::string&) (p. 418)
KeyPath() const (p. 419)
KeySize() const (p. 419)

6.271.3.35 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. 411) method. Note that the **InitializeCredentials()** (p. 415) 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. 415)
CredentialsFound() const (p. 413)
KeyPath() const (p. 419)
CertificatePath(const std::string&) (p. 411)

KeyPassword(const std::string&) (p. 418)
KeySize(int) (p. 420)

6.271.3.36 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. 415)
CredentialsFound() const (p. 413)
KeyPath(const std::string&) (p. 418)
CertificatePath() const (p. 411)
KeyPassword() const (p. 418)
KeySize() const (p. 419)

6.271.3.37 int Arc::UserConfig::KeySize ( ) const [inline]

Get key size.

Get size/strengt of private key of credentials obtained from Short Lived Credentials **Service** (p. 341).

Returns

The key size, as an integer, is returned.

See also

KeySize(int) (p. 420)
KeyPath() const (p. 419)
KeyPassword() const (p. 418)

6.271.3.38 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. 341).

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. 419)
KeyPath(const std::string&) (p. 418)
KeyPassword(const std::string&) (p. 418)

6.271.3.39 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. 208) 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. 411))
- keypath (**KeyPath(const std::string&)** (p. 418))
- proxypath (**ProxyPath(const std::string&)** (p. 424))
- cacertificatesdirectory (**CACertificatesDirectory(const std::string&)** (p. 409))
- cacertificatepath (**CACertificatePath(const std::string&)** (p. 408))
- timeout (**Timeout(int)** (p. 426))
- joblist (**JobListFile(const std::string&)** (p. 417))
- defaultservices (**AddServices(const std::list<std::string>&, const std::list<std::string>&, ServiceType)** (p. 405))
- rejectservices (**AddServices(const std::list<std::string>&, const std::list<std::string>&, ServiceType)** (p. 405))
- verbosity (**Verbosity(const std::string&)** (p. 429))
- brokername (**Broker(const std::string&)** (p. 407) or **Broker(const std::string&, const std::string&)** (p. 408))
- brokerarguments (**Broker(const std::string&)** (p. 407) or **Broker(const std::string&, const std::string&)** (p. 408))

- bartender (Bartender(const std::list<URL>&))
- vomsserverpath (**VOMSServerPath(const std::string&)** (p. 430))
- username (**UserName(const std::string&)** (p. 427))
- password (**Password(const std::string&)** (p. 423))
- keypassword (**KeyPassword(const std::string&)** (p. 418))
- keysize (**KeySize(int)** (p. 420))
- certificatelifetime (**CertificateLifeTime(const Period&)** (p. 410))
- slcs (**SLCS(const URL&)** (p. 425))
- storedirectory (**StoreDirectory(const std::string&)** (p. 426))
- jobdownloaddirectory (**JobDownloadDirectory(const std::string&)** (p. 416))
- idpname (**IdPName(const std::string&)** (p. 414))

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. 404) and **AddServices(const std::list<std::string>&, const std::list<std::string>&, ServiceType)** (p. 405) 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. 387) 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.
ignoreJob-ListFile	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.

See also

SaveToFile() (p. 424)

6.271.3.40 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. 422)

6.271.3.41 bool Arc::UserConfig::operator! (void) const `[inline]`

Check for non-validity.

See **operator bool()** (p. 422) for a description.

See also

operator bool() (p. 422)

6.271.3.42 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. 67) 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

6.271.3.43 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. 422)

6.271.3.44 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. 341).

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. 423)

6.271.3.45 const std::string& Arc::UserConfig::Password ( ) const `[inline]`

Get password.

Get password which is used for requesting credentials from Short Lived Credentials **Service** (p. 341).

Returns

The password is returned.

See also

Password(const std::string&) (p. 423)

6.271.3.46 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. 415) method will also try to set this path, by searching in different locations.

The attribute associated with this setter method is 'proxypath'

Parameters

newProxy-Path	is the path to a user proxy.

Returns

This method always returns `true`.

See also

InitializeCredentials() (p. 415)
CredentialsFound() (p. 413)
ProxyPath() const (p. 424)

6.271.3.47 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. 424)

6.271.3.48 bool Arc::UserConfig::SaveToFile (const std::string & *filename*) const

Save to INI file.

This method will save the object data as a INI file. The saved file can be loaded with the LoadConfigurationFile method.

Parameters

filename	the name of the file which the data will be saved to.

Returns

`false` if unable to get handle on file, otherwise `true` is returned.

See also

LoadConfigurationFile() (p. 420)

6.271.3.49 void Arc::UserConfig::SetUser (const User & *u*) `[inline]`

Set **User** (p. 398) for filesystem access.

Sometimes it is desirable to use the identity of another user when accessing the filesystem. This user can be specified through this method. By default this user is the same as the user running the process.

Parameters

u	**User** (p. 398) identity to use

6.271.3.50 const URL& Arc::UserConfig::SLCS ( ) const `[inline]`

Get the **URL** (p. 387) to the Short Lived Certificate **Service** (p. 341) (SLCS).

Returns

The SLCS is returned.

See also

SLCS(const URL&) (p. 425)

6.271.3.51 bool Arc::UserConfig::SLCS (const URL & *newSLCS*) `[inline]`

Set the **URL** (p. 387) to the Short Lived Certificate **Service** (p. 341) (SLCS).

The attribute associated with this setter method is 'slcs'.

Parameters

newSLCS	is the **URL** (p. 387) to the SLCS

Returns

This method always returns `true`.

See also

SLCS() const (p. 425)

6.271.3.52 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. 100) 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

6.271.3.53 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. 100) Servide.

Returns

The path to the store directory is returned.

See also

StoreDirectory(const std::string&) (p. 426)

6.271.3.54 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. 427)
DEFAULT_TIMEOUT (p. 430)

6.271.3.55 int Arc::UserConfig::Timeout ( ) const `[inline]`

Get timeout.

Returns the timeout in seconds.

Returns

timeout in seconds.

See also

Timeout(int) (p. 426)
DEFAULT_TIMEOUT (p. 430)

6.271.3.56 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. 341).

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. 428)

6.271.3.57 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. 341).

Returns

The username is returned.

See also

UserName(const std::string&) (p. 427)

6.271.3.58 bool Arc::UserConfig::UtilsDirPath (const std::string & *dir*)

Set path to directory storing utility files for DataPoints.

Some DataPoints can store information on remote services in local files. This method sets the path to the directory containing these files. For example arc∗ tools set it to ARCUSERDIRECTORY and A-REX sets it to the control directory. The directory is created if it does not exist.

Parameters

path	is the new utils dir path.

Returns

This method always returns `true`.

6.271.3.59 const std::string& Arc::UserConfig::UtilsDirPath ( ) const `[inline]`

Get path to directory storing utility files for DataPoints.

Returns

The utils dir path

See also

UtilsDirPath(const std::string&) (p. 428)

6.271.3.60 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. 429)

6.271.3.61 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. 429)

6.271.3.62 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. 430)

6.271.3.63 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. 429)

6.271.4 Field Documentation

6.271.4.1 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()

6.271.4.2 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. 69)
Broker(const std::string&) (p. 407)
Broker(const std::string&, const std::string&) (p. 408)
Broker() const (p. 407)

6.271.4.3 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. 426).

See also

Timeout(int) (p. 426)
Timeout() const (p. 427)

6.271.4.4 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.

6.271.4.5 const std::string **Arc::UserConfig::EXAMPLECONFIG** `[static]`

Path to example configuration.

The *EXAMPLECONFIG* variable is the path to the example configuration file.

6.271.4.6 const std::string **Arc::UserConfig::SYSCONFIG** `[static]`

Path to system configuration.

The *SYSCONFIG* variable is the path to the system configuration file. This variable is only equal to SYSCONFIGARCLOC if ARC is installed in the root (highly unlikely).

6.271.4.7 const std::string **Arc::UserConfig::SYSCONFIGARCLOC** `[static]`

Path to system configuration at ARC location.

The *SYSCONFIGARCLOC* variable is the path to the system configuration file which reside at the ARC installation location.

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

- UserConfig.h

6.272 Arc::UsernameToken Class Reference

Interface for manipulation of WS-Security according to Username Token **Profile** (p. 315).

`#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)

6.272.1 Detailed Description

Interface for manipulation of WS-Security according to Username Token **Profile** (p. 315).

6.272.2 Member Enumeration Documentation

6.272.2.1 enum **Arc::UsernameToken::PasswordType**

SOAP header element

6.272.3 Constructor & Destructor Documentation

6.272.3.1 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.

6.272.3.2 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. 431) wsu:ID="...">
pwdtype	<wsse:Password Type="...">...</wsse:Password>

6.272.3.3 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. 262) Authentication Code
iteration	<wsse11:Iteration>...</wsse11:Iteration>

6.272.4 Member Function Documentation

6.272.4.1 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. 431) class using obtained derived_key.

