Hosting Environment (Daemon) Reference Manual

Generated by Doxygen 1.4.7

Tue Jul 12 13:57:31 2011

Contents

Chapter 1

Hosting Environment (Daemon) Namespace Index

1.1 Hosting Environment (Daemon) Namespace List

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

Chapter 2

Hosting Environment (Daemon) Hierarchical Index

2.1 Hosting Environment (Daemon) Class Hierarchy

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

Chapter 3

Hosting Environment (Daemon) Data Structure Index

3.1 Hosting Environment (Daemon) Data Structures

Here are the data structures with brief descriptions:

Chapter 4

Hosting Environment (Daemon) Namespace Documentation

4.1 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.

- class ClientTCP

  Class for setting up a MCC chain for TCP communication.

- struct **HTTPClientInfo**
- class ClientHTTP

  Class for setting up a MCC chain for HTTP communication.

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

- class **SAML2SSOHTTPClient**
- class ApplicationEnvironment

  ApplicationEnvironment.

- class ExecutionTarget

  ExecutionTarget.

- class GLUE2

  GLUE2 parser.

- class Job

  Job.

- class JobController

  Must be specialiced for each supported middleware flavour.

- 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 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 ArcVersion

  Determines ARC HED libraries version.

- 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 SimpleFIFO

    Class representing a named pipe.

- class **SQLiteDatabase**
- class **SQLiteQuery**
- 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 to hold general URLs.

- class URLLocation

    Class to hold a resolved URL location.

- class PathIterator

  Class to iterate through elements of path.

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

  User configuration class

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

  Wrapper for pointer with automatic destruction.

- class CountedPointer

  Wrapper for pointer with automatic destruction and mutiple references.

- class **NS**
- class XMLNode

  Wrapper for LibXML library Tree interface.

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

- class DataBuffer

  Represents set of buffers.

- class DataCallback
- class DataHandle

  This class is a wrapper around the DataPoint class.

- class DataMover

- class DataPoint

  A DataPoint represents a data resource and is an abstraction of a URL.

- class DataPointLoader

  Class used by DataHandle to load the required DMC.

- class DataPointPluginArgument

  Class representing the arguments passed to DMC plugins.

- class DataPointDirect

  This is a kind of generalized file handle.

- class DataPointIndex

  Complements DataPoint with attributes common for Indexing Service URLs.

- class DataSpeed

  Keeps track of average and instantaneous transfer speed.

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

  FileInfo 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 - extension of Service 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, Service and Plexer objects.

- class MCC

  Message Chain Component - base class for every MCC plugin.

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

  A class for communication of MCC processing results.

- class MCCLoader

  Creator of Message Component Chains (MCC).

- class ChainContext

  Interface to chain specific functionality.

- class MessagePayload

  Base class for content of message passed through chain.

- class MessageContextElement

  Top class for elements contained in message context.

- class MessageContext

  Handler for content of message context.

- class MessageAuthContext

  Handler for content of message auth∗ context.

- class Message

  Object being passed through chain of MCCs.

- class AttributeIterator

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

- class MessageAttributes

  A class for storage of attribute values.

- class MessageAuth

  Contains authencity information, authorization tokens and decisions.

- class PayloadRawInterface

  Random Access Payload for Message objects.

- struct **PayloadRawBuf**
- class PayloadRaw

  Raw byte multi-buffer.

- class PayloadSOAP

  Payload of Message with SOAP content.

- class PayloadStreamInterface

  Stream-like Payload for Message object.

- class PayloadStream

  POSIX handle as Payload.

- class PlexerEntry

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

- class Plexer

  The Plexer class, used for routing messages to services.

- class CIStringValue

  This class implements case insensitive strings as security attributes.

- class SecAttrValue

  This is an abstract interface to a security attribute.

- class SecAttrFormat

  Export/import format.

- class SecAttr

  This is an abstract interface to a security attribute.

- class MultiSecAttr

  Container of multiple SecAttr attributes.

- class Service

  Service - last component in a Message Chain.

- class **ServicePluginArgument**
- class SOAPMessage

  Message restricted to SOAP payload.

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

  Interface for manipulation of WS-Adressing Endpoint Reference.

- class WSAHeader

  Interface for manipulation WS-Addressing information in SOAP header.

- class SAMLToken

  Class for manipulating SAML Token Profile.

- class UsernameToken

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

- class X509Token

  Class for manipulating X.509 Token Profile.

- class PayloadWSRF

  This class combines MessagePayload with WSRF.

- class WSRP

Base class for WS-ResourceProperties structures.

- class WSRPFault

  Base class for WS-ResourceProperties faults.

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

  Base class for every WSRF message.

- class WSRFBaseFault

  Base class for WSRF fault messages.

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

  Extends XMLNode class to support XML security operation.

Typedefs

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

Enumerations

- enum TimeFormat
- enum LogLevel
- enum LogFormat
- enum escape_type { , escape_octal, escape_hex }
- enum StatusKind { ,

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

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

Functions

- std::ostream & operator<< (std::ostream &, const Period &)
- std::ostream & operator<< (std::ostream &, const Time &)
- std::string TimeStamp (const TimeFormat &=Time::GetFormat())
- std::string TimeStamp (Time, const TimeFormat &=Time::GetFormat())
- bool FileCopy (const std::string &source_path, const std::string &destination_path, uid_t uid, gid_t gid)
- bool FileCopy (const std::string &source_path, const std::string &destination_path)
- bool FileCopy (const std::string &source_path, int destination_handle)
- bool FileCopy (int source_handle, const std::string &destination_path)
- bool FileCopy (int source_handle, int destination_handle)
- bool FileRead (const std::string &filename, std::list< std::string > &data, uid_t uid=0, gid_t gid=0)
- bool 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)
- bool TmpFileCreate (std::string &filename, const std::string &data, uid_t uid=0, gid_t gid=0)
- 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="")
- void tokenize (const std::string &str, std::list< std::string > &tokens, const std::string &delimiters=" ", const std::string &start_quotes="", const std::string &end_quotes="")
- std::string::size_type get_token (std::string &token, const std::string &str, std::string::size_type pos, const std::string &delimiters=" ", const std::string &start_quotes="", const std::string &end_quotes="")
- std::string trim (const std::string &str, const char *sep=NULL)
- std::string strip (const std::string &str)
- std::string uri_encode (const std::string &str, bool encode_slash)
- std::string uri_unencode (const std::string &str)
- std::string convert_to_rdn (const std::string &dn)
- std::string escape_chars (const std::string &str, const std::string &chars, char esc, bool excl, escape_type type=escape_char)
- std::string unescape_chars (const std::string &str, char esc, escape_type type=escape_char)
- bool CreateThreadFunction (void(*func)(void *), void *arg, SimpleCounter *count=NULL)
- std::list< URL > ReadURLList (const URL &urllist)
- std::string GetEnv (const std::string &var)
- std::string GetEnv (const std::string &var, bool &found)
- bool SetEnv (const std::string &var, const std::string &value, bool overwrite=true)
- void UnsetEnv (const std::string &var)
- void EnvLockWrap (bool all=false)
- void EnvLockUnwrap (bool all=false)
- void EnvLockUnwrapComplete (void)
- std::string StrError (int errnum=errno)
- bool MatchXMLName (const XMLNode &node1, const XMLNode &node2)
- bool MatchXMLName (const XMLNode &node, const char *name)
- bool MatchXMLName (const XMLNode &node, const std::string &name)
- bool MatchXMLNamespace (const XMLNode &node1, const XMLNode &node2)
- bool MatchXMLNamespace (const XMLNode &node, const char *uri)
- bool MatchXMLNamespace (const XMLNode &node, const std::string &uri)
- bool createVOMSAC (std::string &codedac, Credential &issuer_cred, Credential &holder_cred, std::vector< std::string > &fqan, std::vector< std::string > &targets, std::vector< std::string > &attributes, std::string &voname, std::string &uri, int lifetime)
- bool addVOMSAC (ArcCredential::AC **&aclist, std::string &acorder, std::string &decodedac)
- bool parseVOMSAC (X509 *holder, const std::string &ca_cert_dir, const std::string &ca_cert_file, const 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)
- std::string getCredentialProperty (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

4.1.1 Detailed Description

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

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

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

4.1.2 Typedef Documentation

4.1.2.1 typedef Plugin∗(∗) Arc::get_plugin_instance(PluginArgument ∗arg)

Constructor function of ARC lodable component.

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

4.1.2.2 typedef std::multimap<std::string,std::string> Arc::AttrMap

A typefed of a multimap for storage of message attributes.

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

4.1.2.3 typedef AttrMap::const_iterator Arc::AttrConstIter

A typedef of a const_iterator for AttrMap.

This typedef is used as a shorthand for a const_iterator for AttrMap. It is used extensively within the MessageAttributes class as well as the AttributesIterator class, but is not visible externally.

4.1.2.4 typedef AttrMap::iterator Arc::AttrIter

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

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

4.1.3 Enumeration Type Documentation

4.1.3.1 enum Arc::TimeFormat

An enumeration that contains the possible textual timeformats.

4.1.3.2 enum Arc::LogLevel

Logging levels.

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

4.1.3.3 enum Arc::LogFormat

Output formats.

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

4.1.3.4 enum Arc::escape_type

Type of escaping or encoding to use.

Enumerator:

escape_octal place the escape character before the character being escaped

escape_hex hex encoding of the character

4.1.3.5 enum Arc::StatusKind

Status kinds (types).

This enum defines a set of possible status kinds.

Enumerator:

STATUS_OK Default status - undefined error.

GENERIC_ERROR No error.

PARSING_ERROR Error does not fit any class.

PROTOCOL_RECOGNIZED_ERROR Error detected while parsing request/response.

UNKNOWN_SERVICE_ERROR Message does not fit into expected protocol.

BUSY_ERROR There is no destination configured for this message.

SESSION_CLOSE Message can't be processed now.

4.1.3.6 enum Arc::WSAFault

WS-Addressing possible faults.

Enumerator:

WSAFaultUnknown This is not a fault

WSAFaultInvalidAddressingHeader This is not a WS-Addressing fault

4.1.4 Function Documentation

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

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

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

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

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

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

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

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

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

Copy file source_path to file destination_path. Specified uid and gid are used for accessing filesystem.

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

Copy file source_path to file destination_path.

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

Copy file source_path to file handle destination_handle.

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

Copy from file handle source_handle to file destination_path.

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

Copy from file handle source_handle to file handle destination_handle.

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

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

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

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 should be used in addition to FileRead. If uid/gid are zero then no real switch of uid/gid is done.

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

Stat a file and put info into the st struct.

4.1.4.13 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.

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

Make symbolic or hard link of file.

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

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

4.1.4.16 std::string Arc::FileReadLink (const std::string & *path*)

Returns path at which symbolic link is pointing.

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

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

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

Deletes file at path.

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

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

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

Create a new directory.

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

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

4.1.4.22 bool Arc::DirDelete (const std::string & *path*)

Delete a directory and its content.

4.1.4.23 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.

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

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

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

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

Permissions of the created file are determined using the current umask. If uid/gid are zero then no real switch of uid/gid is done.

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

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

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

Generates a unique identifier using the system uuid libraries.

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

Printing of LogLevel values to ostreams.

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

4.1.4.29 LogLevel Arc::string_to_level (const std::string & *str*)

Convert string to a LogLevel.

4.1.4.30 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.

ll a Arc::LogLevel reference which will be set to the matching Arc::LogLevel upon successful parsing.

Returns:

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

See also:

LogLevel

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

Same as istring_to_level except it is case-sensitive.

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

Convert LogLevel to a string.

4.1.4.33 LogLevel Arc::old_level_to_level (unsigned int *old_level*)

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

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

This method converts a string to any type.

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

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

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

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

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

This method converts to lower case of the string.

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

This method converts to upper case of the string.

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

This method tokenizes string.

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

This method tokenizes string.

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

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

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

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

4.1.4.43 std::string Arc::strip (const std::string & *str*)

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

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

be encoded

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

4.1.4.45 std::string Arc::uri_unencode (const std::string & *str*)

This method unencodes the -encoded URI str.

4.1.4.46 std::string Arc::convert_to_rdn (const std::string & *dn*)

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

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

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

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

Unescape or encode characters in str escaped with esc.

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

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.

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

Reads a list of URLs from a file.

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

Portable function for getting environment variables.

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

Portable function for getting environment variables.

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

Portable function for setting environment variables.

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

Portable function for unsetting environment variables.

4.1.4.55 void Arc::EnvLockWrap (bool *all* = `false`)

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

4.1.4.56 void Arc::EnvLockUnwrap (bool *all* = `false`)

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

4.1.4.57 void Arc::EnvLockUnwrapComplete (void)

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

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

Portable function for obtaining description of last system error.

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

Returns true if underlying XML elements have same names

4.1.4.60 bool Arc::MatchXMLName (const XMLNode & *node*, const char ∗ *name*)

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

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

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

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

Returns true if underlying XML elements belong to same namespaces

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

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

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

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

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

Create AC(Attribute Certificate) with voms specific format.

Parameters:

codedac The coded AC as output of this method

issuer_cred The issuer credential which is used to sign the AC

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

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

Add decoded AC string into a list of AC objects

Parameters:

aclist The list of AC objects (output)

acorder The order of AC objects (output)

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

4.1.4.67 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.

4.1.4.68 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.

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

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

4.1.4.70 std::string Arc::getCredentialProperty (const Arc::Credential & *u*, const std::string & *property*)

Extract the needed field from the certificate

4.1.4.71 bool Arc::OpenSSLInit (void)

This function initializes OpenSSL library.

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

4.1.4.72 void Arc::HandleOpenSSLError (void)

Prints chain of accumulaed OpenSSL errors if any available.

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

Prints chain of accumulaed OpenSSL errors if any available.

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

Conversion to string.

Conversion from StatusKind to string.

Parameters:

kind The StatusKind to convert.

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

Returns pointer to main memory chunk of Message payload.

If no buffer is present or if payload is not of PayloadRawInterface type NULL is returned.

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

Makes WS-Addressing fault.

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

4.1.4.77 WSAFault Arc::WSAFaultExtract (SOAPEnvelope & *message*)

Gets WS-addressing fault.

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

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

callback method for inputing passphrase of key file

4.1.4.79 bool Arc::init_xmlsec (void)

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

4.1.4.80 bool Arc::final_xmlsec (void)

Finalize the xml security library

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

Get certificate in string format from certificate file

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

Get key in xmlSecKey structure from key in string format

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

Get key in xmlSecKey structure from key file

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

Get public key in string format from certificate file

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

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

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

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

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

Load public key from a certificate file into key manager

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

Load public key from a certificate string into key manager

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

Load trusted certificate from certificate file into key manager

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

Load trusted certificate from cetrtificate string into key manager

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

Load trusted cetificates from a file or directory into key manager

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

Generate a new child XMLNode with specified name

4.1.5 Variable Documentation

4.1.5.1 const Glib::TimeVal Arc::ETERNAL

A time very far in the future.

4.1.5.2 const Glib::TimeVal Arc::HISTORIC

A time very far in the past.

4.1.5.3 const size_t Arc::thread_stacksize = (16 ∗ 1024 ∗ 1024)

Defines size of stack assigned to every new thread.

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

4.1.5.4 Logger Arc::CredentialLogger

Logger to be used by all modules of credentials library

4.1.5.5 const char∗ Arc::plugins_table_name

Name of symbol 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 elements. The array is terminated by element with all components set to NULL.

4.2 ArcCredential Namespace Reference

Data Structures

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

Enumerations

- enum certType {
  CERT_TYPE_EEC, CERT_TYPE_CA, CERT_TYPE_GSI_3_IMPERSONATION_PROXY, CERT_TYPE_GSI_3_INDEPENDENT_PROXY,
  CERT_TYPE_GSI_3_LIMITED_PROXY, CERT_TYPE_GSI_3_RESTRICTED_PROXY, CERT_TYPE_GSI_2_PROXY, CERT_TYPE_GSI_2_LIMITED_PROXY,
  CERT_TYPE_RFC_IMPERSONATION_PROXY, CERT_TYPE_RFC_INDEPENDENT_-PROXY, CERT_TYPE_RFC_LIMITED_PROXY, CERT_TYPE_RFC_RESTRICTED_PROXY,
  CERT_TYPE_RFC_ANYLANGUAGE_PROXY }

4.2.1 Detailed Description

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

4.2.2 Enumeration Type Documentation

4.2.2.1 enum ArcCredential::certType

Enumerator:

CERT_TYPE_EEC A end entity certificate

CERT_TYPE_CA A CA certificate

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

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

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

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

CERT_TYPE_GSI_2_PROXY A legacy Globus impersonation proxy

CERT_TYPE_GSI_2_LIMITED_PROXY A legacy Globus limited impersonation proxy

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

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

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

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

CERT_TYPE_RFC_ANYLANGUAGE_PROXY RFC anyLanguage proxy

4.3 DataStaging Namespace Reference

DataStaging contains all components for data transfer scheduling and execution.

Data Structures

- class DataDelivery

  DataDelivery transfers data between specified physical locations.

- class DataDeliveryComm

  This class starts, monitors and controls a Delivery process.

- class TransferParameters

  Represents limits and properties of a DTR transfer.

- class CacheParameters

  The configured cache directories.

- class DTRCallback

  The base class from which all callback-enabled classes should be derived.

- class DTR

  Data Transfer Request.

- class DTRList

  Global list of all active DTRs in the system.

- class DTRStatus

  Class representing the status of a DTR.

- class DTRErrorStatus

  A class to represent error states reported by various components.

- class Generator

  Simple Generator implementation.

- class Processor

  The Processor performs pre- and post-transfer operations.

- class Scheduler

  The Scheduler is the control centre of the data staging framework.

- class TransferShares

  TransferShares is used to implement fair-sharing and priorities.

Enumerations

- enum StagingProcesses
- enum ProcessState
- enum CacheState {
  CACHEABLE, NON_CACHEABLE, CACHE_RENEW, CACHE_ALREADY_PRESENT,
  CACHE_DOWNLOADED, CACHE_LOCKED, CACHE_SKIP, CACHE_NOT_USED }

4.3.1 Detailed Description

DataStaging contains all components for data transfer scheduling and execution.

4.3.2 Enumeration Type Documentation

4.3.2.1 enum DataStaging::StagingProcesses

Components of the data staging framework.

4.3.2.2 enum DataStaging::ProcessState

Internal state of staging processes.

4.3.2.3 enum DataStaging::CacheState

Represents possible cache states of this DTR.

Enumerator:

CACHEABLE Source should be cached.

NON_CACHEABLE Source should not be cached.

CACHE_RENEW Cache file should be deleted then re-downloaded.

CACHE_ALREADY_PRESENT Source is available in cache from before.

CACHE_DOWNLOADED Source has just been downloaded and put in cache.

CACHE_LOCKED Cache file is locked.

CACHE_SKIP Source is cacheable but due to some problem should not be cached.

CACHE_NOT_USED Cache was started but was not used.

Chapter 5

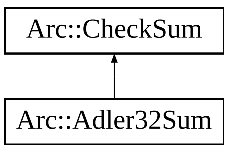
Hosting Environment (Daemon) Data Structure Documentation

5.1 Arc::Adler32Sum Class Reference

Implementation of Adler32 checksum.

`#include <CheckSum.h>`

Inheritance diagram for Arc::Adler32Sum::



5.1.1 Detailed Description

Implementation of Adler32 checksum.

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

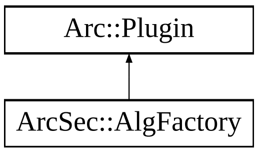
- CheckSum.h

5.2 ArcSec::AlgFactory Class Reference

Interface for algorithm factory class.

`#include <AlgFactory.h>`

Inheritance diagram for ArcSec::AlgFactory::



Public Member Functions

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

5.2.1 Detailed Description

Interface for algorithm factory class.

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

5.2.2 Member Function Documentation

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

creat algorithm object based on the type algorithm type

Parameters:

type The type of combining algorithm

Returns:

The object of CombiningAlg

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

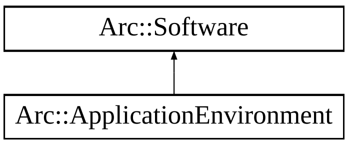
- AlgFactory.h

5.3 Arc::ApplicationEnvironment Class Reference

ApplicationEnvironment.

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

Inheritance diagram for Arc::ApplicationEnvironment::



5.3.1 Detailed Description

ApplicationEnvironment.

The ApplicationEnviroment is closely related to the definition given in GLUE2. By extending the Software class the two GLUE2 attributes AppName and AppVersion are mapped to two private members. However these can be obtained through the 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

5.4 Arc::ArcLocation Class Reference

Determines ARC installation location.

`#include <ArcLocation.h>`

Static Public Member Functions

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

5.4.1 Detailed Description

Determines ARC installation location.

5.4.2 Member Function Documentation

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

Returns ARC installation location.

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

Returns ARC plugins directory location.

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

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

Initializes location information.

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

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

- ArcLocation.h

5.5 Arc::ArcVersion Class Reference

Determines ARC HED libraries version.

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

5.5.1 Detailed Description

Determines ARC HED libraries version.

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

- ArcVersion.h

5.6 ArcSec::Attr Struct Reference

Attr contains a tuple of attribute type and value.

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

5.6.1 Detailed Description

Attr contains a tuple of attribute type and value.

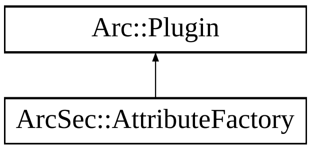
The documentation for this struct was generated from the following file:

- Request.h

5.7 ArcSec::AttributeFactory Class Reference

`#include <AttributeFactory.h>`

Inheritance diagram for ArcSec::AttributeFactory::



5.7.1 Detailed Description

Base attribute factory class

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

- AttributeFactory.h

5.8 Arc::AttributeIterator Class Reference

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

`#include <MessageAttributes.h>`

Public Member Functions

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

Protected Member Functions

- AttributeIterator (AttrConstIter begin, AttrConstIter end)

Protected Attributes

- AttrConstIter current_
- AttrConstIter end_

Friends

- class MessageAttributes

5.8.1 Detailed Description

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

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

Typical usage is:

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

5.8.2 Constructor & Destructor Documentation

5.8.2.1 Arc::AttributeIterator::AttributeIterator ()

Default constructor.

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

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

Protected constructor used by the MessageAttributes class.

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

Parameters:

begin A const_iterator pointing to the first matching key-value pair in the internal multimap of the MessageAttributes class.

end A const_iterator pointing to the first key-value pair in the internal multimap of the Message-Attributes class where the key is larger than the key searched for.

5.8.3 Member Function Documentation

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

Predicate method for iteration termination.

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

Returns:

Returns true if there are more values, otherwise false.

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

The key of attribute.

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

5.8.3.3 const std::string& Arc::AttributeIterator::operator ∗ () const

The dereference operator.

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

Returns:

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

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

The postfix advance operator.

Advances the iterator to the next value. Works intuitively.

Returns:

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

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

The prefix advance operator.

Advances the iterator to the next value. Works intuitively.

Returns:

A const reference to this iterator.

5.8.3.6 const std::string$*$ Arc::AttributeIterator::operator $\rightarrow$ () const

The arrow operator.

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

5.8.4 Friends And Related Function Documentation

5.8.4.1 friend class MessageAttributes `[friend]`

The MessageAttributes class is a friend.

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

5.8.5 Field Documentation

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

A const_iterator pointing to the current key-value pair.

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

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

A const_iterator pointing beyond the last key-value pair.

A const_iterator pointing to the first key-value pair in the internal multimap of the MessageAttributes class where the key is larger than the key searched for.

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

- MessageAttributes.h

5.9 ArcSec::AttributeProxy Class Reference

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

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

Public Member Functions

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

5.9.1 Detailed Description

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

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

5.9.2 Member Function Documentation

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

Create a AttributeValue object according to the information inside the XMLNode as parameter.

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

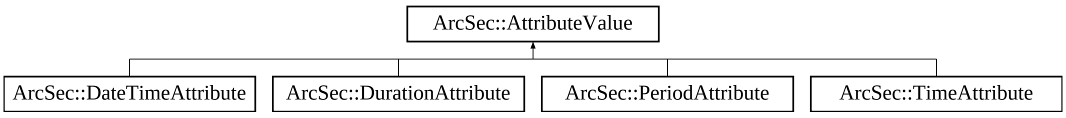
- AttributeProxy.h

5.10 ArcSec::AttributeValue Class Reference

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

`#include <AttributeValue.h>`

Inheritance diagram for ArcSec::AttributeValue::



Public Member Functions

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

5.10.1 Detailed Description

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

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

5.10.2 Member Function Documentation

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

encode the value in a string format

Implemented in ArcSec::DateTimeAttribute, ArcSec::TimeAttribute, ArcSec::DurationAttribute, and ArcSec::PeriodAttribute.

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

Evluate whether "this" equale to the parameter value

Implemented in ArcSec::DateTimeAttribute, ArcSec::TimeAttribute, ArcSec::DurationAttribute, and ArcSec::PeriodAttribute.

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

Get the AttributeId of the <Attribute>

Implemented in ArcSec::DateTimeAttribute, ArcSec::TimeAttribute, ArcSec::DurationAttribute, and ArcSec::PeriodAttribute.

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

Get the DataType of the <Attribute>

Implemented in ArcSec::DateTimeAttribute, ArcSec::TimeAttribute, ArcSec::DurationAttribute, and ArcSec::PeriodAttribute.

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

- AttributeValue.h

5.11 ArcSec::Attrs Class Reference

Attrs is a container for one or more Attr.

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

5.11.1 Detailed Description

Attrs is a container for one or more Attr.

Attrs includes includes methonds for inserting, getting items, and counting size as well

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

- Request.h

5.12 ArcSec::AuthzRequestSection Struct Reference

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

5.12.1 Detailed Description

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

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

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

- PDP.h

5.13 Arc::AutoPointer< T > Class Template Reference

Wrapper for pointer with automatic destruction.

`#include <Utils.h>`

Public Member Functions

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

5.13.1 Detailed Description

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

Wrapper for pointer with automatic destruction.

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

5.13.2 Constructor & Destructor Documentation

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

NULL pointer constructor.

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

Constructor which wraps pointer.

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

Destructor destroys wrapped object using delete().

5.13.3 Member Function Documentation

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

For refering wrapped object.

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

Returns false if pointer is NULL and true otherwise.

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

Cast to original pointer.

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

Returns true if pointer is NULL and false otherwise.

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

For refering wrapped object.

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

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

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

- Utils.h

5.14 Arc::BaseConfig Class Reference

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

Public Member Functions

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

5.14.1 Detailed Description

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

5.14.2 Member Function Documentation

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

Add CA directory

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

Add CA file

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

Add certificate

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

Add configuration overlay

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

Adds non-standard location of plugins

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

Add private key

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

Add credentials proxy

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

Read overlay from file

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

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

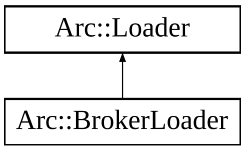
The documentation for this class was generated from the following file:

- ArcConfig.h

5.15 Arc::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

5.15.1 Detailed Description

Class responsible for loading Broker plugins The Broker objects returned by a BrokerLoader must not be used after the BrokerLoader goes out of scope.

5.15.2 Constructor & Destructor Documentation

5.15.2.1 Arc::BrokerLoader::BrokerLoader ()

Constructor Creates a new BrokerLoader.

5.15.2.2 Arc::BrokerLoader::∼BrokerLoader ()

Destructor Calling the destructor destroys all Brokers loaded by the BrokerLoader instance.

5.15.3 Member Function Documentation

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

Retrieve the list of loaded Brokers.

Returns:

A reference to the list of Brokers.

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

Load a new Broker

Parameters:

name The name of the Broker to load.

usercfg The UserConfig object for the new Broker.

Returns:

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

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

- Broker.h

5.16 Arc::CacheParameters Struct Reference

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

5.16.1 Detailed Description

Contains data on the parameters of a cache.

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

- FileCache.h

5.17 DataStaging::CacheParameters Class Reference

The configured cache directories.

```
#include <DTR.h>
```

Public Member Functions

- CacheParameters (void)
- CacheParameters (std::vector< std::string > caches, std::vector< std::string > remote_caches, std::vector< std::string > drain_caches)

Data Fields

- std::vector< std::string > cache_dirs
- std::vector< std::string > remote_cache_dirs
- std::vector< std::string > drain_cache_dirs

5.17.1 Detailed Description

The configured cache directories.

5.17.2 Constructor & Destructor Documentation

5.17.2.1 DataStaging::CacheParameters::CacheParameters (void) `[inline]`

Constructor with empty lists initialised.

5.17.2.2 DataStaging::CacheParameters::CacheParameters (std::vector< std::string > *caches*, std::vector< std::string > *remote_caches*, std::vector< std::string > *drain_caches*)

Constructor with supplied cache lists.

5.17.3 Field Documentation

5.17.3.1 std::vector<std::string> DataStaging::CacheParameters::cache_dirs

List of (cache dir [link dir]).

5.17.3.2 std::vector<std::string> DataStaging::CacheParameters::drain_cache_dirs

List of draining caches. Not necessary for data staging but here for completeness.

5.17.3.3 std::vector<std::string> DataStaging::CacheParameters::remote_cache_dirs

List of (cache dir [link dir]) for remote caches.

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

- DTR.h

5.18 Arc::ChainContext Class Reference

Interface to chain specific functionality.

`#include <MCCLoader.h>`

Public Member Functions

- operator PluginsFactory ∗ ()

5.18.1 Detailed Description

Interface to chain specific functionality.

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

5.18.2 Member Function Documentation

5.18.2.1 Arc::ChainContext::operator PluginsFactory ∗ () `[inline]`

Returns associated PluginsFactory object

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

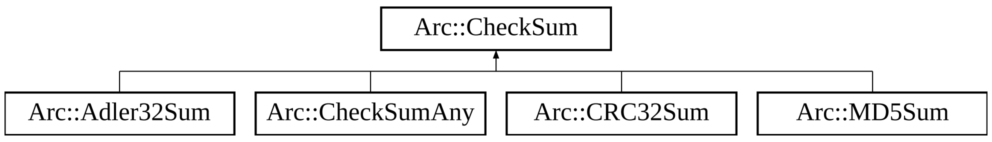
- MCCLoader.h

5.19 Arc::CheckSum Class Reference

Defines interface for variuos checksum manipulations.

`#include <CheckSum.h>`

Inheritance diagram for Arc::CheckSum::



5.19.1 Detailed Description

Defines interface for variuos checksum manipulations.

This class is used during data transfers through DataBuffer class

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

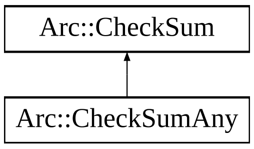
- CheckSum.h

5.20 Arc::CheckSumAny Class Reference

Wraper for CheckSum class.

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

Inheritance diagram for Arc::CheckSumAny::



5.20.1 Detailed Description

Wraper for CheckSum class.

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

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

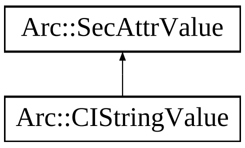
- CheckSum.h

5.21 Arc::CIStringValue Class Reference

This class implements case insensitive strings as security attributes.

```
#include <CIStringValue.h>
```

Inheritance diagram for Arc::CIStringValue::



Public Member Functions

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

Protected Member Functions

- virtual bool equal (SecAttrValue &b)

5.21.1 Detailed Description

This class implements case insensitive strings as security attributes.

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

5.21.2 Constructor & Destructor Documentation

5.21.2.1 Arc::CIStringValue::CIStringValue ()

Default constructor

5.21.2.2 Arc::CIStringValue::CIStringValue (const char ∗ *ss*)

This is a constructor that takes a string litteral.

5.21.2.3 Arc::CIStringValue::CIStringValue (const std::string & *ss*)

This is a constructor that takes a string object.

5.21.3 Member Function Documentation

5.21.3.1 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.

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

This function returns false if the string is empty or uninitialized

Reimplemented from Arc::SecAttrValue.

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

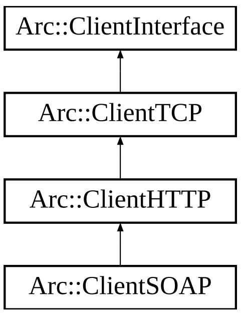
- CIStringValue.h

5.22 Arc::ClientHTTP Class Reference

Class for setting up a MCC chain for HTTP communication.

`#include <ClientInterface.h>`

Inheritance diagram for Arc::ClientHTTP::



5.22.1 Detailed Description

Class for setting up a MCC chain for HTTP communication.

The ClientHTTP class inherits from the ClientTCP class and adds an HTTP MCC to the chain.

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

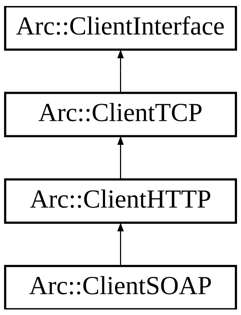
- ClientInterface.h

5.23 Arc::ClientInterface Class Reference

Utility base class for MCC.

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

Inheritance diagram for Arc::ClientInterface::



5.23.1 Detailed Description

Utility base class for MCC.

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

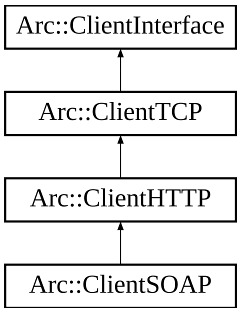
The documentation for this class was generated from the following file:

- ClientInterface.h

5.24 Arc::ClientSOAP Class Reference

`#include <ClientInterface.h>`

Inheritance diagram for Arc::ClientSOAP::



Public Member Functions

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

5.24.1 Detailed Description

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

5.24.2 Constructor & Destructor Documentation

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

Constructor creates MCC chain and connects to server.

5.24.3 Member Function Documentation

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

Adds security handler to configuration of SOAP MCC

Reimplemented from Arc::ClientHTTP.

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

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

Reimplemented from Arc::ClientHTTP.

5.24.3.3 virtual bool Arc::ClientSOAP::Load () `[virtual]`

Instantiates pluggable elements according to generated configuration

Reimplemented from Arc::ClientHTTP.

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

Send SOAP request with specified SOAP action and receive response.

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

Send SOAP request and receive response.

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

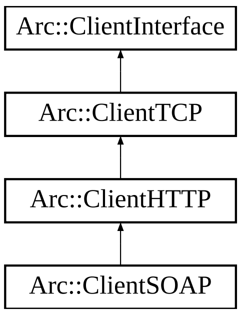
- ClientInterface.h

5.25 Arc::ClientTCP Class Reference

Class for setting up a MCC chain for TCP communication.

`#include <ClientInterface.h>`

Inheritance diagram for Arc::ClientTCP::



5.25.1 Detailed Description

Class for setting up a MCC chain for TCP communication.

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

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

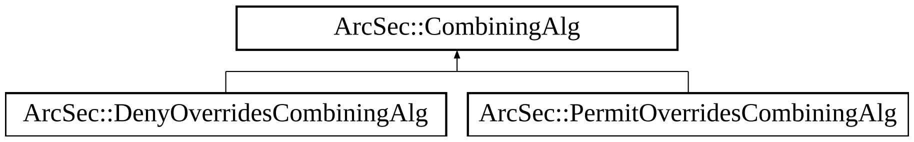
- ClientInterface.h

5.26 ArcSec::CombiningAlg Class Reference

Interface for combining algrithm.

`#include <CombiningAlg.h>`

Inheritance diagram for ArcSec::CombiningAlg::


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

Public Member Functions

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

5.26.1 Detailed Description

Interface for combining algrithm.

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

5.26.2 Member Function Documentation

5.26.2.1 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, and ArcSec::PermitOverridesCombiningAlg.

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

Get the identifier of the combining algorithm class

Returns:

The identity of the algorithm

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

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

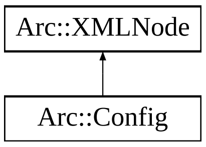
- CombiningAlg.h

5.27 Arc::Config Class Reference

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

`#include <ArcConfig.h>`

Inheritance diagram for Arc::Config::



Public Member Functions

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

5.27.1 Detailed Description

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

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

5.27.2 Constructor & Destructor Documentation

5.27.2.1 Arc::Config::Config () `[inline]`

Creates empty XML tree

5.27.2.2 Arc::Config::Config (const char ∗ *filename*)

Loads configuration document from file 'filename'

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

Parse configuration document from memory

5.27.2.4 Arc::Config::Config (XMLNode *xml*) `[inline]`

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

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

Copy constructor used by language bindings

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

Copy constructor used by language bindings

5.27.3 Member Function Documentation

5.27.3.1 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 subtree

5.27.3.2 bool Arc::Config::parse (const char ∗ *filename*)

Parse configuration document from file 'filename'

5.27.3.3 void Arc::Config::print (void)

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

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

Save to file

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

Set the file name of config file

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

- ArcConfig.h

5.28 Arc::ConfusaCertHandler Class Reference

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

Public Member Functions

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

5.28.1 Detailed Description

Wrapper around Credential handling the Confusa specifics.