6.272.4.2 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

6.272.4.3 Arc::UsernameToken::operator bool (void)

Returns true of constructor succeeded

6.272.4.4 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

6.273 Arc::UserSwitch Class Reference

```
#include <User.h>
```

6.273.1 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. 325) class while spawning new process.

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

- User.h

6.274 Arc::VOMSACInfo Class Reference

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

- VOMSUtil.h

6.275 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)

6.275.1 Detailed Description

Stores definitions for making decision if VOMS server is trusted

6.275.2 Constructor & Destructor Documentation

6.275.2.1 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:VOMSCertTrus
<tls:VOMSCertTrustDN>----NEXT CHAIN---</tls:VOMSCertTrustDN> <tls:VOMSCertTrustDN>/DC=ch
<tls:VOMSCertTrustDN>/DC=ch/DC=cern/CN=CERN Trusted Certification Authority</tls:VOMSCertTrustD
</tls:VOMSCertTrustDNChain> the other way is: <tls:VOMSCertTrustDNChain>
<tls:VOMSCertTrustDN>/O=Grid/O=NorduGrid/CN=host/arthur.hep.lu.se</tls:VOMSCertTrustDN>
<tls:VOMSCertTrustDN>/O=Grid/O=NorduGrid/CN=NorduGrid Certification Authority</tls:VOMSCertTrus

</tls:VOMSCertTrustDNChain> <tls:VOMSCertTrustDNChain> <tls:VOMSCertTrustDN>/DC=ch/DC=cern/OU=comp
<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) .

6.275.2.2 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. 436) and **AddRegex()** (p. 436) for more information.

6.275.3 Member Function Documentation

6.275.3.1 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.

6.275.3.2 VOMSTrustChain& Arc::VOMSTrustList::AddChain (void)

Adds empty chain of trusted DNs to list.

6.275.3.3 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

6.276 Arc::WSAEndpointReference Class Reference

Interface for manipulation of WS-Adressing Endpoint Reference.

```
#include <WSA.h>
```

Public Member Functions

- **WSAEndpointReference** (**XMLNode** epr)
- **WSAEndpointReference** (const **WSAEndpointReference** &wsa)
- **WSAEndpointReference** (const std::string &address)
- **WSAEndpointReference** (void)
- ∼**WSAEndpointReference** (void)
- std::string **Address** (void) const
- void **Address** (const std::string &uri)
- **WSAEndpointReference** & **operator=** (const std::string &address)
- **XMLNode ReferenceParameters** (void)
- **XMLNode MetaData** (void)
- **operator XMLNode** (void)

6.276.1 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.

6.276.2 Constructor & Destructor Documentation

6.276.2.1 Arc::WSAEndpointReference::WSAEndpointReference (XMLNode *epr*)

Link to top level EPR XML node Linking to existing EPR in XML tree

6.276.2.2 Arc::WSAEndpointReference::WSAEndpointReference (const WSAEndpointReference & *wsa*)

Copy constructor

6.276.2.3 Arc::WSAEndpointReference::WSAEndpointReference (const std::string & *address*)

Creating independent EPR - not implemented

6.276.2.4 Arc::WSAEndpointReference::WSAEndpointReference (void)

Dummy constructor - creates invalid instance

6.276.2.5 Arc::WSAEndpointReference::∼WSAEndpointReference (void)

Destructor. All empty elements of EPR XML are destroyed here too

6.276.3 Member Function Documentation

6.276.3.1 std::string Arc::WSAEndpointReference::Address (void) const

Returns Address (**URL** (p. 387)) encoded in EPR

6.276.3.2 void Arc::WSAEndpointReference::Address (const std::string & *uri*)

Assigns new Address value. If EPR had no Address element it is created.

6.276.3.3 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.

6.276.3.4 Arc::WSAEndpointReference::operator XMLNode (void)

Returns reference to EPR top XML node

6.276.3.5 WSAEndpointReference& Arc::WSAEndpointReference::operator= (const std::string & *address*)

Same as Address(uri)

6.276.3.6 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

6.277 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_**

6.277.1 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.

6.277.2 Constructor & Destructor Documentation

6.277.2.1 Arc::WSAHeader::WSAHeader (SOAPEnvelope & *soap*)

Linking to a header of existing SOAP message

6.277.2.2 Arc::WSAHeader::WSAHeader (const std::string & *action*)

Creating independent SOAP header - not implemented

6.277.3 Member Function Documentation

6.277.3.1 std::string Arc::WSAHeader::Action (void) const

Returns content of Action element of SOAP Header.

6.277.3.2 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.

6.277.3.3 static bool Arc::WSAHeader::Check (SOAPEnvelope & *soap*) [static]

Tells if specified SOAP message has WSA header

6.277.3.4 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.

6.277.3.5 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.

6.277.3.6 std::string Arc::WSAHeader::MessageID (void) const

Returns content of MessageID element of SOAP Header.

6.277.3.7 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.

6.277.3.8 XMLNode Arc::WSAHeader::NewReferenceParameter (const std::string & *name*)

Creates new ReferenceParameter element with specified name. Returns reference to created element.

6.277.3.9 Arc::WSAHeader::operator XMLNode (void)

Returns reference to SOAP Header - not implemented

6.277.3.10 XMLNode Arc::WSAHeader::ReferenceParameter (const std::string & *name*)

Returns first ReferenceParameter element with specified name

6.277.3.11 XMLNode Arc::WSAHeader::ReferenceParameter (int *n*)

Return n-th ReferenceParameter element

6.277.3.12 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.

6.277.3.13 std::string Arc::WSAHeader::RelatesTo (void) const

Returns content of RelatesTo element of SOAP Header.

6.277.3.14 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.

6.277.3.15 std::string Arc::WSAHeader::RelationshipType (void) const

Returns content of RelationshipType element of SOAP Header.

6.277.3.16 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.

6.277.3.17 std::string Arc::WSAHeader::To (void) const

Returns content of To element of SOAP Header.

6.277.3.18 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.

6.277.4 Field Documentation

6.277.4.1 bool **Arc::WSAHeader::header_allocated_** `[protected]`

SOAP header element

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

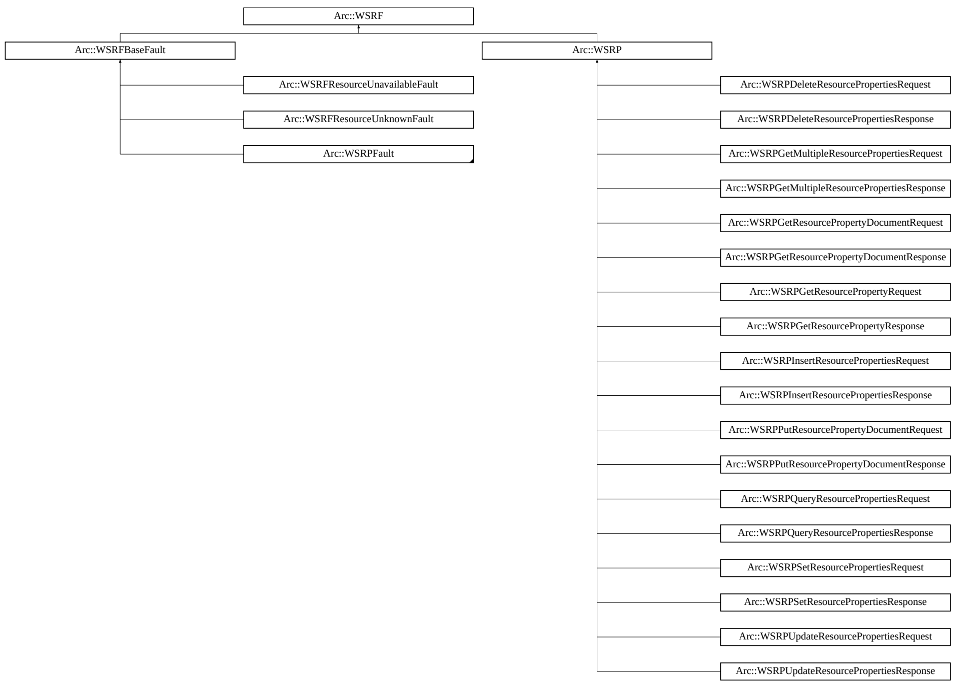
- WSA.h

6.278 Arc::WSRF Class Reference

Base class for every **WSRF** (p. 441) 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_**

6.278.1 Detailed Description

Base class for every **WSRF** (p. 441) message. This class is not intended to be used directly. Use it like reference while passing through unknown **WSRF** (p. 441) message or use classes derived from it.

6.278.2 Constructor & Destructor Documentation

6.278.2.1 Arc::WSRF::WSRF (SOAPEnvelope & *soap*, const std::string & *action* = "")

Constructor - creates object out of supplied SOAP tree.