5.28.2 Constructor & Destructor Documentation

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

Create a new ConfusaCertHandler for DN dn and given keysize Basically Confusa cert handler wraps around Credential

5.28.3 Member Function Documentation

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

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

5.28.3.2 std::string Arc::ConfusaCertHandler::getCertRequestB64 ()

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

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

- ConfusaCertHandler.h

5.29 Arc::ConfusaParserUtils Class Reference

`#include <ConfusaParserUtils.h>`

Static Public Member Functions

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

5.29.1 Detailed Description

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

5.29.2 Member Function Documentation

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

Destroy a libxml2 doc representation

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

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

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

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

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

Construct a lixml2 doc representation from the xml file

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

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

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

urlencode the passed string

5.29.2.7 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

5.30 Arc::CountedPointer< T > Class Template Reference

Wrapper for pointer with automatic destruction and mutiple references.

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

Public Member Functions

- T & operator $*$ (void) const
- T $*$ operator $\rightarrow$ (void) const
- operator bool (void) const
- bool operator! (void) const
- operator T $*$ (void) const
- T $*$ Release (void)

Data Structures

- class **Base**

5.30.1 Detailed Description

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

Wrapper for pointer with automatic destruction and mutiple references.

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

5.30.2 Member Function Documentation

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

For refering wrapped object.

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

Returns false if pointer is NULL and true otherwise.

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

Cast to original pointer.

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

Returns true if pointer is NULL and false otherwise.

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

For refering wrapped object.

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

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

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

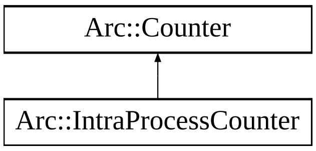
- Utils.h

5.31 Arc::Counter Class Reference

A class defining a common interface for counters.

`#include <Counter.h>`

Inheritance diagram for Arc::Counter::



Public Member Functions

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

Protected Types

- typedef unsigned long long int IDType

Protected Member Functions

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

Friends

- class CounterTicket
- class ExpirationReminder

5.31.1 Detailed Description

A class defining a common interface for counters.

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

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

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

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

Typical usage is:

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

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

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

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

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

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

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

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

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

5.31.2 Member Typedef Documentation

5.31.2.1 typedef unsigned long long int Arc::Counter::IDType `[protected]`

A typedef of identification numbers for reservation.

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

5.31.3 Constructor & Destructor Documentation

5.31.3.1 Arc::Counter::Counter () `[protected]`

Default constructor.

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

5.31.3.2 virtual Arc::Counter::∼Counter () `[virtual]`

The destructor.

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

5.31.4 Member Function Documentation

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

Cancellation of a reservation.

This method cancels a reservation. It is called by the CounterTicket that corresponds to the reservation.

Parameters:

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

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

Changes the excess limit of the counter.

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

Parameters:

amount The amount by which to change the excess limit.

Returns:

The new excess limit.

Implemented in Arc::IntraProcessCounter.

5.31.4.3 virtual int Arc::Counter::changeLimit (int *amount*) `[pure virtual]`

Changes the limit of the counter.

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

Parameters:

amount The amount by which to change the limit.

Returns:

The new limit.

Implemented in Arc::IntraProcessCounter.

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

Extension of a reservation.

This method extends a reservation. It is called by the CounterTicket that corresponds to the reservation.

Parameters:

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

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

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

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

A "relay method" for a constructor of the CounterTicket class.

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

Parameters:

reservationID The identity number of the reservation corresponding to the CounterTicket.

expiryTime the expiry time of the reservation corresponding to the CounterTicket.

counter The Counter from which the reservation has been made.

Returns:

The counter ticket that has been created.

5.31.4.6 Glib::TimeVal Arc::Counter::getCurrentTime () `[protected]`

Get the current time.

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

5.31.4.7 virtual int Arc::Counter::getExcess () `[pure virtual]`

Returns the excess limit of the counter.

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

Returns:

The excess limit.

Implemented in Arc::IntraProcessCounter.

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

A "relay method" for the constructor of ExpirationReminder.

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

Parameters:

expTime the expiry time of the reservation corresponding to the ExpirationReminder.

resID The identity number of the reservation corresponding to the ExpirationReminder.

Returns:

The ExpirationReminder that has been created.

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

Computes an expiry time.

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

Parameters:

duration The duration.

Returns:

The expiry time.

5.31.4.10 virtual int Arc::Counter::getLimit () `[pure virtual]`

Returns the current limit of the counter.

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

Returns:

The current limit of the counter.

Implemented in Arc::IntraProcessCounter.

5.31.4.11 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.

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

Makes a reservation from the counter.

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

Parameters:

amount The amount to reserve, default value is 1.

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

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

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

Returns:

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

Implemented in Arc::IntraProcessCounter.

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

Sets the excess limit of the counter.

This method sets a new excess limit for the counter.

Parameters:

newExcess The new excess limit, an absolute number.

Returns:

The new excess limit.

Implemented in Arc::IntraProcessCounter.

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

Sets the limit of the counter.

This method sets a new limit for the counter.

Parameters:

newLimit The new limit, an absolute number.

Returns:

The new limit.

Implemented in Arc::IntraProcessCounter.

5.31.5 Friends And Related Function Documentation

5.31.5.1 friend class CounterTicket `[friend]`

The CounterTicket class needs to be a friend.

5.31.5.2 friend class ExpirationReminder `[friend]`

The ExpirationReminder class needs to be a friend.

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

- Counter.h

5.32 Arc::CounterTicket Class Reference

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

`#include <Counter.h>`

Public Member Functions

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

Friends

- class Counter

5.32.1 Detailed Description

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

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

Typical usage is:

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

5.32.2 Constructor & Destructor Documentation

5.32.2.1 Arc::CounterTicket::CounterTicket ()

The default constructor.

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

5.32.3 Member Function Documentation

5.32.3.1 void Arc::CounterTicket::cancel ()

Cancels a resrvation.

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

5.32.3.2 void Arc::CounterTicket::extend (Glib::TimeVal *duration*)

Extends a reservation.

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

Parameters:

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

5.32.3.3 bool Arc::CounterTicket::isValid ()

Returns the validity of a CounterTicket.

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

Returns:

The validity of the ticket.

5.32.4 Friends And Related Function Documentation

5.32.4.1 friend class Counter `[friend]`

The Counter class needs to be a friend.

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

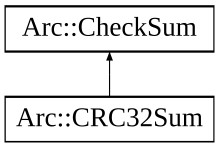
- Counter.h

5.33 Arc::CRC32Sum Class Reference

Implementation of CRC32 checksum.

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

Inheritance diagram for Arc::CRC32Sum::



5.33.1 Detailed Description

Implementation of CRC32 checksum.

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

- CheckSum.h

5.34 Arc::CredentialError Class Reference

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

Public Member Functions

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

5.34.1 Detailed Description

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

5.34.2 Constructor & Destructor Documentation

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

This is the constructor of the CredentialError class.

Parameters:

what An explanation of the error.

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

- Credential.h

5.35 Arc::CredentialStore Class Reference

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

5.35.1 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:

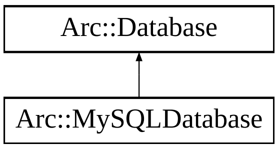
- CredentialStore.h

5.36 Arc::Database Class Reference

Interface for calling database client library.

```
#include <DBInterface.h>
```

Inheritance diagram for Arc::Database::



Public Member Functions

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

5.36.1 Detailed Description

Interface for calling database client library.

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

5.36.2 Constructor & Destructor Documentation

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

Default constructor

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

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

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

Copy constructor

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

Deconstructor

5.36.3 Member Function Documentation

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

Close the connection with database server

Implemented in Arc::MySQLDatabase.

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

Do connection with database server

Parameters:

dbname The database name which will be used.

user The username which will be used to access database.

password The password which will be used to access database.

Implemented in Arc::MySQLDatabase.

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

Enable ssl communication for the connection

Parameters:

keyfile The location of key file.

certfile The location of certificate file.

cafile The location of ca file.

capath The location of ca directory

Implemented in Arc::MySQLDatabase.

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

Get the connection status

Implemented in Arc::MySQLDatabase.

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

Ask database server to shutdown

Implemented in Arc::MySQLDatabase.

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

- DBInterface.h

5.37 Arc::DataBuffer Class Reference

Represents set of buffers.

`#include <DataBuffer.h>`

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

Data Structures

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

5.37.1 Detailed Description

Represents set of buffers.

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

5.37.2 Constructor & Destructor Documentation

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

Contructor

Parameters:

size size of every buffer in bytes.

blocks number of buffers.

5.37.2.2 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 itself.

5.37.2.3 Arc::DataBuffer::∼DataBuffer ()

Destructor.

5.37.3 Member Function Documentation

5.37.3.1 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 itself.

Returns:

integer position in the list of checksum objects.

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

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

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

Returns CheckSum object specified in constructor, returns NULL if index is not in list.

Parameters:

index of the checksum in question.

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

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

Parameters:

index of the checksum in question.

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

Returns offset following last piece of data transferred.

5.37.3.6 bool Arc::DataBuffer::eof_read ()

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

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

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

5.37.3.8 bool Arc::DataBuffer::eof_write ()

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

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

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

5.37.3.10 bool Arc::DataBuffer::error ()

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

5.37.3.11 bool Arc::DataBuffer::error_read ()

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

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

Informs object if error accured on 'read' side.

Parameters:

v true if error.

5.37.3.13 bool Arc::DataBuffer::error_transfer ()

Returns true if eror occured inside object.

5.37.3.14 bool Arc::DataBuffer::error_write ()

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

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

Informs object if error accured on 'write' side.

Parameters:

v true if error.

5.37.3.16 bool Arc::DataBuffer::for_read ()

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

5.37.3.17 bool Arc::DataBuffer::for_read (int & *handle*, unsigned int & *length*, bool *wait*)

Request buffer for READING INTO it.

Parameters:

handle returns buffer's number.

length returns size of buffer

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

Returns:

true on success

5.37.3.18 bool Arc::DataBuffer::for_write ()

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

5.37.3.19 bool Arc::DataBuffer::for_write (int & *handle*, unsigned int & *length*, unsigned long long int & *offset*, bool *wait*)

Request buffer for WRITING FROM it.

Parameters:

handle returns buffer's number.

length returns size of buffer

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

5.37.3.20 bool Arc::DataBuffer::is_notwritten (char ∗ *buf*)

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

Parameters:

buf - address of buffer

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

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

Parameters:

handle buffer's number.

5.37.3.22 bool Arc::DataBuffer::is_read (char ∗ *buf*, unsigned int *length*, unsigned long long int *offset*)

Informs object that data was read into buffer.

Parameters:

buf - address of buffer

length amount of data.

offset offset in stream, file, etc.

5.37.3.23 bool Arc::DataBuffer::is_read (int *handle*, unsigned int *length*, unsigned long long int *offset*)

Informs object that data was read into buffer.

Parameters:

handle buffer's number.

length amount of data.

offset offset in stream, file, etc.

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

Informs object that data was written from buffer.

Parameters:

buf - address of buffer

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

Informs object that data was written from buffer.

Parameters:

handle buffer's number.

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

Check if DataBuffer object is initialized.

5.37.3.27]

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

Direct access to buffer by number.

5.37.3.28 bool Arc::DataBuffer::set (CheckSum ∗ *cksum* = `NULL`, unsigned int *size* = `65536`, int *blocks* = `3`)

Reinitialize buffers with different parameters.

Parameters:

size size of every buffer in bytes.

blocks number of buffers.

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

5.37.3.29 bool Arc::DataBuffer::wait_any ()

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

5.37.3.30 bool Arc::DataBuffer::wait_eof ()

Wait till end of transfer happens on any side.

5.37.3.31 bool Arc::DataBuffer::wait_eof_read ()

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

5.37.3.32 bool Arc::DataBuffer::wait_eof_write ()

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

5.37.3.33 bool Arc::DataBuffer::wait_read ()

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

5.37.3.34 bool Arc::DataBuffer::wait_used ()

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

5.37.3.35 bool Arc::DataBuffer::wait_write ()

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

5.37.4 Field Documentation

5.37.4.1 DataSpeed Arc::DataBuffer::speed

This object controls transfer speed.

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

- DataBuffer.h

5.38 Arc::DataCallback Class Reference

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

5.38.1 Detailed Description

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

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

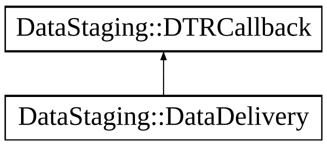
- DataCallback.h

5.39 DataStaging::DataDelivery Class Reference

DataDelivery transfers data between specified physical locations.

`#include <DataDelivery.h>`

Inheritance diagram for DataStaging::DataDelivery::



Public Member Functions

- DataDelivery ()
- ∼DataDelivery ()
- virtual void receiveDTR (DTR &)
- bool cancelDTR (DTR ∗)
- bool start ()
- bool stop ()
- void SetTransferParameters (const TransferParameters &params)

5.39.1 Detailed Description

DataDelivery transfers data between specified physical locations.

All meta-operations for a DTR such as resolving replicas must be done before sending to DataDelivery. Calling receiveDTR() starts a new process which performs data transfer as specified in DTR.

5.39.2 Constructor & Destructor Documentation

5.39.2.1 DataStaging::DataDelivery::DataDelivery ()

Constructor.

5.39.2.2 DataStaging::DataDelivery::∼DataDelivery () `[inline]`

Destructor calls stop() and waits for cancelled processes to exit.

5.39.3 Member Function Documentation

5.39.3.1 bool DataStaging::DataDelivery::cancelDTR (DTR ∗)

Kill the process corresponding to the given DTR.

5.39.3.2 virtual void DataStaging::DataDelivery::receiveDTR (DTR &) `[virtual]`

Pass a DTR to Delivery.

This method is called by the scheduler to pass a DTR to the delivery. The DataDelivery starts a process to do the processing, and then returns. DataDelivery's own thread then monitors the started process.

Implements DataStaging::DTRCallback.

5.39.3.3 void DataStaging::DataDelivery::SetTransferParameters (const TransferParameters & *params*)

Set transfer limits.

5.39.3.4 bool DataStaging::DataDelivery::start ()

Start the Delivery thread, which runs until stop() is called.

5.39.3.5 bool DataStaging::DataDelivery::stop ()

Tell the delivery to shut down all processes and threads and exit.

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

- DataDelivery.h

5.40 DataStaging::DataDeliveryComm Class Reference

This class starts, monitors and controls a Delivery process.

```
#include <DataDeliveryComm.h>
```

Public Types

- CommInit
- CommNoError
- CommTimeout
- CommClosed
- CommExited
- CommFailed
- enum CommStatusType {
  CommInit, CommNoError, CommTimeout, CommClosed,
  CommExited, CommFailed }

Public Member Functions

- DataDeliveryComm (const DTR &dtr, const TransferParameters &params)
- ∼DataDeliveryComm (void)
- Status GetStatus (void) const
- const std::string GetError (void) const
- operator bool (void)
- bool operator! (void)

Protected Member Functions

- void PullStatus (void)

Protected Attributes

- Glib::Mutex lock_
- Status status_
- Status status_buf_
- unsigned int status_pos_
- Arc::Run ∗ child_
- DataDeliveryCommHandler ∗ handler_
- std::string dtr_id
- TransferParameters transfer_params
- Arc::Time last_comm
- Arc::Logger ∗ logger_

Data Structures

- struct Status

  Plain C struct for passing information from child process back to main thread.

5.40.1 Detailed Description

This class starts, monitors and controls a Delivery process.

5.40.2 Member Enumeration Documentation

5.40.2.1 enum DataStaging::DataDeliveryComm::CommStatusType

Communication status with child process.

Enumerator:

CommInit Initializing/starting child, rest of information not valid.

CommNoError Communication going on smoothly.

CommTimeout Communication experienced timeout.

CommClosed Communication channel was closed.

CommExited Child exited. Mostly same as CommClosed but exit detected before pipe closed.

CommFailed Child exited with exit code != 0. Child reports error in such way. If we have CommFailed and no error code reported that normally means segfault or external kill.

5.40.3 Constructor & Destructor Documentation

5.40.3.1 DataStaging::DataDeliveryComm::DataDeliveryComm (const DTR & *dtr*, const TransferParameters & *params*)

Starts external executable with parameters taken from DTR and supplied transfer limits.

5.40.3.2 DataStaging::DataDeliveryComm::∼DataDeliveryComm (void)

Destroy object. This stops the child process.

5.40.4 Member Function Documentation

5.40.4.1 const std::string DataStaging::DataDeliveryComm::GetError (void) const `[inline]`

Get explanation of error.

5.40.4.2 Status DataStaging::DataDeliveryComm::GetStatus (void) const

Obtain status of transfer.

5.40.4.3 DataStaging::DataDeliveryComm::operator bool (void) `[inline]`

Returns true child process exists.

5.40.4.4 bool DataStaging::DataDeliveryComm::operator! (void) `[inline]`

Returns true if child process does not exist.

5.40.4.5 void DataStaging::DataDeliveryComm::PullStatus (void) `[protected]`

Check for new state from child and fill state accordingly.

Detects communication and delivery failures and delivery termination.

5.40.5 Field Documentation

5.40.5.1 Arc::Run∗ DataStaging::DataDeliveryComm::child_ `[protected]`

Child process.

5.40.5.2 std::string DataStaging::DataDeliveryComm::dtr_id `[protected]`

ID of the DTR this object is handling.

5.40.5.3 DataDeliveryCommHandler∗ DataStaging::DataDeliveryComm::handler_ `[protected]`

Pointer to singleton handler of all DataDeilveryComm objects.

5.40.5.4 Arc::Time DataStaging::DataDeliveryComm::last_comm `[protected]`

Time last communication was received from child.

5.40.5.5 Glib::Mutex DataStaging::DataDeliveryComm::lock_ `[protected]`

Lock to protect access to child process.

5.40.5.6 Arc::Logger∗ DataStaging::DataDeliveryComm::logger_ `[protected]`

Logger object.

5.40.5.7 Status DataStaging::DataDeliveryComm::status_ `[protected]`

Current status of transfer.

5.40.5.8 Status DataStaging::DataDeliveryComm::status_buf_ `[protected]`

Latest status from child is read into this buffer.