6.278.2.2 Arc::WSRF::WSRF (bool *fault* = `false`, const std::string & *action* = "")

Constructor - creates new **WSRF** (p. 441) object

6.278.3 Member Function Documentation

6.278.3.1 virtual Arc::WSRF::operator bool (void) `[inline, virtual]`

Returns true if instance is valid

References valid_.

6.278.3.2 void Arc::WSRF::set_namespaces (void) `[protected]`

true if object represents valid **WSRF** (p. 441) message set WS Resource namespaces and default prefixes in SOAP message

Reimplemented in **Arc::WSRP** (p. 447), and **Arc::WSRFBaseFault** (p. 444).

6.278.3.3 virtual SOAPEnvelope& Arc::WSRF::SOAP (void) `[inline, virtual]`

Direct access to underlying SOAP element

6.278.4 Field Documentation

6.278.4.1 bool **Arc::WSRF::allocated_** `[protected]`

Associated SOAP message - it's SOAP message after all

6.278.4.2 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:

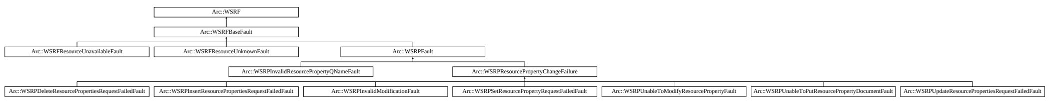
- WSRF.h

6.279 Arc::WSRFBaseFault Class Reference

Base class for **WSRF** (p. 441) 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)

6.279.1 Detailed Description

Base class for **WSRF** (p. 441) fault messages. Use classes inherited from it for specific faults.

6.279.2 Constructor & Destructor Documentation

6.279.2.1 Arc::WSRFBaseFault::WSRFBaseFault (SOAPEnvelope & *soap*)

Constructor - creates object out of supplied SOAP tree.

6.279.2.2 Arc::WSRFBaseFault::WSRFBaseFault (const std::string & *type*)

Constructor - creates new **WSRF** (p. 441) fault

6.279.3 Member Function Documentation

6.279.3.1 void Arc::WSRFBaseFault::set_namespaces (void) `[protected]`

set WS-ResourceProperties namespaces and default prefixes in SOAP message

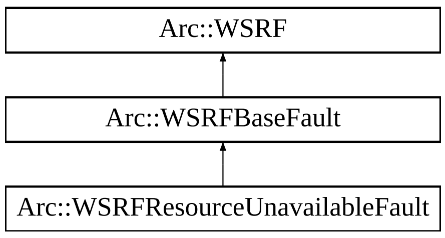
Reimplemented from **Arc::WSRF** (p. 442).

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

- WSRFBaseFault.h

6.280 Arc::WSRFResourceUnavailableFault Class Reference

Inheritance diagram for Arc::WSRFResourceUnavailableFault:

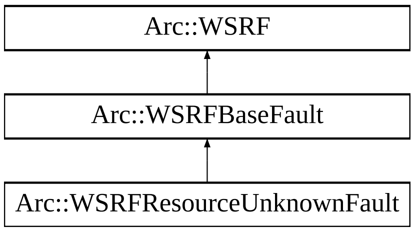


The documentation for this class was generated from the following file:

- WSRFBaseFault.h

6.281 Arc::WSRFResourceUnknownFault Class Reference

Inheritance diagram for Arc::WSRFResourceUnknownFault:



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

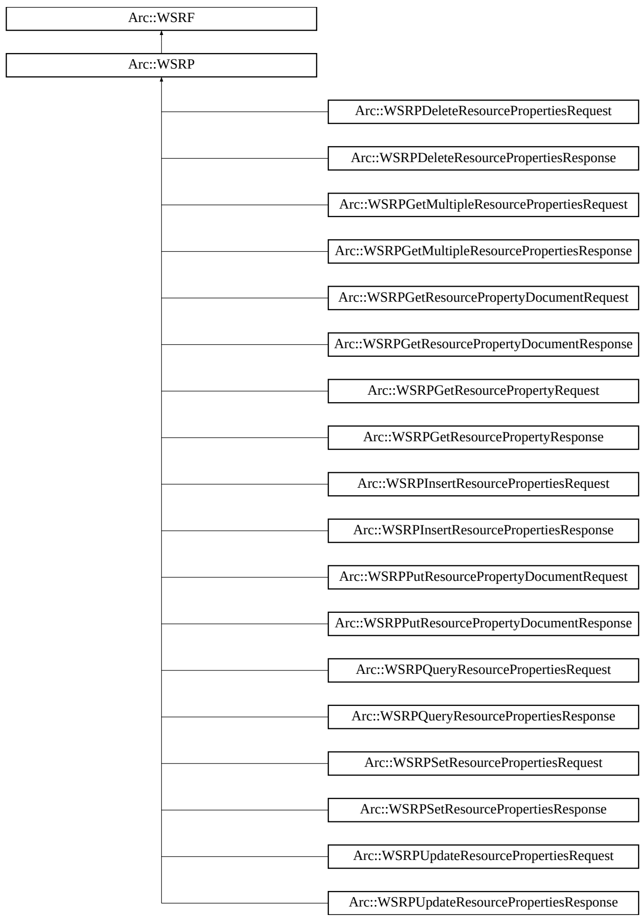
- WSRFBaseFault.h

6.282 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)

6.282.1 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.

6.282.2 Constructor & Destructor Documentation

6.282.2.1 Arc::WSRP::WSRP (bool *fault* = `false`, const std::string & *action* = "")

Constructor - prepares object for creation of new **WSRP** (p. 445) request/response/fault

6.282.2.2 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.

6.282.3 Member Function Documentation

6.282.3.1 void Arc::WSRP::set_namespaces (void) `[protected]`

set WS-ResourceProperties namespaces and default prefixes in SOAP message

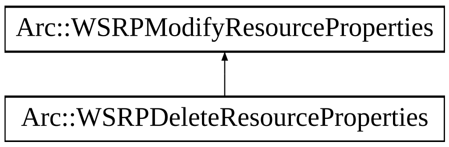
Reimplemented from **Arc::WSRF** (p. 442).

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

- WSResourceProperties.h

6.283 Arc::WSRPDeleteResourceProperties Class Reference

Inheritance diagram for Arc::WSRPDeleteResourceProperties:

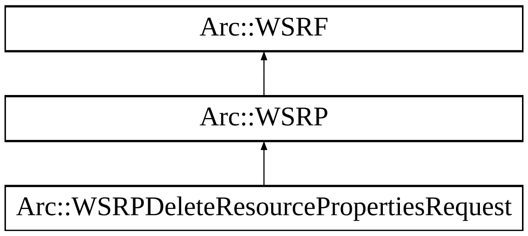

Arc::WSRPModifyResourceProperties
Arc::WSRPDeleteResourceProperties

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

- WSResourceProperties.h

6.284 Arc::WSRPDeleteResourcePropertiesRequest Class Reference

Inheritance diagram for Arc::WSRPDeleteResourcePropertiesRequest:

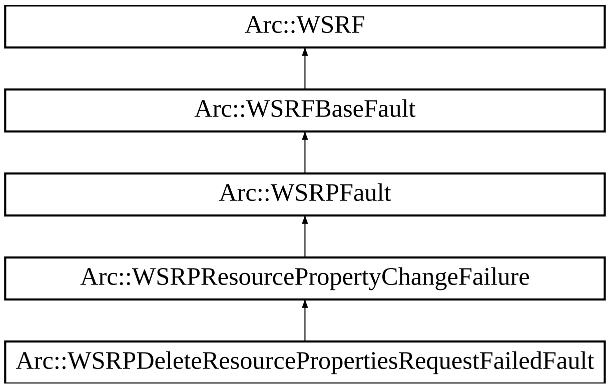


The documentation for this class was generated from the following file:

- WSResourceProperties.h

6.285 Arc::WSRPDeleteResourcePropertiesRequestFailedFault Class Reference

Inheritance diagram for Arc::WSRPDeleteResourcePropertiesRequestFailedFault:

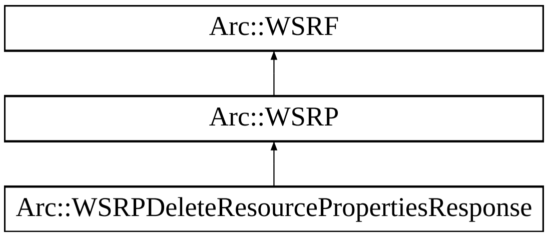

Arc::WSRF
Arc::WSRFBaseFault
Arc::WSRPFault
Arc::WSRPResourcePropertyChangeFailure
Arc::WSRPDeleteResourcePropertiesRequestFailedFault

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

- WSResourceProperties.h

6.286 Arc::WSRPDeleteResourcePropertiesResponse Class Reference

Inheritance diagram for Arc::WSRPDeleteResourcePropertiesResponse:



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

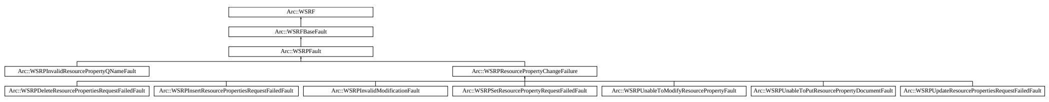
- WSResourceProperties.h

6.287 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)

6.287.1 Detailed Description

Base class for WS-ResourceProperties faults.