5.40.5.9 unsigned int DataStaging::DataDeliveryComm::status_pos_ `[protected]`

Reading position of Status buffer.

5.40.5.10 TransferParameters DataStaging::DataDeliveryComm::transfer_params [protected]

Transfer limits.

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

- DataDeliveryComm.h

5.41 DataStaging::DataDeliveryComm::Status Struct Reference

Plain C struct for passing information from child process back to main thread.

```
#include <DataDeliveryComm.h>
```

Data Fields

- CommStatusType commstatus
- time_t timestamp
- DTRStatus::DTRStatusType status
- DTRErrorStatus::DTRErrorStatusType error
- DTRErrorStatus::DTRErrorLocation error_location
- char error_desc [256]
- unsigned int streams
- unsigned long long int transfered
- unsigned long long int offset
- unsigned long long int size
- unsigned int speed
- char checksum [128]

5.41.1 Detailed Description

Plain C struct for passing information from child process back to main thread.

5.41.2 Field Documentation

5.41.2.1 char DataStaging::DataDeliveryComm::Status::checksum[128]

Calculated checksum.

5.41.2.2 CommStatusType DataStaging::DataDeliveryComm::Status::commstatus

Communication state (filled by parent).

5.41.2.3 DTRErrorStatus::DTRErrorStatusType DataStaging::DataDeliveryComm::Status::error

Error type.

5.41.2.4 char DataStaging::DataDeliveryComm::Status::error_desc[256]

Error description.

5.41.2.5 DTRErrorStatus::DTRErrorLocation DataStaging::DataDeliveryComm::Status::error_-location

Where error happened.

5.41.2.6 unsigned long long int DataStaging::DataDeliveryComm::Status::offset

Last position to which file has no missing pieces.

5.41.2.7 unsigned long long int DataStaging::DataDeliveryComm::Status::size

File size as obtained by protocol.

5.41.2.8 unsigned int DataStaging::DataDeliveryComm::Status::speed

Current transfer speed in bytes/sec duiring last ∼minute.

5.41.2.9 DTRStatus::DTRStatusType DataStaging::DataDeliveryComm::Status::status

Generic status.

5.41.2.10 unsigned int DataStaging::DataDeliveryComm::Status::streams

Number of transfer streams active.

5.41.2.11 time_t DataStaging::DataDeliveryComm::Status::timestamp

Time when information was generated (filled by child).

5.41.2.12 unsigned long long int DataStaging::DataDeliveryComm::Status::transfered

Number of bytes transfered.

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

- DataDeliveryComm.h

5.42 Arc::DataHandle Class Reference

This class is a wrapper around the DataPoint class.

`#include <DataHandle.h>`

Public Member Functions

- DataHandle (const URL &url, const UserConfig &usercfg)
- ∼DataHandle ()
- DataPoint ∗ operator → ()
- const DataPoint ∗ operator → () const
- DataPoint & operator ∗ ()
- const DataPoint & operator ∗ () const
- bool operator! () const
- operator bool () const

5.42.1 Detailed Description

This class is a wrapper around the DataPoint class.

It simplifies the construction, use and destruction of DataPoint objects and should be used instead of DataPoint classes directly. The appropriate DataPoint subclass is created automatically and stored internally in DataHandle. A DataHandle instance can be thought of as a pointer to the DataPoint instance and the DataPoint can be accessed through the usual dereference operators. A DataHandle cannot be copied.

5.42.2 Constructor & Destructor Documentation

5.42.2.1 Arc::DataHandle::DataHandle (const URL & *url*, const UserConfig & *usercfg*) `[inline]`

Construct a new DataHandle.

5.42.2.2 Arc::DataHandle::∼DataHandle () `[inline]`

Destructor.

5.42.3 Member Function Documentation

5.42.3.1 const DataPoint& Arc::DataHandle::operator ∗ (void) const `[inline]`

Returns a const reference to a DataPoint object.

5.42.3.2 DataPoint& Arc::DataHandle::operator ∗ (void) `[inline]`

Returns a reference to a DataPoint object.

5.42.3.3 Arc::DataHandle::operator bool (void) const `[inline]`

Returns true if the DataHandle is valid.

5.42.3.4 bool Arc::DataHandle::operator! (void) const `[inline]`

Returns true if the DataHandle is not valid.

5.42.3.5 const DataPoint∗ Arc::DataHandle::operator → (void) const `[inline]`

Returns a const pointer to a DataPoint object.

5.42.3.6 DataPoint∗ Arc::DataHandle::operator → (void) `[inline]`

Returns a pointer to a DataPoint object.

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

- DataHandle.h

5.43 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)

5.43.1 Detailed Description

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

5.43.2 Constructor & Destructor Documentation

5.43.2.1 Arc::DataMover::DataMover ()

Constructor.

5.43.2.2 Arc::DataMover::∼DataMover ()

Destructor.

5.43.3 Member Function Documentation

5.43.3.1 void Arc::DataMover::checks (bool *v*)

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

Parameters:

v true if allowed (default is true).

5.43.3.2 bool Arc::DataMover::checks ()

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

5.43.3.3 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.

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

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

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

Set if transfer will be retried in case of failure.

5.43.3.6 bool Arc::DataMover::retry ()

Check if transfer will be retried in case of failure.

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

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

5.43.3.8 void Arc::DataMover::set_default_max_inactivity_time (time_t *max_inactivity_time*) `[inline]`

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

5.43.3.9 void Arc::DataMover::set_default_min_average_speed (unsigned long long int *min_average_speed*) `[inline]`

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

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

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

5.43.3.11 DataStatus Arc::DataMover::Transfer (DataPoint & *source*, DataPoint & *destination*, FileCache & *cache*, const URLMap & *map*, unsigned long long int *min_speed*, time_t *min_speed_time*, unsigned long long int *min_average_speed*, time_t *max_inactivity_time*, callback *cb* = `NULL`, void ∗ *arg* = `NULL`, const char ∗ *prefix* = `NULL`)

Initiates transfer from 'source' to 'destination'.

Parameters:

min_speed minimal allowed current speed.

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

min_average_speed minimal allowed average speed.

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

5.43.3.12 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.

destination destination URL.

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

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

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

arg passed to 'cb'.

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

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

Activate printing information about transfer status.

Parameters:

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

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

Activate printing information about transfer status.

5.43.3.15 bool Arc::DataMover::verbose ()

Check if printing information about transfer status is activated.

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

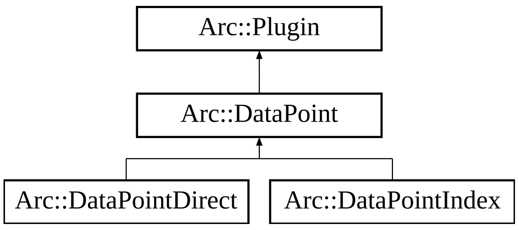
- DataMover.h

5.44 Arc::DataPoint Class Reference

A DataPoint represents a data resource and is an abstraction of a URL.

`#include <DataPoint.h>`

Inheritance diagram for Arc::DataPoint::



Public Types

- ACCESS_LATENCY_ZERO
- ACCESS_LATENCY_SMALL
- ACCESS_LATENCY_LARGE
- INFO_TYPE_MINIMAL = 0
- 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
- enum DataPointAccessLatency { ACCESS_LATENCY_ZERO, ACCESS_LATENCY_SMALL, ACCESS_LATENCY_LARGE }
- enum DataPointInfoType {

  INFO_TYPE_MINIMAL = 0, 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

- 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, const std::list< std::string > &transport_protocols)

- virtual DataStatus PrepareWriting (unsigned int timeout, unsigned int &wait_time, const std::list< std::string > &transport_protocols)
- 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 () const =0
- virtual bool ProvidesMeta () const =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 DataStatus ClearLocations ()=0
- virtual int AddCheckSumObject (CheckSum ∗cksum)=0
- virtual const CheckSum ∗ GetCheckSumObject (int index) const =0
- virtual void SortLocations (const std::string &pattern, const URLMap &url_map)=0

Protected Member Functions

- DataPoint (const URL &url, const UserConfig &usercfg)

Protected Attributes

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

5.44.1 Detailed Description

A DataPoint represents a data resource and is an abstraction of a URL.

DataPoint uses ARC's Plugin mechanism to dynamically load the required Data Manager Component (DMC) when necessary. A DMC typically defines a subclass of DataPoint (e.g. DataPointHTTP) and is responsible for a specific protocol (e.g. http). DataPoints should not be used directly, instead the DataHandle wrapper class should be used, which automatically loads the correct DMC.

DataPoint defines methods for access to the data resource. To transfer data between two DataPoints, DataMover::Transfer() can be used.

There are two subclasses of DataPoint, DataPointDirect and DataPointIndex. None of these three classes can be instantiated directly. DataPointDirect and its subclasses handle "physical" resources through protocols such as file, http and gsiftp. These classes implement methods such as StartReading() and StartWriting(). DataPointIndex and its subclasses handle resources such as indexes and catalogs and implement methods like Resolve() and PreRegister().

When creating a new DMC, a subclass of either DataPointDirect or DataPointIndex should be created, and the appropriate methods implemented. DataPoint itself has no direct external dependencies, but plugins may rely on third-party components. The new DMC must also add itself to the list of available plugins and provide an Instance() method which returns a new instance of itself, if the supplied arguments are valid for the protocol. Here is an example implementation of a new DMC for protocol MyProtocol which represents a physical resource accessible through protocol my://

```
#include <arc/data/DataPointDirect.h>

namespace Arc {

class DataPointMyProtocol : public DataPointDirect {
 public:
  DataPointMyProtocol(const URL& url, const UserConfig& usercfg);
  static Plugin* Instance(PluginArgument *arg);
  virtual DataStatus StartReading(DataBuffer& buffer);
  ...
};

DataPointMyProtocol::DataPointMyProtocol(const URL& url, const UserConfig& usercfg) {
  ...
}

DataPointMyProtocol::StartReading(DataBuffer& buffer) { ... }

...

Plugin* DataPointMyProtocol::Instance(PluginArgument *arg) {
  DataPointPluginArgument *dmcarg = dynamic_cast<DataPointPluginArgument*>(arg);
  if (!dmcarg)
    return NULL;
  if (((const URL &)(*dmcarg)).Protocol() != "my")
    return NULL;
  return new DataPointMyProtocol(*dmcarg, *dmcarg);
}

} // namespace Arc

Arc::PluginDescriptor PLUGINS_TABLE_NAME[] = {
  { "my", "HED:DMC", 0, &Arc::DataPointMyProtocol::Instance },
  { NULL, NULL, 0, NULL }
};
```

5.44.2 Member Enumeration Documentation

5.44.2.1 enum Arc::DataPoint::DataPointAccessLatency

Describes the latency to access this URL.

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 can be accessed instantly.

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

ACCESS_LATENCY_LARGE URL has a large access latency, for example staged from tape.

5.44.2.2 enum Arc::DataPoint::DataPointInfoType

Describes type of information about URL to request.

Enumerator:

INFO_TYPE_MINIMAL Whatever protocol can get with no additional effort.

INFO_TYPE_NAME	Only name of object (relative).
INFO_TYPE_TYPE	Type of object - currently file or dir.
INFO_TYPE_TIMES	Timestamps associated with object.
INFO_TYPE_CONTENT	Metadata describing content, like size, checksum, etc.
INFO_TYPE_ACCESS	Access control - ownership, permission, etc.
INFO_TYPE_STRUCT	Fine structure - replicas, transfer locations, redirections.
INFO_TYPE_REST	All the other parameters.
INFO_TYPE_ALL	All the parameters.

5.44.3 Constructor & Destructor Documentation

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

Destructor.

5.44.3.2 Arc::DataPoint::DataPoint (const URL & *url*, const UserConfig & *usercfg*) `[protected]`

Constructor.

Constructor is protected because DataPoints should not be created directly. Subclasses should however call this in their constructors to set various common attributes.

Parameters:

url	The URL representing the DataPoint
usercfg	User configuration object

5.44.4 Member Function Documentation

5.44.4.1 virtual bool Arc::DataPoint::AcceptsMeta () const `[pure virtual]`

If endpoint can have any use from meta information.

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

5.44.4.2 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 DataPointer itself.

Returns:

integer position in the list of checksum objects.

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

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

Add URL to list.

Parameters:

url Location URL to add.

meta Location meta information.

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

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

Get suggested number of buffers for transfers.

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

5.44.4.5 virtual long long int Arc::DataPoint::BufSize () const `[pure virtual]`

Get suggested buffer size for transfers.

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

5.44.4.6 virtual bool Arc::DataPoint::Cache () const `[virtual]`

Returns true if file is cacheable.

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

Query the DataPoint to check if object is accessible.

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

Implemented in Arc::DataPointIndex.

5.44.4.8 virtual bool Arc::DataPoint::CheckCheckSum () const `[virtual]`

Check if meta-information 'checksum' is available.

5.44.4.9 virtual bool Arc::DataPoint::CheckCreated () const `[virtual]`

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

5.44.4.10 virtual bool Arc::DataPoint::CheckSize () const `[virtual]`

Check if meta-information 'size' is available.

5.44.4.11 virtual bool Arc::DataPoint::CheckValid () const `[virtual]`

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

5.44.4.12 virtual DataStatus Arc::DataPoint::ClearLocations () `[pure virtual]`

Remove all locations.

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

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

Compare metadata of DataPoint and current location.

Returns inconsistency error or error encountered during operation, or success

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

5.44.4.14 virtual bool Arc::DataPoint::CompareMeta (const DataPoint & *p*) const `[virtual]`

Compare meta information from another object.

Undefined values are not used for comparison.

Parameters:

p object to which to compare.

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

Returns current (resolved) URL.

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

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

Returns meta information used to create current URL.

Usage differs between different indexing services.

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

5.44.4.17 virtual const std::string Arc::DataPoint::DefaultCheckSum () const `[virtual]`

Default checksum type.

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

Finish reading from the URL.

Must be called after transfer of physical file has completed and if PrepareReading() 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.

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

Finish writing to the URL.

Must be called after transfer of physical file has completed and if PrepareWriting() 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.

5.44.4.20 virtual DataPointAccessLatency Arc::DataPoint::GetAccessLatency () const `[virtual]`

Get value of meta-information ’access latency’.

Reimplemented in Arc::DataPointIndex.

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

Check if additional checks before transfer will be performed.

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

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

Get value of meta-information ’checksum’.

5.44.4.23 virtual const CheckSum∗ Arc::DataPoint::GetCheckSumObject (int *index*) const `[pure virtual]`

Get CheckSum object at given position in list.

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

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

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

5.44.4.25 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.

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

Check if heavy security during data transfer is allowed.

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

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

Get value of meta-information 'size'.

5.44.4.28 virtual int Arc::DataPoint::GetTries () const `[virtual]`

Returns number of retries left.

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

Returns the URL that was passed to the constructor.

5.44.4.30 virtual const UserConfig& Arc::DataPoint::GetUserConfig () const `[virtual]`

Returns the UserConfig that was passed to the constructor.

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

Get value of meta-information 'validity time'.

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

Returns true if number of resolved URLs is not 0.

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

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

Check if URL is an Indexing Service.

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

5.44.4.34 virtual bool Arc::DataPoint::IsStageable () const `[virtual]`

If URL should be staged or queried for Transport URL (TURL).

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

5.44.4.35 virtual bool Arc::DataPoint::LastLocation () `[pure virtual]`

Returns true if the current location is the last.

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

5.44.4.36 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 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.

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

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

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

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

Returns false if out of retries.

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

5.44.4.39 virtual bool Arc::DataPoint::NextLocation () `[pure virtual]`

Switch to next location in list of URLs.

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

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

5.44.4.40 virtual void Arc::DataPoint::NextTry (void) `[virtual]`

Decrease number of retries left.

5.44.4.41 virtual Arc::DataPoint::operator bool () const `[virtual]`

Is DataPoint valid?

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

Is DataPoint valid?

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

Request passive transfers for FTP-like protocols.

Parameters:

true to request.

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

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

Index Service 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 under same name.

Implemented in Arc::DataPointDirect.

5.44.4.45 virtual DataStatus Arc::DataPoint::PrepareReading (unsigned int *timeout*, unsigned int & *wait_time*, const std::list< std::string > & *transport_protocols*) `[virtual]`

Prepare DataPoint 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() 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(). When file preparation has finished, the physical file(s) to read from can be found from TransferLocations().

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.

transport_protocols A list of possible transport protocols for the physical file in order of preference.

Reimplemented in Arc::DataPointIndex.

5.44.4.46 virtual DataStatus Arc::DataPoint::PrepareWriting (unsigned int *timeout*, unsigned int & *wait_time*, const std::list< std::string > & *transport_protocols*) `[virtual]`

Prepare DataPoint 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() 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().

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.

transport_protocols A list of possible transport protocols for the physical file in order of preference.

Reimplemented in Arc::DataPointIndex.

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

Index service preregistration.

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

Parameters:

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

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

Implemented in Arc::DataPointDirect.

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

Index Service preunregistration.

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

Parameters:

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

Implemented in Arc::DataPointDirect.

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

If endpoint can provide at least some meta information directly.

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

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

Set range of bytes to retrieve.

Default values correspond to whole file.

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

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

Parameters:

v true if allowed (default is false).

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

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

Check if file is registered in Indexing Service.

Proper value is obtainable only after Resolve.

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

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

Remove/delete object at URL.

Implemented in Arc::DataPointIndex.

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

Remove current URL from list.

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

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

Remove locations present in another DataPoint object.

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

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

Resolves index service URL into list of ordinary URLs.

Also obtains meta information about the file.

Parameters:

source true if DataPoint object represents source of information.

Implemented in Arc::DataPointDirect.

5.44.4.57 virtual void Arc::DataPoint::SetAccessLatency (const DataPointAccessLatency & *latency*) `[virtual]`

Set value of meta-information 'access latency'.

5.44.4.58 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, and Arc::DataPointIndex.

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

Set value of meta-information 'checksum'.

Reimplemented in Arc::DataPointIndex.

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

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

5.44.4.61 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.

5.44.4.62 virtual void Arc::DataPoint::SetSecure (bool *v*) `[pure virtual]`

Allow/disallow heavy security during data transfer.

Parameters:

v true if allowed (default depends on protocol).

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

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

Set value of meta-information 'size'.

Reimplemented in Arc::DataPointIndex.

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

Set number of retries.

Reimplemented in Arc::DataPointIndex.

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

Assigns new URL. 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 is not suitable or method is not implemented false is returned.

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

Set value of meta-information 'validity time'.

5.44.4.67 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, and Arc::DataPointIndex.

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

Start reading data from URL.

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

Parameters:

buffer operation will use this buffer to put information into. Should not be destroyed before StopReading() was called and returned.

Implemented in Arc::DataPointIndex.

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

Start writing data to URL.

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

Parameters:

buffer operation will use this buffer to get information from. Should not be destroyed before stop_writing was called and returned.

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

Implemented in Arc::DataPointIndex.

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

Retrieve information about this object.

If the DataPoint 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.

5.44.4.71 virtual DataStatus Arc::DataPoint::StopReading () `[pure virtual]`

Stop reading.

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

Implemented in Arc::DataPointIndex.

5.44.4.72 virtual DataStatus Arc::DataPoint::StopWriting () `[pure virtual]`

Stop writing.

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

Implemented in Arc::DataPointIndex.

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

Returns a string representation of the DataPoint.

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

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

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 from this list. It is up to the caller to choose the best URL and instantiate new DataPoint for handling it. For consistency protocols which do not require redirections return original URL. For protocols which need redirection calling StartReading and StartWriting will use first URL in the list.

Reimplemented in Arc::DataPointIndex.

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

Index Service unregistration.

Remove information about file registered in Indexing Service.

Parameters:

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

Implemented in Arc::DataPointDirect.

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

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

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

5.44.5 Field Documentation

5.44.5.1 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:

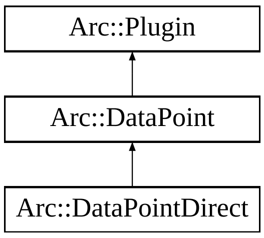
- DataPoint.h

5.45 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 const CheckSum ∗ GetCheckSumObject (int index) const
- 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 () const
- virtual bool ProvidesMeta () const
- 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 DataStatus ClearLocations ()
- virtual void SortLocations (const std::string &, const URLMap &)

5.45.1 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 to pass actual data. It also provides other operations like querying parameters of remote object. It is used by higher-level classes DataMove and DataMovePar to provide data transfer service for application.

5.45.2 Member Function Documentation

5.45.2.1 virtual bool Arc::DataPointDirect::AcceptsMeta () const `[virtual]`

If endpoint can have any use from meta information.

Implements Arc::DataPoint.

5.45.2.2 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.

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

Add URL to list.

Parameters:

url Location URL to add.

meta Location meta information.

Implements Arc::DataPoint.

5.45.2.4 virtual int Arc::DataPointDirect::BufNum () const `[virtual]`

Get suggested number of buffers for transfers.

Implements Arc::DataPoint.

5.45.2.5 virtual long long int Arc::DataPointDirect::BufSize () const `[virtual]`

Get suggested buffer size for transfers.

Implements Arc::DataPoint.

5.45.2.6 virtual DataStatus Arc::DataPointDirect::ClearLocations () `[virtual]`

Remove all locations.

Implements Arc::DataPoint.

5.45.2.7 virtual DataStatus Arc::DataPointDirect::CompareLocationMetadata () const `[virtual]`

Compare metadata of DataPoint and current location.

Returns inconsistency error or error encountered during operation, or success

Implements Arc::DataPoint.

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

Returns current (resolved) URL.

Implements Arc::DataPoint.

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

Returns meta information used to create current URL.

Usage differs between different indexing services.

Implements Arc::DataPoint.

5.45.2.10 virtual bool Arc::DataPointDirect::GetAdditionalChecks () const `[virtual]`

Check if additional checks before transfer will be performed.

Implements Arc::DataPoint.

5.45.2.11 virtual const CheckSum* Arc::DataPointDirect::GetCheckSumObject (int *index*) const `[virtual]`

Get CheckSum object at given position in list.

Implements Arc::DataPoint.

5.45.2.12 virtual bool Arc::DataPointDirect::GetSecure () const `[virtual]`

Check if heavy security during data transfer is allowed.

Implements Arc::DataPoint.

5.45.2.13 virtual bool Arc::DataPointDirect::HaveLocations () const `[virtual]`

Returns true if number of resolved URLs is not 0.

Implements Arc::DataPoint.

5.45.2.14 virtual bool Arc::DataPointDirect::IsIndex () const `[virtual]`

Check if URL is an Indexing Service.

Implements Arc::DataPoint.

5.45.2.15 virtual bool Arc::DataPointDirect::IsStageable () const `[virtual]`

If URL should be staged or queried for Transport URL (TURL).

Reimplemented from Arc::DataPoint.

5.45.2.16 virtual bool Arc::DataPointDirect::LastLocation () `[virtual]`

Returns true if the current location is the last.

Implements Arc::DataPoint.

5.45.2.17 virtual bool Arc::DataPointDirect::Local () const `[virtual]`

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

Implements Arc::DataPoint.

5.45.2.18 virtual bool Arc::DataPointDirect::LocationValid () const `[virtual]`

Returns false if out of retries.

Implements Arc::DataPoint.

5.45.2.19 virtual bool Arc::DataPointDirect::NextLocation () `[virtual]`

Switch to next location in list of URLs.

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

Implements Arc::DataPoint.

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

Request passive transfers for FTP-like protocols.

Parameters:

true to request.

Implements Arc::DataPoint.

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

Index Service 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 under same name.

Implements Arc::DataPoint.

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

Index service preregistration.

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

Parameters:

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

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

Implements Arc::DataPoint.

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

Index Service preunregistration.

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

Parameters:

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

Implements Arc::DataPoint.

5.45.2.24 virtual bool Arc::DataPointDirect::ProvidesMeta () const [virtual]

If endpoint can provide at least some meta information directly.

Implements Arc::DataPoint.

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

Set range of bytes to retrieve.

Default values correspond to whole file.

Implements Arc::DataPoint.

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

Parameters:

v true if allowed (default is false).

Implements Arc::DataPoint.

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

Check if file is registered in Indexing Service.

Proper value is obtainable only after Resolve.

Implements Arc::DataPoint.

5.45.2.28 virtual DataStatus Arc::DataPointDirect::RemoveLocation () `[virtual]`

Remove current URL from list.

Implements Arc::DataPoint.

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

Remove locations present in another DataPoint object.

Implements Arc::DataPoint.

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

Resolves index service URL into list of ordinary URLs.

Also obtains meta information about the file.

Parameters:

source true if DataPoint object represents source of information.

Implements Arc::DataPoint.

5.45.2.31 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.

5.45.2.32 virtual void Arc::DataPointDirect::SetSecure (bool *v*) `[virtual]`

Allow/disallow heavy security during data transfer.

Parameters:

v true if allowed (default depends on protocol).

Implements Arc::DataPoint.

5.45.2.33 virtual void Arc::DataPointDirect::SortLocations (const std::string &, const URLMap &) `[inline, virtual]`

Sort locations according to the specified pattern.

Parameters:

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

Implements Arc::DataPoint.

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

Index Service unregistration.

Remove information about file registered in Indexing Service.

Parameters:

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

Implements Arc::DataPoint.

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

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

Implements Arc::DataPoint.

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

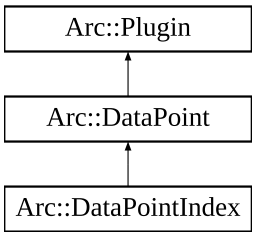
- DataPointDirect.h

5.46 Arc::DataPointIndex Class Reference

Complements DataPoint with attributes common for Indexing Service 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 ClearLocations ()
- 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 () const
- virtual bool ProvidesMeta () const
- 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, const std::list< std::string > &transport_protocols)
- virtual DataStatus PrepareWriting (unsigned int timeout, unsigned int &wait_time, const std::list< std::string > &transport_protocols)
- 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)
- virtual const CheckSum ∗ GetCheckSumObject (int index) const

5.46.1 Detailed Description

Complements DataPoint with attributes common for Indexing Service URLs.

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

5.46.2 Member Function Documentation

5.46.2.1 virtual bool Arc::DataPointIndex::AcceptsMeta () const `[virtual]`

If endpoint can have any use from meta information.

Implements Arc::DataPoint.

5.46.2.2 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.

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

Add URL to list.

Parameters:

- ***url*** Location URL to add.
- ***meta*** Location meta information.

Implements Arc::DataPoint.

5.46.2.4 virtual int Arc::DataPointIndex::BufNum () const `[virtual]`

Get suggested number of buffers for transfers.

Implements Arc::DataPoint.

5.46.2.5 virtual long long int Arc::DataPointIndex::BufSize () const `[virtual]`

Get suggested buffer size for transfers.

Implements Arc::DataPoint.

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

Query the DataPoint to check if object is accessible.

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

Implements Arc::DataPoint.

5.46.2.7 virtual DataStatus Arc::DataPointIndex::ClearLocations () `[virtual]`

Remove all locations.

Implements Arc::DataPoint.

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

Compare metadata of DataPoint and current location.

Returns inconsistency error or error encountered during operation, or success

Implements Arc::DataPoint.

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

Returns current (resolved) URL.

Implements Arc::DataPoint.

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

Returns meta information used to create current URL.

Usage differs between different indexing services.

Implements Arc::DataPoint.

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

Finish reading from the URL.

Must be called after transfer of physical file has completed and if PrepareReading() 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.

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

Finish writing to the URL.

Must be called after transfer of physical file has completed and if PrepareWriting() 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.

5.46.2.13 virtual DataPointAccessLatency Arc::DataPointIndex::GetAccessLatency () const `[virtual]`

Get value of meta-information 'access latency'.

Reimplemented from Arc::DataPoint.

5.46.2.14 virtual bool Arc::DataPointIndex::GetAdditionalChecks () const `[virtual]`

Check if additional checks before transfer will be performed.

Implements Arc::DataPoint.

5.46.2.15 virtual const CheckSum∗ Arc::DataPointIndex::GetCheckSumObject (int *index*) const `[virtual]`

Get CheckSum object at given position in list.

Implements Arc::DataPoint.

5.46.2.16 virtual bool Arc::DataPointIndex::GetSecure () const `[virtual]`

Check if heavy security during data transfer is allowed.

Implements Arc::DataPoint.

5.46.2.17 virtual bool Arc::DataPointIndex::HaveLocations () const `[virtual]`

Returns true if number of resolved URLs is not 0.

Implements Arc::DataPoint.

5.46.2.18 virtual bool Arc::DataPointIndex::IsIndex () const `[virtual]`

Check if URL is an Indexing Service.

Implements Arc::DataPoint.

5.46.2.19 virtual bool Arc::DataPointIndex::IsStageable () const `[virtual]`

If URL should be staged or queried for Transport URL (TURL).

Reimplemented from Arc::DataPoint.

5.46.2.20 virtual bool Arc::DataPointIndex::LastLocation () `[virtual]`

Returns true if the current location is the last.

Implements Arc::DataPoint.

5.46.2.21 virtual bool Arc::DataPointIndex::Local () const `[virtual]`

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

Implements Arc::DataPoint.

5.46.2.22 virtual bool Arc::DataPointIndex::LocationValid () const `[virtual]`

Returns false if out of retries.

Implements Arc::DataPoint.

5.46.2.23 virtual bool Arc::DataPointIndex::NextLocation () `[virtual]`

Switch to next location in list of URLs.

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

Implements Arc::DataPoint.

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

Request passive transfers for FTP-like protocols.

Parameters:

true to request.

Implements Arc::DataPoint.

5.46.2.25 virtual DataStatus Arc::DataPointIndex::PrepareReading (unsigned int *timeout*, unsigned int & *wait_time*, const std::list< std::string > & *transport_protocols*) `[virtual]`

Prepare DataPoint 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() 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(). When file preparation has finished, the physical file(s) to read from can be found from TransferLocations().

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.

transport_protocols A list of possible transport protocols for the physical file in order of preference.

Reimplemented from Arc::DataPoint.

5.46.2.26 virtual DataStatus Arc::DataPointIndex::PrepareWriting (unsigned int *timeout*, unsigned int & *wait_time*, const std::list< std::string > & *transport_protocols*) `[virtual]`

Prepare DataPoint 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() 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().

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.

transport_protocols A list of possible transport protocols for the physical file in order of preference.

Reimplemented from Arc::DataPoint.

5.46.2.27 virtual bool Arc::DataPointIndex::ProvidesMeta () const `[virtual]`

If endpoint can provide at least some meta information directly.

Implements Arc::DataPoint.

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

Set range of bytes to retrieve.

Default values correspond to whole file.

Implements Arc::DataPoint.

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

Parameters:

v true if allowed (default is false).

Implements Arc::DataPoint.

5.46.2.30 virtual bool Arc::DataPointIndex::Registered () const `[virtual]`

Check if file is registered in Indexing Service.

Proper value is obtainable only after Resolve.

Implements Arc::DataPoint.

5.46.2.31 virtual DataStatus Arc::DataPointIndex::Remove () `[virtual]`

Remove/delete object at URL.

Implements Arc::DataPoint.

5.46.2.32 virtual DataStatus Arc::DataPointIndex::RemoveLocation () `[virtual]`

Remove current URL from list.

Implements Arc::DataPoint.

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

Remove locations present in another DataPoint object.

Implements Arc::DataPoint.

5.46.2.34 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.

5.46.2.35 virtual void Arc::DataPointIndex::SetCheckSum (const std::string & *val*) `[virtual]`

Set value of meta-information 'checksum'.

Reimplemented from Arc::DataPoint.

5.46.2.36 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.

5.46.2.37 virtual void Arc::DataPointIndex::SetSecure (bool *v*) `[virtual]`

Allow/disallow heavy security during data transfer.

Parameters:

v true if allowed (default depends on protocol).

Implements Arc::DataPoint.

5.46.2.38 virtual void Arc::DataPointIndex::SetSize (const unsigned long long int *val*) `[virtual]`

Set value of meta-information 'size'.

Reimplemented from Arc::DataPoint.

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

Set number of retries.

Reimplemented from Arc::DataPoint.

5.46.2.40 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.

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

Start reading data from URL.

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

Parameters:

buffer operation will use this buffer to put information into. Should not be destroyed before StopReading() was called and returned.

Implements Arc::DataPoint.

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

Start writing data to URL.

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

Parameters:

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

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

Implements Arc::DataPoint.

5.46.2.43 virtual DataStatus Arc::DataPointIndex::StopReading () `[virtual]`

Stop reading.

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

Implements Arc::DataPoint.

5.46.2.44 virtual DataStatus Arc::DataPointIndex::StopWriting () [virtual]

Stop writing.

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

Implements Arc::DataPoint.

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

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

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 from this list. It is up to the caller to choose the best URL and instantiate new DataPoint for handling it. For consistency protocols which do not require redirections return original URL. For protocols which need redirection calling StartReading and StartWriting will use first URL in the list.

Reimplemented from Arc::DataPoint.

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

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

Implements Arc::DataPoint.

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

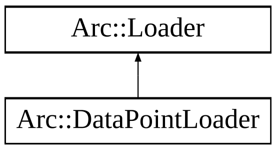
- DataPointIndex.h

5.47 Arc::DataPointLoader Class Reference

Class used by DataHandle to load the required DMC.

`#include <DataPoint.h>`

Inheritance diagram for Arc::DataPointLoader::



5.47.1 Detailed Description

Class used by DataHandle to load the required DMC.

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

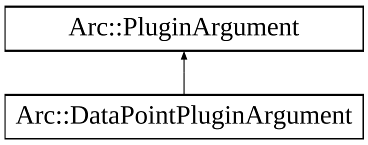
- DataPoint.h

5.48 Arc::DataPointPluginArgument Class Reference

Class representing the arguments passed to DMC plugins.

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

Inheritance diagram for Arc::DataPointPluginArgument::



5.48.1 Detailed Description

Class representing the arguments passed to DMC plugins.

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

- DataPoint.h

5.49 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)

5.49.1 Detailed Description

Keeps track of average and instantaneous transfer speed.

Also detects data transfer inactivity and other transfer timeouts.

5.49.2 Constructor & Destructor Documentation

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

Constructor

Parameters:

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

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

Constructor

Parameters:

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

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

min_speed_time

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

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

5.49.2.3 Arc::DataSpeed::∼DataSpeed (void)

Destructor.

5.49.3 Member Function Documentation

5.49.3.1 time_t Arc::DataSpeed::get_max_inactivity_time () `[inline]`

Get inactivity timeout.

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

Turn off speed control.

Parameters:

disable true to turn off.

5.49.3.3 bool Arc::DataSpeed::max_inactivity_time_failure () `[inline]`

Check if maximal inactivity time error was triggered.

5.49.3.4 bool Arc::DataSpeed::min_average_speed_failure () `[inline]`

Check if minimal average speed error was triggered.

5.49.3.5 bool Arc::DataSpeed::min_speed_failure () `[inline]`

Check if minimal speed error was triggered.

5.49.3.6 void Arc::DataSpeed::reset (void)

Reset all counters and triggers.

5.49.3.7 void Arc::DataSpeed::set_base (time_t *base_* = `DATASPEED_AVERAGING_PERIOD`)

Set averaging time period.

Parameters:

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

5.49.3.8 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.

5.49.3.9 void Arc::DataSpeed::set_max_inactivity_time (time_t *max_inactivity_time*)

Set inactivity tiemout.

Parameters:

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

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

Set minmal avaerage speed.

Parameters:

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

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

Set minimal allowed speed.

Parameters:

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

min_speed_time

5.49.3.12 void Arc::DataSpeed::set_progress_indicator (show_progress_t *func* = `NULL`)

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

Parameters:

pointer to function which prints information.

5.49.3.13 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).

5.49.3.14 unsigned long long int Arc::DataSpeed::transferred_size (void) `[inline]`

Returns amount of data this object knows about.

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

Check if speed information is going to be printed.

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

Print information about current speed and amout of data.

Parameters:

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

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

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

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

- DataSpeed.h

5.50 Arc::DataStatus Class Reference

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

Public Types

- 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

- 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, LocationAlreadyExists-Error = 19,
  NotSupportedForDirectDataPointsError = 20, UnimplementedError = 21, IsReadingError = 22, Is-WritingError = 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 }

5.50.1 Detailed Description

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

5.50.2 Member Enumeration Documentation

5.50.2.1 enum Arc::DataStatus::DataStatusType

Enumerator:

Success Operation completed successfully.

ReadAcquireError Source is bad URL or can't be used due to some reason.

WriteAcquireError Destination is bad URL or can't be used due to some reason.

ReadResolveError Resolving of index service URL for source failed.

WriteResolveError Resolving of index service URL 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 failed.

PostRegisterError Last stage of registration of index service URL failed.

UnregisterError Unregistration of index service URL failed.

CacheError Error in caching procedure.

CredentialsExpiredError Error due to provided credentials are expired.

DeleteError Error deleting location or URL.