6.287.2 Constructor & Destructor Documentation

6.287.2.1 Arc::WSRPFault::WSRPFault (SOAPEnvelope & *soap*)

Constructor - creates object out of supplied SOAP tree.

6.287.2.2 Arc::WSRPFault::WSRPFault (const std::string & *type*)

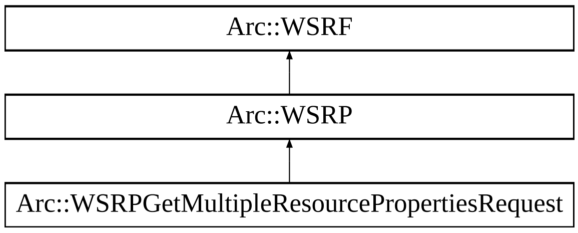
Constructor - creates new **WSRP** (p. 445) fault

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

- WSResourceProperties.h

6.288 Arc::WSRPGetMultipleResourcePropertiesRequest Class Reference

Inheritance diagram for Arc::WSRPGetMultipleResourcePropertiesRequest:

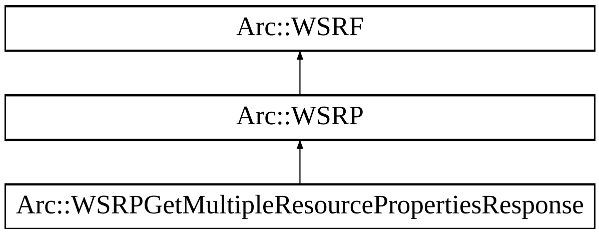


The documentation for this class was generated from the following file:

- WSResourceProperties.h

6.289 Arc::WSRPGetMultipleResourcePropertiesResponse Class Reference

Inheritance diagram for Arc::WSRPGetMultipleResourcePropertiesResponse:

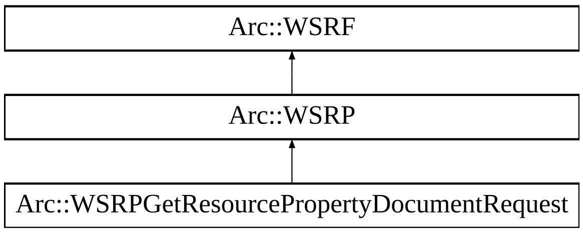


The documentation for this class was generated from the following file:

- WSResourceProperties.h

6.290 Arc::WSRPGetResourcePropertyDocumentRequest Class Reference

Inheritance diagram for Arc::WSRPGetResourcePropertyDocumentRequest:

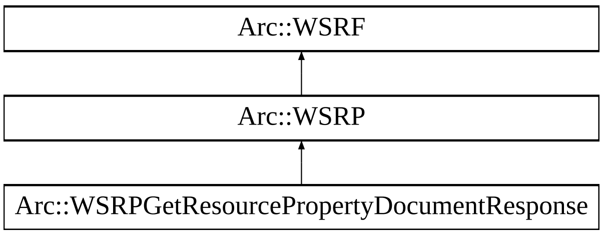


The documentation for this class was generated from the following file:

- WSResourceProperties.h

6.291 Arc::WSRPGetResourcePropertyDocumentResponse Class Reference

Inheritance diagram for Arc::WSRPGetResourcePropertyDocumentResponse:

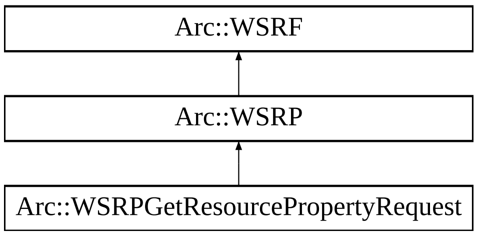


The documentation for this class was generated from the following file:

- WSResourceProperties.h

6.292 Arc::WSRPGetResourcePropertyRequest Class Reference

Inheritance diagram for Arc::WSRPGetResourcePropertyRequest:

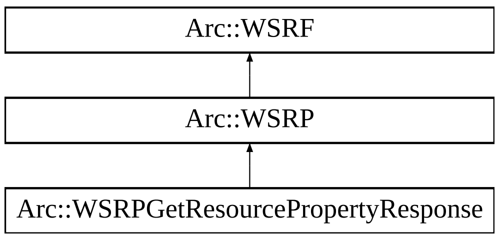


The documentation for this class was generated from the following file:

- WSResourceProperties.h

6.293 Arc::WSRPGetResourcePropertyResponse Class Reference

Inheritance diagram for Arc::WSRPGetResourcePropertyResponse:

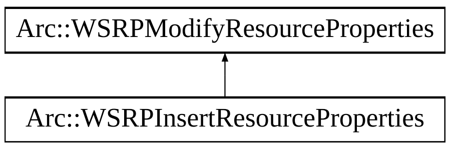


The documentation for this class was generated from the following file:

- WSResourceProperties.h

6.294 Arc::WSRPInsertResourceProperties Class Reference

Inheritance diagram for Arc::WSRPInsertResourceProperties:

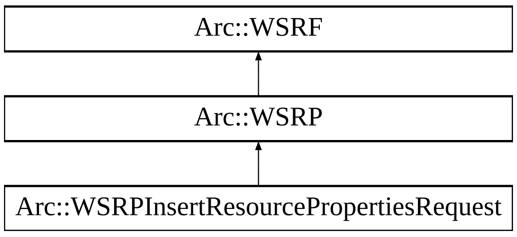

Arc::WSRPModifyResourceProperties
Arc::WSRPInsertResourceProperties

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

- WSResourceProperties.h

6.295 Arc::WSRPInsertResourcePropertiesRequest Class Reference

Inheritance diagram for Arc::WSRPInsertResourcePropertiesRequest:

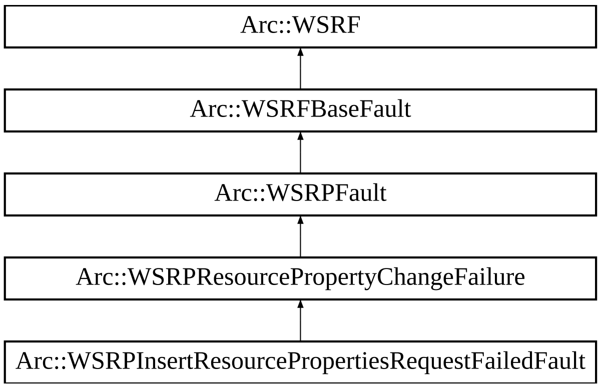


The documentation for this class was generated from the following file:

- WSResourceProperties.h

6.296 Arc::WSRPInsertResourcePropertiesRequestFailedFault Class Reference

Inheritance diagram for Arc::WSRPInsertResourcePropertiesRequestFailedFault:

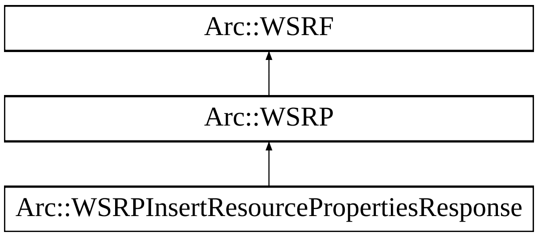

Arc::WSRF
Arc::WSRFBaseFault
Arc::WSRPFault
Arc::WSRPResourcePropertyChangeFailure
Arc::WSRPInsertResourcePropertiesRequestFailedFault

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

- WSResourceProperties.h

6.297 Arc::WSRPInsertResourcePropertiesResponse Class Reference

Inheritance diagram for Arc::WSRPInsertResourcePropertiesResponse:

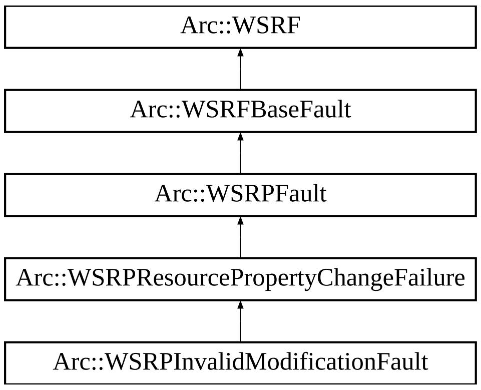


The documentation for this class was generated from the following file:

- WSResourceProperties.h

6.298 Arc::WSRPInvalidModificationFault Class Reference

Inheritance diagram for Arc::WSRPInvalidModificationFault:

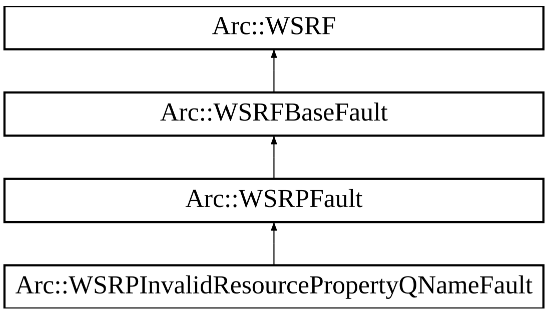

Arc::WSRF
Arc::WSRFBaseFault
Arc::WSRPFault
Arc::WSRPResourcePropertyChangeFailure
Arc::WSRPInvalidModificationFault

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

- WSResourceProperties.h

6.299 Arc::WSRPInvalidResourcePropertyQNameFault Class Reference

Inheritance diagram for Arc::WSRPInvalidResourcePropertyQNameFault:

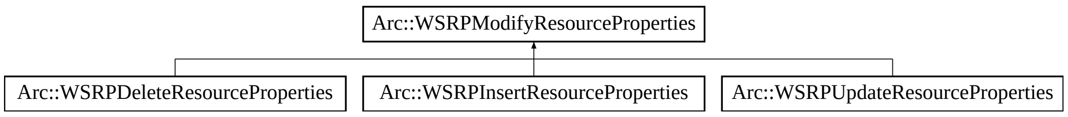


The documentation for this class was generated from the following file:

- WSResourceProperties.h

6.300 Arc::WSRPModifyResourceProperties Class Reference

Inheritance diagram for Arc::WSRPModifyResourceProperties:

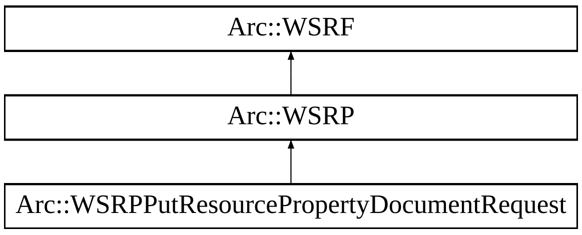

Arc::WSRPModifyResourceProperties
Arc::WSRPDeleteResourceProperties
Arc::WSRPInsertResourceProperties
Arc::WSRPUpdateResourceProperties

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

- WSResourceProperties.h

6.301 Arc::WSRPPutResourcePropertyDocumentRequest Class Reference

Inheritance diagram for Arc::WSRPPutResourcePropertyDocumentRequest:

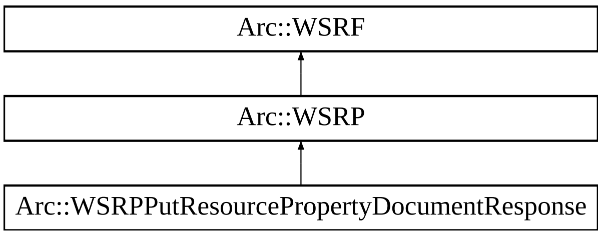


The documentation for this class was generated from the following file:

- WSResourceProperties.h

6.302 Arc::WSRPPutResourcePropertyDocumentResponse Class Reference

Inheritance diagram for Arc::WSRPPutResourcePropertyDocumentResponse:

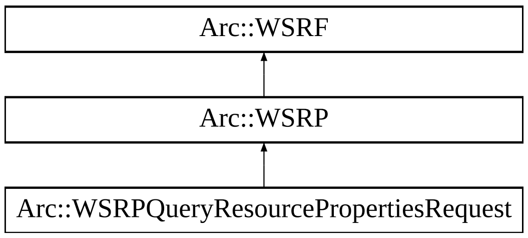


The documentation for this class was generated from the following file:

- WSResourceProperties.h

6.303 Arc::WSRPQueryResourcePropertiesRequest Class Reference

Inheritance diagram for Arc::WSRPQueryResourcePropertiesRequest:

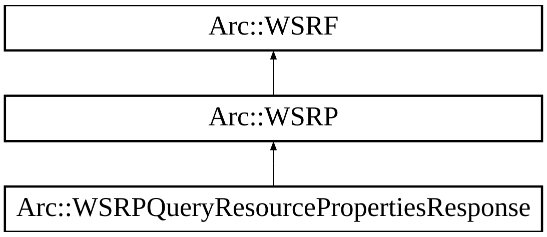


The documentation for this class was generated from the following file:

- WSResourceProperties.h

6.304 Arc::WSRPQueryResourcePropertiesResponse Class Reference

Inheritance diagram for Arc::WSRPQueryResourcePropertiesResponse:



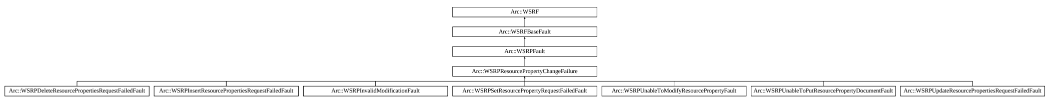
The documentation for this class was generated from the following file:

- WSResourceProperties.h

6.305 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)

6.305.1 Detailed Description

Base class for WS-ResourceProperties faults which contain ResourcePropertyChange-Failure

6.305.2 Constructor & Destructor Documentation

6.305.2.1 Arc::WSRPResourcePropertyChangeFailure::WSRPResourcePropertyChangeFailure (SOAPEnvelope & *soap*) `[inline]`

Constructor - creates object out of supplied SOAP tree.

6.305.2.2 Arc::WSRPResourcePropertyChangeFailure::WSRPResourcePropertyChangeFailure (const std::string & *type*) `[inline]`

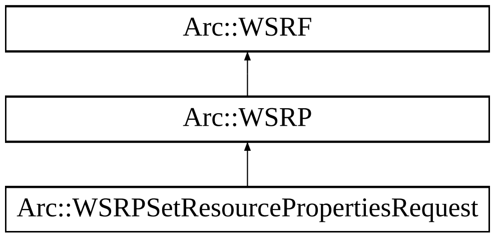
Constructor - creates new **WSRP** (p. 445) fault

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

- WSResourceProperties.h

6.306 Arc::WSRPSetResourcePropertiesRequest Class Reference

Inheritance diagram for Arc::WSRPSetResourcePropertiesRequest:

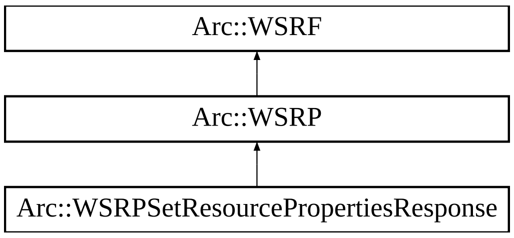


The documentation for this class was generated from the following file:

- WSResourceProperties.h

6.307 Arc::WSRPSetResourcePropertiesResponse Class Reference

Inheritance diagram for Arc::WSRPSetResourcePropertiesResponse:

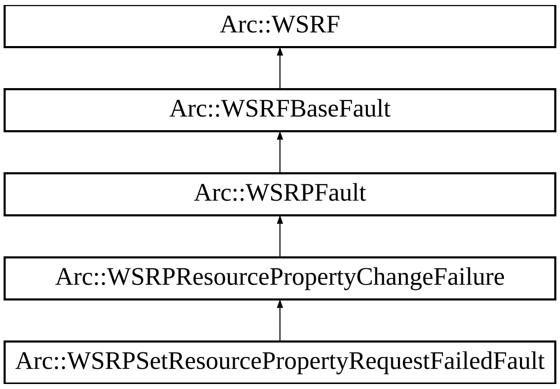


The documentation for this class was generated from the following file:

- WSResourceProperties.h

6.308 Arc::WSRPSetResourcePropertyRequestFailedFault Class Reference

Inheritance diagram for Arc::WSRPSetResourcePropertyRequestFailedFault:

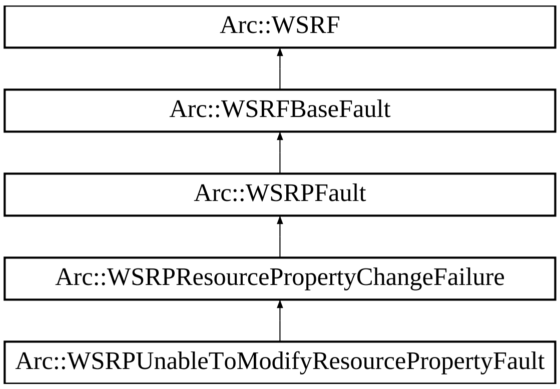

Arc::WSRF
Arc::WSRFBaseFault
Arc::WSRPFault
Arc::WSRPResourcePropertyChangeFailure
Arc::WSRPSetResourcePropertyRequestFailedFault