NoLocationError No valid location available.

LocationAlreadyExistsError No valid location available.

NotSupportedForDirectDataPointsError Operation has no sense for this kind of URL.

UnimplementedError Feature is unimplemented.

IsReadingError DataPoint is already reading.

IsWritingError DataPoint 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.

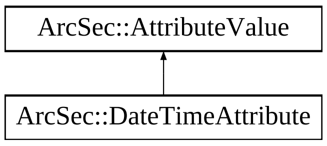
The documentation for this class was generated from the following file:

- DataStatus.h

5.51 ArcSec::DateTimeAttribute Class Reference

`#include <DateTimeAttribute.h>`

Inheritance diagram for ArcSec::DateTimeAttribute::



Public Member Functions

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

5.51.1 Detailed Description

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

5.51.2 Member Function Documentation

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

encode the value in a string format

Implements ArcSec::AttributeValue.

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

Evluate whether "this" equale to the parameter value

Implements ArcSec::AttributeValue.

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

Get the AttributeId of the <Attribute>

Implements ArcSec::AttributeValue.

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

Get the DataType of the <Attribute>

Implements ArcSec::AttributeValue.

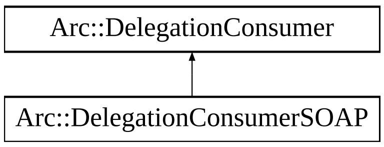
The documentation for this class was generated from the following file:

- DateTimeAttribute.h

5.52 Arc::DelegationConsumer Class Reference

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

Inheritance diagram for Arc::DelegationConsumer::



Public Member Functions

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

Protected Member Functions

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

5.52.1 Detailed Description

A consumer of delegated X509 credentials. During delegation procedure this class acquires delegated credentials aka proxy - certificate, private key and chain of previous certificates. Delegation procedure consists of calling Request() method for generating certificate request followed by call to Acquire() method for making complete credentials from certificate chain.

5.52.2 Constructor & Destructor Documentation

5.52.2.1 Arc::DelegationConsumer::DelegationConsumer (void)

Creates object with new private key

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

Creates object with provided private key

5.52.3 Member Function Documentation

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

Includes the functionality of Acquire(content) plus extracting the credential identity.

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

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

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

Stores content of this object into a string

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

Private key

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

Return identifier of this object - not implemented

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

Creates private key

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

Make X509 certificate request from internal private key

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

Restores content of object from string

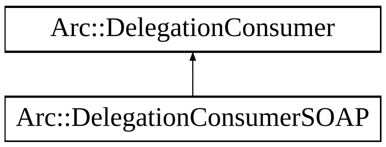
The documentation for this class was generated from the following file:

- DelegationInterface.h

5.53 Arc::DelegationConsumerSOAP Class Reference

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

Inheritance diagram for Arc::DelegationConsumerSOAP::



Public Member Functions

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

5.53.1 Detailed Description

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

5.53.2 Constructor & Destructor Documentation

5.53.2.1 Arc::DelegationConsumerSOAP::DelegationConsumerSOAP (void)

Creates object with new private key

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

Creates object with specified private key

5.53.3 Member Function Documentation

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

Process SOAP message which starts delagation. Generated message in 'out' is meant to be sent back to DelagationProviderSOAP. Argument 'id' contains identifier of procedure and is used only to produce SOAP message.

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

Similar to UpdateCredentials but takes only DelegatedToken XML element

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

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

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

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

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

- DelegationInterface.h

5.54 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_

5.54.1 Detailed Description

Manages multiple delegated credentials. Delegation consumers are created automatically with DelegateCredentialsInit method up to max_size_ and assigned unique identifier. It's methods are similar to those of DelegationConsumerSOAP with identifier included in SOAP message used to route execution to one of managed DelegationConsumerSOAP instances.

5.54.2 Member Function Documentation

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

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

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

See DelegationConsumerSOAP::DelegatedToken

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

See DelegationConsumerSOAP::UpdateCredentials

5.54.3 Field Documentation

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

If true delegation consumer is deleted when connection context is destroyed

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

Lifetime of unused delegation consumer

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

Max. number of delegation consumers

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

Max. times same delegation consumer may accept credentials

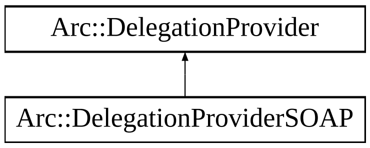
The documentation for this class was generated from the following file:

- DelegationInterface.h

5.55 Arc::DelegationProvider Class Reference

`#include <DelegationInterface.h>`

Inheritance diagram for Arc::DelegationProvider::



Public Member Functions

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

5.55.1 Detailed Description

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

5.55.2 Constructor & Destructor Documentation

5.55.2.1 Arc::DelegationProvider::DelegationProvider (const std::string & *credentials*)

Creates instance from provided credentials. Credentials are used to sign delegated credentials. Arguments should contain PEM-encoded certificate, private key and optionally certificates chain.

5.55.2.2 Arc::DelegationProvider::DelegationProvider (const std::string & *cert_file*, const std::string & *key_file*, std::istream ∗ *inpwd* = `NULL`)

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

5.55.3 Member Function Documentation

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

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

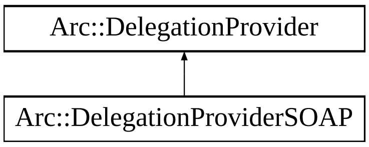
The documentation for this class was generated from the following file:

- DelegationInterface.h

5.56 Arc::DelegationProviderSOAP Class Reference

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

Inheritance diagram for Arc::DelegationProviderSOAP::



Public Member Functions

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

5.56.1 Detailed Description

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

5.56.2 Constructor & Destructor Documentation

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

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

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

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

5.56.3 Member Function Documentation

5.56.3.1 bool Arc::DelegationProviderSOAP::DelegateCredentialsInit (MCCInterface & *mcc_interface*, MessageAttributes ∗ *attributes_in*, MessageAttributes ∗ *attributes_out*, MessageContext ∗ *context*, ServiceType *stype* = `ARCDelegation`)

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

5.56.3.2 bool Arc::DelegationProviderSOAP::DelegateCredentialsInit (MCCInterface & *mcc_interface*, MessageContext ∗ *context*, ServiceType *stype* = `ARCDelegation`)

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

5.56.3.3 bool Arc::DelegationProviderSOAP::DelegatedToken (XMLNode *parent*)

Generates DelegatedToken element. Element is created as child of provided XML element and contains structure described in delegation.wsdl.

5.56.3.4 const std::string& Arc::DelegationProviderSOAP::ID (void) `[inline]`

Returns the identifier provided by service accepting delegated credentials. This identifier may then be used to refer to credentials stored at service.

5.56.3.5 bool Arc::DelegationProviderSOAP::UpdateCredentials (MCCInterface & *mcc_interface*, MessageAttributes ∗ *attributes_in*, MessageAttributes ∗ *attributes_out*, MessageContext ∗ *context*, const DelegationRestrictions & *restrictions* = `DelegationRestrictions()`, ServiceType *stype* = `ARCDelegation`)

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

5.56.3.6 bool Arc::DelegationProviderSOAP::UpdateCredentials (MCCInterface & *mcc_interface*, MessageContext ∗ *context*, const DelegationRestrictions & *restrictions* = `DelegationRestrictions()`, ServiceType *stype* = `ARCDelegation`)

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

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

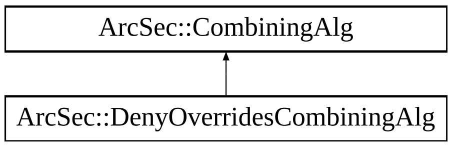
- DelegationInterface.h

5.57 ArcSec::DenyOverridesCombiningAlg Class Reference

Implement the "Deny-Overrides" algorithm.

`#include <DenyOverridesAlg.h>`

Inheritance diagram for ArcSec::DenyOverridesCombiningAlg::



Public Member Functions

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

5.57.1 Detailed Description

Implement the "Deny-Overrides" algorithm.

Deny-Overrides, scans the policy set which is given as the parameters of "combine" method, if gets "deny" result from any policy, then stops scanning and gives "deny" as result, otherwise gives "permit".

5.57.2 Member Function Documentation

5.57.2.1 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.

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

Get the identifier

Implements ArcSec::CombiningAlg.

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

- DenyOverridesAlg.h

5.58 DataStaging::DTR Class Reference

Data Transfer Request.

`#include <DTR.h>`

Public Member Functions

- DTR ()
- DTR (const DTR &dtr)
- DTR (const std::string &source, const std::string &destination, const Arc::UserConfig &usercfg, const std::string &jobid, const uid_t &uid, Arc::Logger ∗log)
- ∼DTR ()
- operator bool () const
- bool operator! () const
- void registerCallback (DTRCallback ∗cb, StagingProcesses owner)
- std::list< DTRCallback ∗ > get_callbacks (const std::map< StagingProcesses, std::list< DTRCallback ∗ > > &proc_callback, StagingProcesses owner)
- void reset ()
- std::string get_id () const
- std::string get_short_id () const
- Arc::DataHandle & get_source ()
- const Arc::DataHandle & get_source () const
- Arc::DataHandle & get_destination ()
- const Arc::DataHandle & get_destination () const
- const Arc::UserConfig & get_usercfg () const
- void set_timeout (time_t value)
- Arc::Time get_timeout () const
- void set_process_time (const Arc::Period &process_time)
- Arc::Time get_process_time () const
- Arc::Time get_creation_time () const
- std::string get_parent_job_id () const
- void set_priority (int pri)
- int get_priority () const
- void set_transfer_share (std::string share_name)
- std::string get_transfer_share () const
- void set_sub_share (const std::string &share)
- std::string get_sub_share () const
- void set_tries_left (unsigned int tries)
- unsigned int get_tries_left () const
- void decrease_tries_left ()
- void set_status (DTRStatus stat)
- DTRStatus get_status ()
- void set_error_status (DTRErrorStatus::DTRErrorStatusType error_stat, DTRErrorStatus::DTRErrorLocation error_loc, const std::string &desc="")
- void reset_error_status ()
- DTRErrorStatus get_error_status ()
- void set_cancel_request ()
- bool cancel_requested () const
- void set_cache_file (const std::string &filename)

- std::string get_cache_file () const
- void set_cache_parameters (const CacheParameters &param)
- const CacheParameters & get_cache_parameters () const
- void set_cache_state (CacheState state)
- CacheState get_cache_state () const
- void set_mapped_source (const std::string &file="")
- std::string get_mapped_source () const
- StagingProcesses get_owner () const
- Arc::User get_local_user () const
- void set_replication (bool rep)
- bool is_replication () const
- void set_force_registration (bool force)
- bool is_force_registration () const
- Arc::Logger ∗ get_logger () const
- void connect_logger ()
- void disconnect_logger ()
- void push (StagingProcesses new_owner)
- bool suspend ()
- bool error () const
- bool is_destined_for_pre_processor () const
- bool is_destined_for_post_processor () const
- bool is_destined_for_delivery () const
- bool came_from_pre_processor () const
- bool came_from_post_processor () const
- bool came_from_delivery () const
- bool came_from_generator () const
- bool is_in_final_state () const

5.58.1 Detailed Description

Data Transfer Request.

DTR stands for Data Transfer Request and a DTR describes a data transfer between two endpoints, a source and a destination. There are several parameters and options relating to the transfer contained in a DTR. The normal workflow is for a Generator to create a DTR and send it to the Scheduler for processing using dtr.push(SCHEDULER). If the Generator is a subclass of DTRCallback, when the Scheduler has finished with the DTR the receiveDTR() callback method is called.

registerCallback(this,DataStaging::GENERATOR) can be used to activate the callback. The following simple Generator code sample illustrates how to use DTRs:

```
class MyGenerator : public DTRCallback {
 public:
  void receiveDTR(DTR& dtr);
  void run();
 private:
  Arc::SimpleCondition cond;
};

void MyGenerator::receiveDTR(DTR& dtr) {
  // DTR received back, so notify waiting condition
  std::cout << "Received DTR " << dtr.get_id() << std::endl;
  cond.signal();
}
```

```
void MyGenerator::run() {
  // start Scheduler thread
  Scheduler scheduler;
  scheduler.start();

  // create a DTR
  DTR dtr(source, destination,...);

  // register this callback
  dtr.registerCallback(this,DataStaging::GENERATOR);
  // this line must be here in order to pass the DTR to the Scheduler
  dtr.registerCallback(&scheduler,DataStaging::SCHEDULER);

  // push the DTR to the Scheduler
  dtr.push(DataStaging::SCHEDULER);

  // wait until callback is called
  cond.wait();
  // DTR is finished, so stop Scheduler
  scheduler.stop();
}
```

A lock protects member variables that are likely to be accessed and modified by multiple threads.

5.58.2 Constructor & Destructor Documentation

5.58.2.1 DataStaging::DTR::DTR ()

Public empty constructor.

5.58.2.2 DataStaging::DTR::DTR (const DTR & *dtr*)

Copy constructor. Must be defined because DataHandle copy constructor is private.

5.58.2.3 DataStaging::DTR::DTR (const std::string & *source*, const std::string & *destination*, const Arc::UserConfig & *usercfg*, const std::string & *jobid*, const uid_t & *uid*, Arc::Logger ∗ *log*)

Normal constructor.

Construct a new DTR.

Parameters:

source Endpoint from which to read data

destination Endpoint to which to write data

usercfg Provides some user configuration information

jobid ID of the job associated with this data transfer

uid UID to use when accessing local file system if source or destination is a local file. If this is different to the current uid then the current uid must have sufficient privileges to change uid.

log Pointer to log object. If NULL the root logger is used.

5.58.2.4 DataStaging::DTR::∼DTR () `[inline]`

Empty destructor.

5.58.3 Member Function Documentation

5.58.3.1 bool DataStaging::DTR::came_from_delivery () const

Returns true if this DTR just came from delivery.

5.58.3.2 bool DataStaging::DTR::came_from_generator () const

Returns true if this DTR just came from the generator.

5.58.3.3 bool DataStaging::DTR::came_from_post_processor () const

Returns true if this DTR just came from the post-processor.

5.58.3.4 bool DataStaging::DTR::came_from_pre_processor () const

Returns true if this DTR just came from the pre-processor.

5.58.3.5 bool DataStaging::DTR::cancel_requested () const `[inline]`

Returns true if cancellation has been requested.

5.58.3.6 void DataStaging::DTR::connect_logger () `[inline]`

Connect log destinations to logger. Only needs to be done after disconnect().

5.58.3.7 void DataStaging::DTR::decrease_tries_left ()

Decrease attempt number.

5.58.3.8 void DataStaging::DTR::disconnect_logger () `[inline]`

Disconnect log destinations from logger.

5.58.3.9 bool DataStaging::DTR::error () const `[inline]`

Did an error happen?

5.58.3.10 std::string DataStaging::DTR::get_cache_file () const `[inline]`

Get cache filename.

5.58.3.11 const CacheParameters& DataStaging::DTR::get_cache_parameters () const `[inline]`

Get cache parameters.

5.58.3.12 CacheState DataStaging::DTR::get_cache_state () const `[inline]`

Get the cache state.

5.58.3.13 std::list<DTRCallback*> DataStaging::DTR::get_callbacks (const std::map< StagingProcesses, std::list< DTRCallback * > > & *proc_callback*, StagingProcesses *owner*)

Get the list of callbacks for this owner. Protected by lock.

5.58.3.14 Arc::Time DataStaging::DTR::get_creation_time () const `[inline]`

Get the creation time.

5.58.3.15 const Arc::DataHandle& DataStaging::DTR::get_destination () const `[inline]`

Get destination handle. Return by reference since DataHandle cannot be copied.

5.58.3.16 Arc::DataHandle& DataStaging::DTR::get_destination () `[inline]`

Get destination handle. Return by reference since DataHandle cannot be copied.

5.58.3.17 DTRErrorStatus DataStaging::DTR::get_error_status ()

Get the error status.

5.58.3.18 std::string DataStaging::DTR::get_id () const `[inline]`

Get the ID of this DTR.

5.58.3.19 Arc::User DataStaging::DTR::get_local_user () const `[inline]`

Get the local user information.

5.58.3.20 Arc::Logger* DataStaging::DTR::get_logger () const `[inline]`

Get Logger object, so that processes can log to this DTR's log.

5.58.3.21 std::string DataStaging::DTR::get_mapped_source () const `[inline]`

Get the mapped file.

5.58.3.22 StagingProcesses DataStaging::DTR::get_owner () const `[inline]`

Find the DTR owner.

5.58.3.23 std::string DataStaging::DTR::get_parent_job_id () const `[inline]`

Get the parent job ID.

5.58.3.24 int DataStaging::DTR::get_priority () const `[inline]`

Get the priority.

5.58.3.25 Arc::Time DataStaging::DTR::get_process_time () const `[inline]`

Get the next processing time for the DTR.

5.58.3.26 std::string DataStaging::DTR::get_short_id () const

Get an abbreviated version of the DTR ID - useful to reduce logging verbosity.

5.58.3.27 const Arc::DataHandle& DataStaging::DTR::get_source () const `[inline]`

Get source handle. Return by reference since DataHandle cannot be copied.

5.58.3.28 Arc::DataHandle& DataStaging::DTR::get_source () `[inline]`

Get source handle. Return by reference since DataHandle cannot be copied.

5.58.3.29 DTRStatus DataStaging::DTR::get_status ()

Get the status. Protected by lock.

5.58.3.30 std::string DataStaging::DTR::get_sub_share () const `[inline]`

Get sub-share.

5.58.3.31 Arc::Time DataStaging::DTR::get_timeout () const `[inline]`

Get the timeout for processing this DTR.

5.58.3.32 std::string DataStaging::DTR::get_transfer_share () const `[inline]`

Get the transfer share. sub_share is automatically added to transfershare.

5.58.3.33 unsigned int DataStaging::DTR::get_tries_left () const `[inline]`

Get the number of attempts remaining.

5.58.3.34 const Arc::UserConfig& DataStaging::DTR::get_usercfg () const `[inline]`

Get the UserConfig object associated with this DTR.

5.58.3.35 bool DataStaging::DTR::is_destined_for_delivery () const

Returns true if this DTR is about to go into delivery.

5.58.3.36 bool DataStaging::DTR::is_destined_for_post_processor () const

Returns true if this DTR is about to go into the post-processor.

5.58.3.37 bool DataStaging::DTR::is_destined_for_pre_processor () const

Returns true if this DTR is about to go into the pre-processor.

5.58.3.38 bool DataStaging::DTR::is_force_registration () const `[inline]`

Get force replication flag.