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

- WSResourceProperties.h

6.309 Arc::WSRPUnableToModifyResourcePropertyFault Class Reference

Inheritance diagram for Arc::WSRPUnableToModifyResourcePropertyFault:

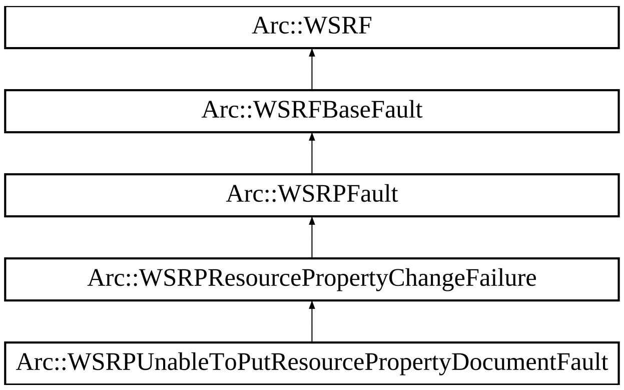

Arc::WSRF
Arc::WSRFBaseFault
Arc::WSRPFault
Arc::WSRPResourcePropertyChangeFailure
Arc::WSRPUnableToModifyResourcePropertyFault

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

- WSResourceProperties.h

6.310 Arc::WSRPUnableToPutResourcePropertyDocumentFault Class Reference

Inheritance diagram for Arc::WSRPUnableToPutResourcePropertyDocumentFault:

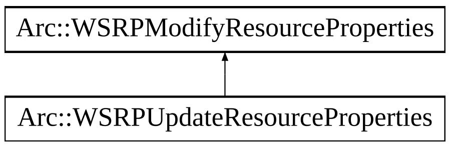

Arc::WSRF
Arc::WSRFBaseFault
Arc::WSRPFault
Arc::WSRPResourcePropertyChangeFailure
Arc::WSRPUnableToPutResourcePropertyDocumentFault

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

- WSResourceProperties.h

6.311 Arc::WSRPUpdateResourceProperties Class Reference

Inheritance diagram for Arc::WSRPUpdateResourceProperties:

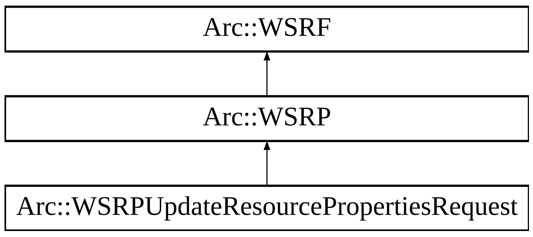

Arc::WSRPModifyResourceProperties
Arc::WSRPUpdateResourceProperties

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

- WSResourceProperties.h

6.312 Arc::WSRPUpdateResourcePropertiesRequest Class Reference

Inheritance diagram for Arc::WSRPUpdateResourcePropertiesRequest:

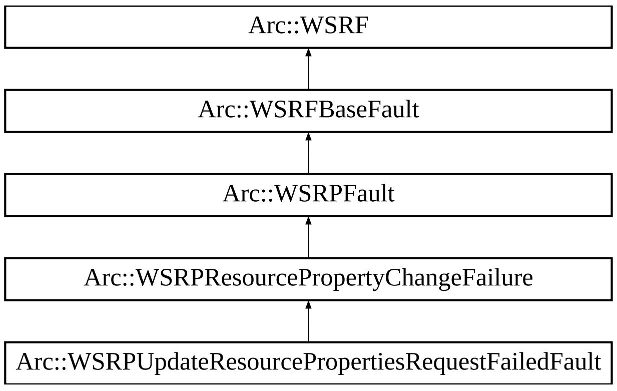


The documentation for this class was generated from the following file:

- WSResourceProperties.h

6.313 Arc::WSRPUpdateResourcePropertiesRequestFailedFault Class Reference

Inheritance diagram for Arc::WSRPUpdateResourcePropertiesRequestFailedFault:

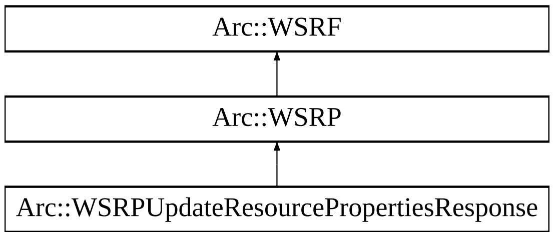

Arc::WSRF
Arc::WSRFBaseFault
Arc::WSRPFault
Arc::WSRPResourcePropertyChangeFailure
Arc::WSRPUpdateResourcePropertiesRequestFailedFault

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

- WSResourceProperties.h

6.314 Arc::WSRPUpdateResourcePropertiesResponse Class Reference

Inheritance diagram for Arc::WSRPUpdateResourcePropertiesResponse:

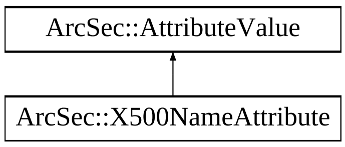


The documentation for this class was generated from the following file:

- WSResourceProperties.h

6.315 ArcSec::X500NameAttribute Class Reference

Inheritance diagram for ArcSec::X500NameAttribute:



Public Member Functions

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

6.315.1 Member Function Documentation

6.315.1.1 virtual std::string ArcSec::X500NameAttribute::encode ( ) `[inline, virtual]`

encode the value in a string format

Implements **ArcSec::AttributeValue** (p. 64).

6.315.1.2 virtual bool ArcSec::X500NameAttribute::equal (**AttributeValue** ∗ *value*, bool *check_id* = `true`) `[virtual]`

Evluate whether "this" equale to the parameter value

Implements **ArcSec::AttributeValue** (p. 64).

6.315.1.3 virtual std::string ArcSec::X500NameAttribute::getId ( ) `[inline, virtual]`

Get the AttributeId of the <Attribute>

Implements **ArcSec::AttributeValue** (p. 64).

6.315.1.4 virtual std::string ArcSec::X500NameAttribute::getType ( ) `[inline, virtual]`

Get the DataType of the <Attribute>

Implements **ArcSec::AttributeValue** (p. 65).

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

- X500NameAttribute.h

6.316 Arc::X509Token Class Reference

Class for manipulating X.509 Token **Profile** (p. 315).

```
#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)

6.316.1 Detailed Description

Class for manipulating X.509 Token **Profile** (p. 315). 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.

6.316.2 Member Enumeration Documentation

6.316.2.1 enum Arc::X509Token::X509TokenType

X509TokeType is for distinguishing two types of operation. It is used as the parameter of constuctor.

6.316.3 Constructor & Destructor Documentation

6.316.3.1 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. 463) 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. 463) is encryption token

6.316.3.2 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.

6.316.3.3 Arc::X509Token::∼X509Token (void)

Deconstructor. Nothing to be done except finalizing the xmlsec library.

6.316.4 Member Function Documentation

6.316.4.1 bool Arc::X509Token::Authenticate (const std::string & *cafile*, const std::string & *capath*)

Check signature by using the certificare information in **X509Token** (p. 463) 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. 463)) 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

6.316.4.2 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.

6.316.4.3 Arc::X509Token::operator bool (void)

Returns true of constructor succeeded

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

- X509Token.h

6.317 Arc::XmlContainer Class Reference

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

- XmlContainer.h

6.318 Arc::XmlDatabase Class Reference

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

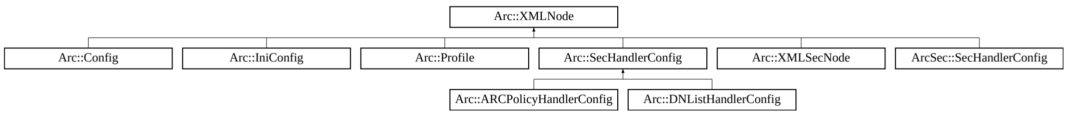
- XmlDatabase.h

6.319 Arc::XMLNode Class Reference

Wrapper for LibXML library Tree interface.

`#include <XMLNode.h>`

Inheritance diagram for Arc::XMLNode:



Public Member Functions

- **XMLNode** (void)
- **XMLNode** (const **XMLNode** &node)
- **XMLNode** (const std::string &xml)

- **XMLNode** (const char ∗xml, int len=-1)
- **XMLNode** (long ptr_addr)
- **XMLNode** (const **NS** &ns, const char ∗name)
- ∼**XMLNode** (void)
- void **New** (**XMLNode** &node) const
- void **Exchange** (**XMLNode** &node)
- void **Move** (**XMLNode** &node)
- void **Swap** (**XMLNode** &node)
- **operator bool** (void) const
- bool **operator!** (void) const
- bool **operator==** (const **XMLNode** &node)
- bool **operator!=** (const **XMLNode** &node)
- bool **Same** (const **XMLNode** &node)
- bool **operator==** (bool val)
- bool **operator!=** (bool val)
- bool **operator==** (const std::string &str)
- bool **operator!=** (const std::string &str)
- bool **operator==** (const char ∗str)
- bool **operator!=** (const char ∗str)
- **XMLNode Child** (int n=0)
- **XMLNode operator[ ]** (const char ∗name) const
- **XMLNode operator[ ]** (const std::string &name) const
- **XMLNode operator[ ]** (int n) const
- void **operator++** (void)
- void **operator--** (void)
- int **Size** (void) const
- **XMLNode Get** (const std::string &name) const
- std::string **Name** (void) const
- std::string **Prefix** (void) const
- std::string **FullName** (void) const
- std::string **Namespace** (void) const
- void **Name** (const char ∗name)
- void **Name** (const std::string &name)
- void **GetXML** (std::string &out_xml_str, bool user_friendly=false) const
- void **GetXML** (std::string &out_xml_str, const std::string &encoding, bool user_-friendly=false) const
- void **GetDoc** (std::string &out_xml_str, bool user_friendly=false) const
- **operator std::string** (void) const
- **XMLNode** & **operator=** (const char ∗content)
- **XMLNode** & **operator=** (const std::string &content)
- void **Set** (const std::string &content)
- **XMLNode** & **operator=** (const **XMLNode** &node)
- **XMLNode Attribute** (int n=0)
- **XMLNode Attribute** (const char ∗name)
- **XMLNode Attribute** (const std::string &name)
- **XMLNode NewAttribute** (const char ∗name)

- **XMLNode NewAttribute** (const std::string &name)
- int **AttributesSize** (void) const
- void **Namespaces** (const **NS** &namespaces, bool keep=false, int recursion=-1)
- **NS Namespaces** (void)
- std::string **NamespacePrefix** (const char ∗urn)
- **XMLNode NewChild** (const char ∗name, int n=-1, bool global_order=false)
- **XMLNode NewChild** (const std::string &name, int n=-1, bool global_order=false)
- **XMLNode NewChild** (const char ∗name, const **NS** &namespaces, int n=-1, bool global_order=false)
- **XMLNode NewChild** (const std::string &name, const **NS** &namespaces, int n=-1, bool global_order=false)
- **XMLNode NewChild** (const **XMLNode** &node, int n=-1, bool global_order=false)
- void **Replace** (const **XMLNode** &node)
- void **Destroy** (void)
- XMLNodeList **Path** (const std::string &path)
- XMLNodeList **XPathLookup** (const std::string &xpathExpr, const **NS** &nsList)
- **XMLNode GetRoot** (void)
- **XMLNode Parent** (void)
- bool **SaveToFile** (const std::string &file_name) const
- bool **SaveToStream** (std::ostream &out) const
- bool **ReadFromFile** (const std::string &file_name)
- bool **ReadFromStream** (std::istream &in)
- bool **Validate** (const std::string &schema_file, std::string &err_msg)

Protected Member Functions

- **XMLNode** (xmlNodePtr node)

Protected Attributes

- bool **is_owner_**
- bool **is_temporary_**

Friends

- bool **MatchXMLName** (const **XMLNode** &node1, const **XMLNode** &node2)
- bool **MatchXMLName** (const **XMLNode** &node, const char ∗name)
- bool **MatchXMLName** (const **XMLNode** &node, const std::string &name)
- bool **MatchXMLNamespace** (const **XMLNode** &node1, const **XMLNode** &node2)
- bool **MatchXMLNamespace** (const **XMLNode** &node, const char ∗uri)
- bool **MatchXMLNamespace** (const **XMLNode** &node, const std::string &uri)

6.319.1 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.

6.319.2 Constructor & Destructor Documentation

6.319.2.1 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.

6.319.2.2 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.

6.319.2.3 Arc::XMLNode::XMLNode (const XMLNode & *node*) `[inline]`

Copies existing instance. Underlying XML element is NOT copied. Ownership is NOT inherited. Strictly speaking it shuld be no const here - but that conflicts with C++.

6.319.2.4 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

6.319.2.5 Arc::XMLNode::XMLNode (const char * *xml*, int *len* = `-1`)

Same as previous

6.319.2.6 Arc::XMLNode::XMLNode (long *ptr_addr*)

Copy constructor. Used by language bindigs

6.319.2.7 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

6.319.2.8 Arc::XMLNode::∼XMLNode (void)

Destructor Also destroys underlying XML document if owned by this instance

6.319.3 Member Function Documentation

6.319.3.1 XMLNode Arc::XMLNode::Attribute (int *n* = 0)

Returns list of all attributes of node.

Returns **XMLNode** (p. 465) instance reresenting n-th attribute of node.

Referenced by Attribute().

6.319.3.2 XMLNode Arc::XMLNode::Attribute (const char ∗ *name*)

Returns **XMLNode** (p. 465) instance representing first attribute of node with specified by name

6.319.3.3 XMLNode Arc::XMLNode::Attribute (const std::string & *name*) [inline]

Returns **XMLNode** (p. 465) instance representing first attribute of node with specified by name

References Attribute().

6.319.3.4 int Arc::XMLNode::AttributesSize (void) const

Returns number of attributes of node

6.319.3.5 XMLNode Arc::XMLNode::Child (int *n* = 0)

Returns **XMLNode** (p. 465) instance representing n-th child of XML element. If such does not exist invalid **XMLNode** (p. 465) instance is returned

6.319.3.6 void Arc::XMLNode::Destroy (void)

Destroys underlying XML element. XML element is unlinked from XML tree and destroyed. After this operation **XMLNode** (p. 465) instance becomes invalid

6.319.3.7 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 documments. 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. 476).

6.319.3.8 std::string Arc::XMLNode::FullName (void) const `[inline]`

Returns prefix:name of XML node

References Name(), and Prefix().

6.319.3.9 XMLNode Arc::XMLNode::Get (const std::string & *name*) const `[inline]`

Same as operator[]

References operator[ ]().

6.319.3.10 void Arc::XMLNode::GetDoc (std::string & *out_xml_str*, bool *user_friendly* = `false`) const

Fills argument with whole XML document textual representation

6.319.3.11 XMLNode Arc::XMLNode::GetRoot (void)

Get the root node from any child node of the tree

6.319.3.12 void Arc::XMLNode::GetXML (std::string & *out_xml_str*, bool *user_friendly* = `false`) const

Fills argument with this instance XML subtree textual representation

6.319.3.13 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

6.319.3.14 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. 472) and **Destroy()** (p. 469) 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.

6.319.3.15 std::string Arc::XMLNode::Name (void) const

Returns name of XML node

Referenced by FullName(), and Name().

6.319.3.16 void Arc::XMLNode::Name (const std::string & *name*) `[inline]`

Assigns new name to XML node

References Name().

6.319.3.17 void Arc::XMLNode::Name (const char ∗ *name*)

Assigns new name to XML node

6.319.3.18 std::string Arc::XMLNode::Namespace (void) const

Returns namespace URI of XML node

6.319.3.19 std::string Arc::XMLNode::NamespacePrefix (const char ∗ *urn*)

Returns prefix of specified namespace. Empty string if no such namespace.

6.319.3.20 NS Arc::XMLNode::Namespaces (void)

Returns namespaces known at this node

6.319.3.21 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.

6.319.3.22 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.

6.319.3.23 XMLNode Arc::XMLNode::NewAttribute (const char ∗ *name*)

Creates new attribute with specified name.

Referenced by NewAttribute().

6.319.3.24 XMLNode Arc::XMLNode::NewAttribute (const std::string & *name*) `[inline]`

Creates new attribute with specified name.

References NewAttribute().

6.319.3.25 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().

6.319.3.26 XMLNode Arc::XMLNode::NewChild (const std::string & *name*, int *n* = `-1`, bool *global_order* = `false`) `[inline]`

Same as **NewChild(const char∗,int,bool)** (p. 472)

References NewChild().

6.319.3.27 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. 472)