5.58.3.39 bool DataStaging::DTR::is_in_final_state () const

Returns true if this DTR is in a final state (finished, failed or cancelled).

5.58.3.40 bool DataStaging::DTR::is_replication () const `[inline]`

Get replication flag.

5.58.3.41 DataStaging::DTR::operator bool (void) const `[inline]`

Is DTR valid?

5.58.3.42 bool DataStaging::DTR::operator! (void) const `[inline]`

Is DTR not valid?

5.58.3.43 void DataStaging::DTR::push (StagingProcesses *new_owner*)

Pass the DTR from one process to another. Protected by lock.

5.58.3.44 void DataStaging::DTR::registerCallback (DTRCallback ∗ *cb*, StagingProcesses *owner*)

Register callback objects to be used during DTR processing.

Objects deriving from DTRCallback can be registered with this method. The callback method of these objects will then be called when the DTR is passed to the specified owner. Protected by lock.

5.58.3.45 void DataStaging::DTR::reset ()

Reset information held on this DTR, such as resolved replicas, error state etc.

Useful when a failed DTR is to be retried.

5.58.3.46 void DataStaging::DTR::reset_error_status ()

Set the error status back to NONE_ERROR and clear other fields.

5.58.3.47 void DataStaging::DTR::set_cache_file (const std::string & *filename*)

Set cache filename.

5.58.3.48 void DataStaging::DTR::set_cache_parameters (const CacheParameters & *param*) `[inline]`

Set cache parameters.

5.58.3.49 void DataStaging::DTR::set_cache_state (CacheState *state*)

Set the cache state.

5.58.3.50 void DataStaging::DTR::set_cancel_request ()

Set the DTR to be cancelled.

5.58.3.51 void DataStaging::DTR::set_error_status (DTRErrorStatus::DTRErrorStatusType *error_stat*, DTRErrorStatus::DTRErrorLocation *error_loc*, const std::string & *desc* = "")

Set the error status.

The DTRErrorStatus last error state field is set to the current status of the DTR. Protected by lock.

5.58.3.52 void DataStaging::DTR::set_force_registration (bool *force*) `[inline]`

Set force replication flag.

5.58.3.53 void DataStaging::DTR::set_mapped_source (const std::string & *file* = "") `[inline]`

Set the mapped file.

5.58.3.54 void DataStaging::DTR::set_priority (int *pri*)

Set the priority.

5.58.3.55 void DataStaging::DTR::set_process_time (const Arc::Period & *process_time*)

Set the next processing time to current time + given time.

5.58.3.56 void DataStaging::DTR::set_replication (bool *rep*) `[inline]`

Set replication flag.

5.58.3.57 void DataStaging::DTR::set_status (DTRStatus *stat*)

Set the status. Protected by lock.

5.58.3.58 void DataStaging::DTR::set_sub_share (const std::string & *share*) `[inline]`

Set sub-share.

5.58.3.59 void DataStaging::DTR::set_timeout (time_t *value*) `[inline]`

Set the timeout for processing this DTR.

5.58.3.60 void DataStaging::DTR::set_transfer_share (std::string *share_name*)

Set the transfer share. sub_share is automatically added to transfershare.

5.58.3.61 void DataStaging::DTR::set_tries_left (unsigned int *tries*)

Set the number of attempts remaining.

5.58.3.62 bool DataStaging::DTR::suspend ()

Suspend the DTR which is in doing transfer in the delivery process.

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

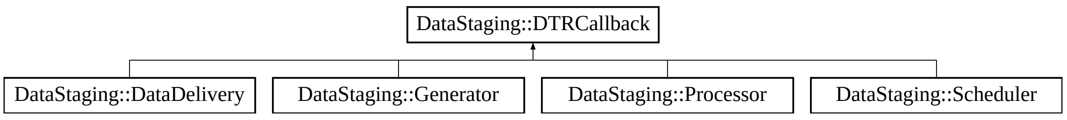
- DTR.h

5.59 DataStaging::DTRCallback Class Reference

The base class from which all callback-enabled classes should be derived.

`#include <DTR.h>`

Inheritance diagram for DataStaging::DTRCallback::



Public Member Functions

- virtual ~DTRCallback ()
- virtual void receiveDTR (DTR &dtr)=0

5.59.1 Detailed Description

The base class from which all callback-enabled classes should be derived.

This class is a container for a callback method which is called when a DTR is to be passed to a component. Several components in data staging (eg Scheduler, Generator) are subclasses of DTRCallback, which allows them to receive DTRs through the callback system.

5.59.2 Constructor & Destructor Documentation

5.59.2.1 virtual DataStaging::DTRCallback::~DTRCallback () `[inline, virtual]`

Empty virtual destructor

5.59.3 Member Function Documentation

5.59.3.1 virtual void DataStaging::DTRCallback::receiveDTR (DTR & *dtr*) `[pure virtual]`

Defines the callback method called when a DTR is pushed to this object. Note that the DTR object is passed by reference and so there is no guarantee that it will exist after this callback method is called.

Implemented in DataStaging::DataDelivery, DataStaging::Generator, DataStaging::Processor, and DataStaging::Scheduler.

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

- DTR.h

5.60 DataStaging::DTRErrorStatus Class Reference

A class to represent error states reported by various components.

`#include <DTRStatus.h>`

Public Types

- NONE_ERROR
- INTERNAL_ERROR
- SELF_REPLICATION_ERROR
- CACHE_ERROR
- TEMPORARY_REMOTE_ERROR
- PERMANENT_REMOTE_ERROR
- TRANSFER_SPEED_ERROR
- STAGING_TIMEOUT_ERROR
- NO_ERROR_LOCATION
- ERROR_SOURCE
- ERROR_DESTINATION
- ERROR_TRANSFER
- ERROR_UNKNOWN
- enum DTRErrorStatusType {

  NONE_ERROR, INTERNAL_ERROR, SELF_REPLICATION_ERROR, CACHE_ERROR,

  TEMPORARY_REMOTE_ERROR, PERMANENT_REMOTE_ERROR, TRANSFER_SPEED_-ERROR, STAGING_TIMEOUT_ERROR }
- enum DTRErrorLocation {

  NO_ERROR_LOCATION, ERROR_SOURCE, ERROR_DESTINATION, ERROR_TRANSFER,

  ERROR_UNKNOWN }

Public Member Functions

- DTRErrorStatus (DTRErrorStatusType status, DTRStatus::DTRStatusType error_state, DTRError-Location location, const std::string &desc="")
- DTRErrorStatus ()
- DTRErrorStatusType GetErrorStatus () const
- DTRStatus::DTRStatusType GetLastErrorState () const
- DTRErrorLocation GetErrorLocation () const
- std::string GetDesc () const
- bool operator== (const DTRErrorStatusType &s) const
- bool operator== (const DTRErrorStatus &s) const
- bool operator!= (const DTRErrorStatusType &s) const
- bool operator!= (const DTRErrorStatus &s) const
- DTRErrorStatus operator= (const DTRErrorStatusType &s)

5.60.1 Detailed Description

A class to represent error states reported by various components.

5.60.2 Member Enumeration Documentation

5.60.2.1 enum DataStaging::DTRErrorStatus::DTRErrorLocation

Describes where the error occurred.

Enumerator:

NO_ERROR_LOCATION	No error.
ERROR_SOURCE	Error with source.
ERROR_DESTINATION	Error with destination.
ERROR_TRANSFER	Error during transfer not directly related to source or destination.
ERROR_UNKNOWN	Error occurred in an unknown location.

5.60.2.2 enum DataStaging::DTRErrorStatus::DTRErrorStatusType

A list of error types.

Enumerator:

NONE_ERROR	No error.
INTERNAL_ERROR	Internal error in Data Staging logic.
SELF_REPLICATION_ERROR	Attempt to replicate a file to itself.
CACHE_ERROR	Permanent error with cache.
TEMPORARY_REMOTE_ERROR	Temporary error with remote service.
PERMANENT_REMOTE_ERROR	Permanent error with remote service.
TRANSFER_SPEED_ERROR	Transfer rate was too slow.
STAGING_TIMEOUT_ERROR	Waited for too long to become staging.

5.60.3 Constructor & Destructor Documentation

5.60.3.1 DataStaging::DTRErrorStatus::DTRErrorStatus (DTRErrorStatusType *status*, DTRStatus::DTRStatusType *error_state*, DTRErrorLocation *location*, const std::string & *desc* = `""`) `[inline]`

Create a new DTRErrorStatus with given error states.

5.60.3.2 DataStaging::DTRErrorStatus::DTRErrorStatus () `[inline]`

Create a new DTRErrorStatus with default none/null error states.

5.60.4 Member Function Documentation

5.60.4.1 std::string DataStaging::DTRErrorStatus::GetDesc () const `[inline]`

Returns the error description.

5.60.4.2 DTRErrorLocation DataStaging::DTRErrorStatus::GetErrorLocation () const `[inline]`

Returns the location at which the error occurred.

5.60.4.3 DTRErrorStatusType DataStaging::DTRErrorStatus::GetErrorStatus () const `[inline]`

Returns the error type.

5.60.4.4 DTRStatus::DTRStatusType DataStaging::DTRErrorStatus::GetLastErrorState () const `[inline]`

Returns the state in which the error occurred.

5.60.4.5 bool DataStaging::DTRErrorStatus::operator!= (const DTRErrorStatus & *s*) const `[inline]`

Returns true if this error status is not the same as the given DTRErrorStatus.

5.60.4.6 bool DataStaging::DTRErrorStatus::operator!= (const DTRErrorStatusType & *s*) const `[inline]`

Returns true if this error status is not the same as the given DTRErrorStatusType.

5.60.4.7 DTRErrorStatus DataStaging::DTRErrorStatus::operator= (const DTRErrorStatusType & *s*) `[inline]`

Make a new DTRErrorStatus with the same error status as the given DTRErrorStatusType.

5.60.4.8 bool DataStaging::DTRErrorStatus::operator== (const DTRErrorStatus & *s*) const `[inline]`

Returns true if this error status is the same as the given DTRErrorStatus.

5.60.4.9 bool DataStaging::DTRErrorStatus::operator== (const DTRErrorStatusType & *s*) const `[inline]`

Returns true if this error status is the same as the given DTRErrorStatusType.

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

- DTRStatus.h

5.61 DataStaging::DTRList Class Reference

Global list of all active DTRs in the system.

`#include <DTRList.h>`

Public Member Functions

- bool add_dtr (const DTR &DTRToAdd)
- bool delete_dtr (DTR ∗DTRToDelete)
- bool filter_dtrs_by_owner (StagingProcesses OwnerToFilter, std::list< DTR ∗ > &FilteredList)
- int number_of_dtrs_by_owner (StagingProcesses OwnerToFilter)
- bool filter_dtrs_by_status (DTRStatus StatusToFilter, std::list< DTR ∗ > &FilteredList)
- bool filter_dtrs_by_next_receiver (StagingProcesses NextReceiver, std::list< DTR ∗ > &Filtered-List)
- bool filter_pending_dtrs (std::list< DTR ∗ > &FilteredList)
- bool filter_dtrs_by_job (const std::string &jobid, std::list< DTR ∗ > &FilteredList)
- std::list< DTR ∗ > all_dtrs ()
- std::list< std::string > all_jobs ()
- void dumpState (const std::string &path)

5.61.1 Detailed Description

Global list of all active DTRs in the system.

This class contains several methods for filtering the list by owner, state etc

5.61.2 Member Function Documentation

5.61.2.1 bool DataStaging::DTRList::add_dtr (const DTR & *DTRToAdd*)

Put a new DTR into the list.

A (pointer to a) copy of the DTR is added to the list, and so DTRToAdd can be deleted after this method is called.

5.61.2.2 std::list<DTR∗> DataStaging::DTRList::all_dtrs ()

Get the list of all DTRs.

5.61.2.3 std::list<std::string> DataStaging::DTRList::all_jobs ()

Get the list of all job IDs.

5.61.2.4 bool DataStaging::DTRList::delete_dtr (DTR ∗ *DTRToDelete*)

Remove a DTR from the list.

The DTRToDelete object is destroyed, and hence should not be used after calling this method.

5.61.2.5 void DataStaging::DTRList::dumpState (const std::string & *path*)

Dump state of all current DTRs to a destination, eg file, database, url...

Currently only file is supported.

Parameters:

path Path to the file in which to dump state.

5.61.2.6 bool DataStaging::DTRList::filter_dtrs_by_job (const std::string & *jobid*, std::list< DTR $*$ > & *FilteredList*)

Get the list of DTRs corresponding to the given job ID.

Parameters:

FilteredList This list is filled with filtered DTRs

5.61.2.7 bool DataStaging::DTRList::filter_dtrs_by_next_receiver (StagingProcesses *NextReceiver*, std::list< DTR $*$ > & *FilteredList*)

Select DTRs that are about to go to the specified process.

This selection is actually a virtual queue for pre-, post-processor and delivery.

Parameters:

FilteredList This list is filled with filtered DTRs

5.61.2.8 bool DataStaging::DTRList::filter_dtrs_by_owner (StagingProcesses *OwnerToFilter*, std::list< DTR $*$ > & *FilteredList*)

Filter the queue to select DTRs owned by a specified process.

Parameters:

FilteredList This list is filled with filtered DTRs

5.61.2.9 bool DataStaging::DTRList::filter_dtrs_by_status (DTRStatus *StatusToFilter*, std::list< DTR $*$ > & *FilteredList*)

Filter the queue to select DTRs with particular status.

If we have only one common queue for all DTRs, this method is necessary to make virtual queues for the DTRs about to go into the pre-, post-processor or delivery stages.

Parameters:

FilteredList This list is filled with filtered DTRs

5.61.2.10 bool DataStaging::DTRList::filter_pending_dtrs (std::list< DTR * > & *FilteredList*)

Select DTRs that have just arrived from pre-, post-processor, delivery or generator.

These DTRs need some reaction from the scheduler. This selection is actually a virtual queue of DTRs that need to be processed.

Parameters:

FilteredList This list is filled with filtered DTRs

5.61.2.11 int DataStaging::DTRList::number_of_dtrs_by_owner (StagingProcesses *OwnerToFilter*)

Returns the number of DTRs owned by a particular process.

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

- DTRList.h

5.62 DataStaging::DTRStatus Class Reference

Class representing the status of a DTR.

```
#include <DTRStatus.h>
```

Public Types

- NEW
- CHECK_CACHE
- RESOLVE
- QUERY_REPLICA
- PRE_CLEAN
- STAGE_PREPARE
- TRANSFER_WAIT
- TRANSFER
- RELEASE_REQUEST
- REGISTER_REPLICA
- PROCESS_CACHE
- DONE
- CANCELLED
- CANCELLED_FINISHED
- ERROR
- CHECKING_CACHE
- CACHE_WAIT
- CACHE_CHECKED
- RESOLVING
- RESOLVED
- QUERYING_REPLICA
- REPLICA_QUERIED
- PRE_CLEANING
- PRE_CLEANED
- STAGING_PREPARING
- STAGING_PREPARING_WAIT
- STAGED_PREPARED
- TRANSFERRING
- TRANSFERRING_CANCEL
- TRANSFERRED
- RELEASING_REQUEST
- REQUEST_RELEASED
- REGISTERING_REPLICA
- REPLICA_REGISTERED
- PROCESSING_CACHE
- CACHE_PROCESSED
- NULL_STATE
- enum DTRStatusType {
  NEW, CHECK_CACHE, RESOLVE, QUERY_REPLICA,
  PRE_CLEAN, STAGE_PREPARE, TRANSFER_WAIT, TRANSFER,
  RELEASE_REQUEST, REGISTER_REPLICA, PROCESS_CACHE, DONE,

CANCELLED, CANCELLED_FINISHED, ERROR, CHECKING_CACHE,
CACHE_WAIT, CACHE_CHECKED, RESOLVING, RESOLVED,
QUERYING_REPLICA, REPLICA_QUERIED, PRE_CLEANING, PRE_CLEANED,
STAGING_PREPARING, STAGING_PREPARING_WAIT, STAGED_PREPARED, TRANSFERRING,
TRANSFERRING_CANCEL, TRANSFERRED, RELEASING_REQUEST, REQUEST_-RELEASED,
REGISTERING_REPLICA, REPLICA_REGISTERED, PROCESSING_CACHE, CACHE_-PROCESSED,
NULL_STATE }

Public Member Functions

- DTRStatus (const DTRStatusType &status, std::string desc="")
- DTRStatus ()
- bool operator== (const DTRStatusType &s) const
- bool operator== (const DTRStatus &s) const
- bool operator!= (const DTRStatusType &s) const
- bool operator!= (const DTRStatus &s) const
- DTRStatus operator= (const DTRStatusType &s)
- std::string str () const
- void SetDesc (const std::string &d)
- std::string GetDesc () const
- DTRStatusType GetStatus () const

5.62.1 Detailed Description

Class representing the status of a DTR.

5.62.2 Member Enumeration Documentation

5.62.2.1 enum DataStaging::DTRStatus::DTRStatusType

Possible state values.

Enumerator:

NEW	Just created.
CHECK_CACHE	Check the cache for the file may be already there.
RESOLVE	Resolve a meta-protocol.
QUERY_REPLICA	Query a replica.
PRE_CLEAN	The destination should be deleted.
STAGE_PREPARE	Prepare or stage the source and/or destination.
TRANSFER_WAIT	Hold the ready transfer.
TRANSFER	Transfer ready and can be started.
RELEASE_REQUEST	Transfer finished, release requests on the storage.

REGISTER_REPLICA Register a new replica of the destination.

PROCESS_CACHE Destination is cacheable, process cache.

DONE Everything completed successfully.

CANCELLED Cancellation request fulfilled successfully.

CANCELLED_FINISHED Cancellation request fulfilled but DTR also completed transfer successfully.

ERROR Error occured.

CHECKING_CACHE Checking the cache.

CACHE_WAIT Cache file is locked, waiting for its release.

CACHE_CHECKED Cache check completed.

RESOLVING Resolving replicas.

RESOLVED Replica resolution completed.

QUERYING_REPLICA Replica is being queried.

REPLICA_QUERIED Replica was queried.

PRE_CLEANING Deleting the destination.

PRE_CLEANED The destination file has been deleted.

STAGING_PREPARING Making a staging or preparing request.

STAGING_PREPARING_WAIT Wait for the status of the staging/preparing request.

STAGED_PREPARED Staging/preparing request completed.

TRANSFERRING Transfer is going.

TRANSFERRING_CANCEL Transfer is on-going but scheduled for cancellation.