6.319.3.28 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. 472)

References NewChild().

6.319.3.29 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

6.319.3.30 Arc::XMLNode::operator bool (void) const `[inline]`

Returns true if instance points to XML element - valid instance

References is_temporary_.

6.319.3.31 Arc::XMLNode::operator std::string (void) const

Returns textual content of node excluding content of children nodes

6.319.3.32 bool Arc::XMLNode::operator! (void) const `[inline]`

Returns true if instance does not point to XML element - invalid instance

References is_temporary_.

6.319.3.33 bool Arc::XMLNode::operator!= (const XMLNode & *node*) `[inline]`

Returns false if 'node' represents same XML element

6.319.3.34 bool Arc::XMLNode::operator!= (bool *val*) `[inline]`

This operator is needed to avoid ambiguity

6.319.3.35 bool Arc::XMLNode::operator!= (const std::string & *str*) `[inline]`

This operator is needed to avoid ambiguity

6.319.3.36 bool Arc::XMLNode::operator!= (const char ∗ *str*) `[inline]`

This operator is needed to avoid ambiguity

6.319.3.37 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.

6.319.3.38 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.

6.319.3.39 XMLNode& Arc::XMLNode::operator= (const char ∗ *content*)

Sets textual content of node. All existing children nodes are discarded.

Referenced by operator=(), and Set().

6.319.3.40 XMLNode& Arc::XMLNode::operator= (const XMLNode & *node*)

Make instance refer to another XML node. Ownership is not inherited. Due to nature of **XMLNode** (p. 465) there should be no const here, but that does not fit into C++.

6.319.3.41 XMLNode& Arc::XMLNode::operator= (const std::string & *content*) `[inline]`

Sets textual content of node. All existing children nodes are discarded.

References operator=().

6.319.3.42 bool Arc::XMLNode::operator== (bool *val*) `[inline]`

This operator is needed to avoid ambiguity

6.319.3.43 bool Arc::XMLNode::operator== (const XMLNode & *node*) `[inline]`

Returns true if 'node' represents same XML element

Referenced by Same().

6.319.3.44 bool Arc::XMLNode::operator== (const char ∗ *str*) `[inline]`

This operator is needed to avoid ambiguity

6.319.3.45 bool Arc::XMLNode::operator== (const std::string & *str*) `[inline]`

This operator is needed to avoid ambiguity

6.319.3.46 XMLNode Arc::XMLNode::operator[] (const char ∗ *name*) const

Returns **XMLNode** (p. 465) instance representing first child element with specified name. Name may be "namespace_prefix:name", "namespace_uri:name" or simply "name". In last case namespace is ignored. If such node does not exist invalid **XMLNode** (p. 465) instance is returned. This method should not be marked const because obtaining unrestricted **XMLNode** (p. 465) of child element allows modification of underlying XML tree. But in order to keep const in other places non-const-handling is passed to programmer. Otherwise C++ compiler goes nuts.

Referenced by Get(), and operator[ ]().

6.319.3.47 XMLNode Arc::XMLNode::operator[] (const std::string & *name*) const [inline]

Similar to previous method

References operator[ ]().

6.319.3.48 XMLNode Arc::XMLNode::operator[] (int *n*) const

Returns **XMLNode** (p. 465) instance representing n-th node in sequence of siblings of same name. It's main purpose is to be used to retrieve element in array of children of same name like node["name"][5]. This method should not be marked const because obtaining unrestricted **XMLNode** (p. 465) of child element allows modification of underlying XML tree. But in order to keep const in other places non-const-handling is passed to programmer. Otherwise C++ compiler goes nuts.

6.319.3.49 XMLNode Arc::XMLNode::Parent (void)

Get the parent node from any child node of the tree

6.319.3.50 XMLNodeList Arc::XMLNode::Path (const std::string & *path*)

Collects nodes corresponding to specified path. This is a convenience function to cover common use of XPath but without performance hit. Path is made of node_name[/node_name[...]] and is relative to current node. node_names are treated in same way as in operator[]. Returns all nodes which are represented by path.

6.319.3.51 std::string Arc::XMLNode::Prefix (void) const

Returns namespace prefix of XML node

Referenced by FullName().

6.319.3.52 bool Arc::XMLNode::ReadFromFile (const std::string & *file_name*)

Read XML document from file and associate it with this node

6.319.3.53 bool Arc::XMLNode::ReadFromStream (std::istream & *in*)

Read XML document from stream and associate it with this node

6.319.3.54 void Arc::XMLNode::Replace (const XMLNode & *node*)

Makes a copy of supplied XML node and makes this instance refere to it

6.319.3.55 bool Arc::XMLNode::Same (const XMLNode & *node*) [inline]

Returns true if 'node' represents same XML element - for bindings

References operator==().

6.319.3.56 bool Arc::XMLNode::SaveToFile (const std::string & *file_name*) const

Save string representation of node to file

6.319.3.57 bool Arc::XMLNode::SaveToStream (std::ostream & *out*) const

Save string representation of node to stream

6.319.3.58 void Arc::XMLNode::Set (const std::string & *content*) [inline]

Same as operator=. Used for bindings.

References operator=().

6.319.3.59 int Arc::XMLNode::Size (void) const

Returns number of children nodes

6.319.3.60 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. 465) 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. 476) method.

6.319.3.61 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

6.319.3.62 XMLNodeList Arc::XMLNode::XPathLookup (const std::string & *xpathExpr,* const NS & *nsList*)

Uses xPath to look up the whole xml structure, Returns a list of **XMLNode** (p. 465) 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. 316) is run on whole XML document but only the elements belonging to this XML subtree are returned.

6.319.4 Friends And Related Function Documentation

6.319.4.1 bool MatchXMLName (const XMLNode & *node1,* const XMLNode & *node2*) `[friend]`

Returns true if underlying XML elements have same names

6.319.4.2 bool MatchXMLName (const XMLNode & *node,* const std::string & *name*) `[friend]`

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

6.319.4.3 bool MatchXMLName (const XMLNode & *node,* const char ∗ *name*) `[friend]`

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

6.319.4.4 bool MatchXMLNamespace (const XMLNode & *node1,* const XMLNode & *node2*) `[friend]`

Returns true if underlying XML elements belong to same namespaces

6.319.4.5 bool MatchXMLNamespace (const XMLNode & *node,* const std::string & *uri*) `[friend]`

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

6.319.4.6 bool MatchXMLNamespace (const **XMLNode** & *node*, const char ∗ *uri*) `[friend]`

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

6.319.5 Field Documentation

6.319.5.1 bool **Arc::XMLNode::is_owner_** `[protected]`

If true node is owned by this instance - hence released in destructor. Normally that may be true only for top level node of XML document.

6.319.5.2 bool **Arc::XMLNode::is_temporary_** `[protected]`

This variable is for future

Referenced by operator bool(), and operator!().

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

- XMLNode.h

6.320 Arc::XMLNodeContainer Class Reference

`#include <XMLNode.h>`

Public Member Functions

- **XMLNodeContainer** (void)
- **XMLNodeContainer** (const **XMLNodeContainer** &)
- **XMLNodeContainer** & **operator=** (const **XMLNodeContainer** &)
- void **Add** (const **XMLNode** &)
- void **Add** (const std::list< **XMLNode** > &)
- void **AddNew** (const **XMLNode** &)
- void **AddNew** (const std::list< **XMLNode** > &)
- int **Size** (void) const
- **XMLNode operator[ ]** (int)
- std::list< **XMLNode** > **Nodes** (void)

6.320.1 Detailed Description

Container for multiple **XMLNode** (p. 465) elements

6.320.2 Constructor & Destructor Documentation

6.320.2.1 Arc::XMLNodeContainer::XMLNodeContainer (void)

Default constructor

6.320.2.2 Arc::XMLNodeContainer::XMLNodeContainer (const XMLNodeContainer &)

Copy constructor. Add nodes from argument. Nodes owning XML document are copied using **AddNew()** (p. 479). Not owning nodes are linked using **Add()** (p. 479) method.

6.320.3 Member Function Documentation

6.320.3.1 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.

6.320.3.2 void Arc::XMLNodeContainer::Add (const std::list< XMLNode > &)

Link multiple XML subtrees to container.

6.320.3.3 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.

6.320.3.4 void Arc::XMLNodeContainer::AddNew (const std::list< XMLNode > &)

Copy multiple XML subtrees to container.

6.320.3.5 std::list<XMLNode> Arc::XMLNodeContainer::Nodes (void)

Returns all stored nodes.

6.320.3.6 XMLNodeContainer& Arc::XMLNodeContainer::operator= (const XMLNodeContainer &)

Same as copy constructor with current nodes being deleted first.

6.320.3.7 XMLNode Arc::XMLNodeContainer::operator[] (int)

Returns n-th node in a store.

6.320.3.8 int Arc::XMLNodeContainer::Size (void) const

Return number of refered/stored nodes.

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

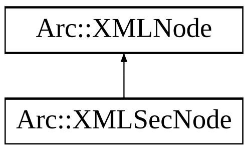
- XMLNode.h

6.321 Arc::XMLSecNode Class Reference

Extends **XMLNode** (p. 465) 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)

6.321.1 Detailed Description

Extends **XMLNode** (p. 465) class to support XML security operation. All **XMLNode** (p. 465) methods are exposed by inheriting from **XMLNode** (p. 465). **XMLSecNode** (p. 480) itself does not own node, instead it uses the node from the base class **XMLN-ode** (p. 465).

6.321.2 Constructor & Destructor Documentation

6.321.2.1 Arc::XMLSecNode::XMLSecNode (XMLNode & *node*)

Create a object based on an **XMLNode** (p. 465) instance.

6.321.3 Member Function Documentation

6.321.3.1 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

6.321.3.2 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

6.321.3.3 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).

6.321.3.4 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

6.321.3.5 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

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. *387) 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@][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:` `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