TRANSFERRED Transfer completed.

RELEASING_REQUEST Releasing staging/preparing request.

REQUEST_RELEASED Release of staging/preparing request completed.

REGISTERING_REPLICA Registering a replica in an index service.

REPLICA_REGISTERED Replica registration completed.

PROCESSING_CACHE Releasing locks and copying/linking cache files to the session dir.

CACHE_PROCESSED Cache processing completed.

NULL_STATE "Stateless" DTR

5.62.3 Constructor & Destructor Documentation

5.62.3.1 DataStaging::DTRStatus::DTRStatus (const DTRStatusType & *status*, std::string *desc* = "") `[inline]`

Make new DTRStatus with given status.

5.62.3.2 DataStaging::DTRStatus::DTRStatus () `[inline]`

Make new DTRStatus with default NEW status.

5.62.4 Member Function Documentation

5.62.4.1 std::string DataStaging::DTRStatus::GetDesc () const `[inline]`

Get the detailed description of the current state.

5.62.4.2 DTRStatusType DataStaging::DTRStatus::GetStatus (void) const `[inline]`

Get the DTRStatusType of the current state.

5.62.4.3 bool DataStaging::DTRStatus::operator!= (const DTRStatus & *s*) const `[inline]`

Returns true if this status is not the same as the given DTRStatus.

5.62.4.4 bool DataStaging::DTRStatus::operator!= (const DTRStatusType & *s*) const `[inline]`

Returns true if this status is not the same as the given DTRStatusType.

5.62.4.5 DTRStatus DataStaging::DTRStatus::operator= (const DTRStatusType & *s*) `[inline]`

Make a new DTRStatus with the same status as the given DTRStatusType.

5.62.4.6 bool DataStaging::DTRStatus::operator== (const DTRStatus & *s*) const `[inline]`

Returns true if this status is the same as the given DTRStatus.

5.62.4.7 bool DataStaging::DTRStatus::operator== (const DTRStatusType & *s*) const `[inline]`

Returns true if this status is the same as the given DTRStatusType.

5.62.4.8 void DataStaging::DTRStatus::SetDesc (const std::string & *d*) `[inline]`

Set the detailed description of the current state.

5.62.4.9 std::string DataStaging::DTRStatus::str () const

Returns a string representation of the current state.

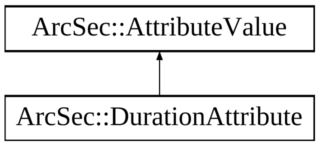
The documentation for this class was generated from the following file:

- DTRStatus.h

5.63 ArcSec::DurationAttribute Class Reference

`#include <DateTimeAttribute.h>`

Inheritance diagram for ArcSec::DurationAttribute::



Public Member Functions

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

5.63.1 Detailed Description

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

5.63.2 Member Function Documentation

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

encode the value in a string format

Implements ArcSec::AttributeValue.

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

Evluate whether "this" equale to the parameter value

Implements ArcSec::AttributeValue.

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

Get the AttributeId of the <Attribute>

Implements ArcSec::AttributeValue.

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

Get the DataType of the <Attribute>

Implements ArcSec::AttributeValue.

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

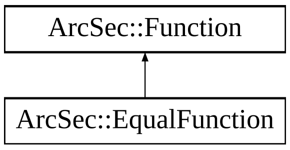
- DateTimeAttribute.h

5.64 ArcSec::EqualFunction Class Reference

Evaluate whether the two values are equal.

`#include <EqualFunction.h>`

Inheritance diagram for ArcSec::EqualFunction::



Public Member Functions

- virtual AttributeValue ∗ evaluate (AttributeValue ∗arg0, AttributeValue ∗arg1, bool check_id=true)
- virtual std::list< AttributeValue ∗ > evaluate (std::list< AttributeValue ∗ > args, bool check_-id=true)

Static Public Member Functions

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

5.64.1 Detailed Description

Evaluate whether the two values are equal.

5.64.2 Member Function Documentation

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

Evaluate a list of AttributeValue objects, and return a list of Attribute objects

Implements ArcSec::Function.

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

Evaluate two AttributeValue objects, and return one AttributeValue object

Implements ArcSec::Function.

5.64.2.3 static std::string ArcSec::EqualFunction::getFunctionName (std::string *datatype*) `[static]`

help function to get the FunctionName

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

- EqualFunction.h

5.65 ArcSec::EvalResult Struct Reference

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

```
#include <Result.h>
```

5.65.1 Detailed Description

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

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

- Result.h

5.66 ArcSec::EvaluationCtx Class Reference

EvaluationCtx, in charge of storing some context information for.

`#include <EvaluationCtx.h>`

Public Member Functions

- EvaluationCtx (Request ∗request)

5.66.1 Detailed Description

EvaluationCtx, in charge of storing some context information for.

5.66.2 Constructor & Destructor Documentation

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

Construct a new EvaluationCtx based on the given request

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

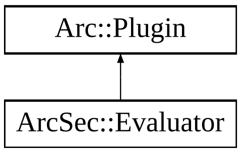
- EvaluationCtx.h

5.67 ArcSec::Evaluator Class Reference

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

`#include <Evaluator.h>`

Inheritance diagram for ArcSec::Evaluator::



Public Member Functions

- virtual Response ∗ evaluate (Request ∗request)=0
- virtual Response ∗ evaluate (const Source &request)=0
- virtual Response ∗ evaluate (Request ∗request, const Source &policy)=0
- virtual Response ∗ evaluate (const Source &request, const Source &policy)=0
- virtual Response ∗ evaluate (Request ∗request, Policy ∗policyobj)=0
- virtual Response ∗ evaluate (const Source &request, Policy ∗policyobj)=0
- virtual AttributeFactory ∗ getAttrFactory ()=0
- virtual FnFactory ∗ getFnFactory ()=0
- virtual AlgFactory ∗ getAlgFactory ()=0
- virtual void addPolicy (const Source &policy, const std::string &id="")=0
- virtual void addPolicy (Policy ∗policy, const std::string &id="")=0
- virtual void setCombiningAlg (EvaluatorCombiningAlg alg)=0
- virtual void setCombiningAlg (CombiningAlg ∗alg=NULL)=0
- virtual const char ∗ getName (void) const =0

Protected Member Functions

- virtual Response ∗ evaluate (EvaluationCtx ∗ctx)=0

5.67.1 Detailed Description

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

5.67.2 Member Function Documentation

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

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

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

Add policy from specified source to the evaluator. Policy will be marked with id.

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

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

5.67.2.4 virtual Response∗ ArcSec::Evaluator::evaluate (const Source & *request*, Policy ∗ *policyobj*) `[pure virtual]`

Evaluate the request from specified source against the specified policy. In some implementations all of the existing policie inside the evaluator may be destroyed by this method.

5.67.2.5 virtual Response∗ ArcSec::Evaluator::evaluate (Request ∗ *request*, Policy ∗ *policyobj*) `[pure virtual]`

Evaluate the specified request against the specified policy. In some implementations all of the existing policy inside the evaluator may be destroyed by this method.

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

Evaluate the request from specified source against the policy from specified source. In some implementations all of the existing policie inside the evaluator may be destroyed by this method.

5.67.2.7 virtual Response∗ ArcSec::Evaluator::evaluate (Request ∗ *request*, const Source & *policy*) `[pure virtual]`

Evaluate the specified request against the policy from specified source. In some implementations all of the existing policies inside the evaluator may be destroyed by this method.

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

Evaluates the request by using a specified source

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

Evaluates the request by using a Request object. Evaluation is done till at least one of policies is satisfied.

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

Get the AlgFactory object

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

Get the AttributeFactory object

5.67.2.12 virtual FnFactory∗ ArcSec::Evaluator::getFnFactory () [pure virtual]

Get the FnFactory object

5.67.2.13 virtual const char∗ ArcSec::Evaluator::getName (void) const [pure virtual]

Get the name of this evaluator

5.67.2.14 virtual void ArcSec::Evaluator::setCombiningAlg (CombiningAlg ∗ *alg* = NULL) [pure virtual]

Specifies loadable combining algorithms. In case of multiple policies their results will be combined using this algorithm. To switch to simple algorithm specify NULL argument.

5.67.2.15 virtual void ArcSec::Evaluator::setCombiningAlg (EvaluatorCombiningAlg *alg*) [pure virtual]

Specifies one of simple combining algorithms. In case of multiple policies their results will be combined using this algorithm.

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

- Evaluator.h

5.68 ArcSec::EvaluatorContext Class Reference

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

`#include <Evaluator.h>`

Public Member Functions

- operator AttributeFactory ∗ ()
- operator FnFactory ∗ ()
- operator AlgFactory ∗ ()

5.68.1 Detailed Description

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

5.68.2 Member Function Documentation

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

Returns associated AlgFactory object

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

Returns associated AttributeFactory object

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

Returns associated FnFactory object

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

- Evaluator.h

5.69 ArcSec::EvaluatorLoader Class Reference

EvaluatorLoader is implemented as a helper class for loading different Evaluator objects, like ArcEvaluator.

`#include <EvaluatorLoader.h>`

Public Member Functions

- Evaluator ∗ getEvaluator (const std::string &classname)
- Evaluator ∗ getEvaluator (const Policy ∗policy)
- Evaluator ∗ getEvaluator (const Request ∗request)
- Request ∗ getRequest (const std::string &classname, const Source &requestsource)
- Request ∗ getRequest (const Source &requestsource)
- Policy ∗ getPolicy (const std::string &classname, const Source &policysource)
- Policy ∗ getPolicy (const Source &policysource)

5.69.1 Detailed Description

EvaluatorLoader is implemented as a helper class for loading different Evaluator objects, like ArcEvaluator.

The object loading is based on the configuration information about evaluator, including information for factory class, request, policy and evaluator itself

5.69.2 Member Function Documentation

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

Get evaluator object suitable for presented request

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

Get evaluator object suitable for presented policy

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

Get evaluator object according to the class name

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

Get proper policy object according to the policy source

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

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

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

Get request object according to the request source

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

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

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

- EvaluatorLoader.h

5.70 Arc::ExecutionTarget Class Reference

ExecutionTarget.

```
#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

5.70.1 Detailed Description

ExecutionTarget.

This class describe a target which accept computing jobs. All of the members contained in this class, with a few exceptions, are directly linked to attributes defined in the GLUE Specification v. 2.0 (GFD-R-P.147).

5.70.2 Constructor & Destructor Documentation

5.70.2.1 Arc::ExecutionTarget::ExecutionTarget ()

Create an ExecutionTarget.

Default constructor to create an ExecutionTarget. Takes no arguments.

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

Create an ExecutionTarget.

Copy constructor.

Parameters:

target ExecutionTarget to copy.

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

Create an ExecutionTarget.

Copy constructor? Needed from Python?

Parameters:

addrptr

5.70.3 Member Function Documentation

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

Get Submitter to the computing resource represented by the ExecutionTarget.

Method which returns a specialized Submitter which can be used for submitting jobs to the computing resource represented by the ExecutionTarget. In order to return the correct specialized Submitter the Grid-Flavour variable must be correctly set.

Parameters:

ucfg UserConfig object with paths to user credentials etc.

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

Create an ExecutionTarget.

Assignment operator

Parameters:

target is ExecutionTarget to copy.

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

DEPRECATED: Print the ExecutionTarget information to std::cout.

This method is deprecated, use the SaveToStream method instead. Method to print the ExecutionTarget attributes to std::cout

Parameters:

longlist is true for long list printing.

See also:

SaveToStream

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

Print the ExecutionTarget information to a std::ostream object.

Method to print the ExecutionTarget attributes to a std::ostream object.

Parameters:

out is the std::ostream to print the attributes to.

longlist should be set to true for printing a long list.

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

Update ExecutionTarget after succesful job submission.

Method to update the ExecutionTarget after a job succesfully has been submitted to the computing resource it represents. E.g. if a job is sent to the computing resource and is expected to enter the queue, then the WaitingJobs attribute is incremented with 1.

Parameters:

jobdesc contains all information about the job submitted.

5.70.4 Field Documentation

5.70.4.1 std::list<ApplicationEnvironment> Arc::ExecutionTarget::ApplicationEnvironments

ApplicationEnvironments.

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

5.70.4.2 std::string Arc::ExecutionTarget::ComputingShareName

ComputingShareName String 0..1.

Human-readable name. This variable represents the ComputingShare.Name attribute of GLUE2.

5.70.4.3 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) 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).

5.70.4.4 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.

5.70.4.5 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.

5.70.4.6 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.

5.70.4.7 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

5.71 Arc::ExpirationReminder Class Reference

A class intended for internal use within counters.

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

Public Member Functions

- bool operator< (const ExpirationReminder &other) const
- Glib::TimeVal getExpiryTime () const
- Counter::IDType getReservationID () const

Friends

- class Counter

5.71.1 Detailed Description

A class intended for internal use within counters.

This class is used for "reminder objects" that are used for automatic deallocation of self-expiring reservations.

5.71.2 Member Function Documentation

5.71.2.1 Glib::TimeVal Arc::ExpirationReminder::getExpiryTime () const

Returns the expiry time.

This method returns the expiry time of the reservation that this ExpirationReminder is associated with.

Returns:

The expiry time.

5.71.2.2 Counter::IDType Arc::ExpirationReminder::getReservationID () const

Returns the identification number of the reservation.

This method returns the identification number of the self-expiring reservation that this ExpirationReminder is associated with.

Returns:

The identification number.

5.71.2.3 bool Arc::ExpirationReminder::operator< (const ExpirationReminder & *other*) const

Less than operator, compares "soonness".

This is the less than operator for the ExpirationReminder class. It compares the priority of such objects with respect to which reservation expires first. It is used when reminder objects are inserted in a priority queue in order to allways place the next reservation to expire at the top.

5.71.3 Friends And Related Function Documentation

5.71.3.1 friend class Counter `[friend]`

The Counter class needs to be a friend.

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

- Counter.h

5.72 Arc::FileAccess Class Reference

Defines interface for accessing filesystems.

```
#include <FileAccess.h>
```

Public Member Functions

- bool ping (void)
- bool setuid (int uid, int gid)
- bool mkdir (const std::string &path, mode_t mode)
- bool mkdirp (const std::string &path, mode_t mode)
- bool link (const std::string &oldpath, const std::string &newpath)
- bool softlink (const std::string &oldpath, const std::string &newpath)
- bool copy (const std::string &oldpath, const std::string &newpath, mode_t mode)
- bool chmod (const std::string &path, mode_t mode)
- bool stat (const std::string &path, struct stat &st)
- bool lstat (const std::string &path, struct stat &st)
- bool fstat (struct stat &st)
- bool ftruncate (off_t length)
- off_t fallocate (off_t length)
- bool readlink (const std::string &path, std::string &linkpath)
- bool remove (const std::string &path)
- bool unlink (const std::string &path)
- bool rmdir (const std::string &path)
- bool rmdirr (const std::string &path)
- bool opendir (const std::string &path)
- bool closedir (void)
- bool readdir (std::string &name)
- bool open (const std::string &path, int flags, mode_t mode)
- bool close (void)
- bool mkstemp (std::string &path, mode_t mode)
- off_t lseek (off_t offset, int whence)
- ssize_t read (void ∗buf, size_t size)
- ssize_t write (const void ∗buf, size_t size)
- ssize_t pread (void ∗buf, size_t size, off_t offset)
- ssize_t pwrite (const void ∗buf, size_t size, off_t offset)
- int geterrno ()
- operator bool (void)
- bool operator! (void)

Static Public Member Functions

- static void testtune (void)

Data Structures

- struct **header_t**

5.72.1 Detailed Description

Defines interface for accessing filesystems.

This class accesses local filesystem through proxy executable which allows to switch user id in multi-threaded systems without introducing conflict with other threads. Its methods are mostly replicas of corresponding POSIX functions with some convenience tweaking.

5.72.2 Member Function Documentation

5.72.2.1 bool Arc::FileAccess::chmod (const std::string & *path*, mode_t *mode*)

Change mode of filesystem object.

5.72.2.2 bool Arc::FileAccess::close (void)

Close open file.

5.72.2.3 bool Arc::FileAccess::closedir (void)

Close open directory.

5.72.2.4 bool Arc::FileAccess::copy (const std::string & *oldpath*, const std::string & *newpath*, mode_t *mode*)

Copy file to new location. If new file is created it is assigned secified mode.

5.72.2.5 off_t Arc::FileAccess::fallocate (off_t *length*)

Allocate disk space for open file.

5.72.2.6 bool Arc::FileAccess::fstat (struct stat & *st*)

stat open file.

5.72.2.7 bool Arc::FileAccess::ftruncate (off_t *length*)

Truncate open file.

5.72.2.8 int Arc::FileAccess::geterrno () `[inline]`

Get errno of last operation. Every operation resets errno.

5.72.2.9 bool Arc::FileAccess::link (const std::string & *oldpath*, const std::string & *newpath*)

Create hard link.

5.72.2.10 off_t Arc::FileAccess::lseek (off_t *offset*, int *whence*)

Change current position in open file.

5.72.2.11 bool Arc::FileAccess::lstat (const std::string & *path*, struct stat & *st*)

stat symbolic link or file.

5.72.2.12 bool Arc::FileAccess::mkdir (const std::string & *path*, mode_t *mode*)

Make a directory and assign it specified mode.

5.72.2.13 bool Arc::FileAccess::mkdirp (const std::string & *path*, mode_t *mode*)

Make a directory and assign it specified mode. If missing all intermediate directories are created too.

5.72.2.14 bool Arc::FileAccess::mkstemp (std::string & *path*, mode_t *mode*)

Open new temporary file for writing. On input path contains template of file name ending with XXXXXX. On output path is path to created file.

5.72.2.15 bool Arc::FileAccess::open (const std::string & *path*, int *flags*, mode_t *mode*)

Open file. Only one file may be open at a time.

5.72.2.16 bool Arc::FileAccess::opendir (const std::string & *path*)

Open directory. Only one directory may be open at a time.

5.72.2.17 Arc::FileAccess::operator bool (void) `[inline]`

Returns true if this instance is in useful condition.

5.72.2.18 bool Arc::FileAccess::operator! (void) `[inline]`

Returns true if this instance is not in useful condition.

5.72.2.19 bool Arc::FileAccess::ping (void)

Check if communication with proxy works.

5.72.2.20 ssize_t Arc::FileAccess::pread (void * *buf*, size_t *size*, off_t *offset*)

Read from open file at specified offset.

5.72.2.21 ssize_t Arc::FileAccess::pwrite (const void ∗ *buf*, size_t *size*, off_t *offset*)

Write to open file at specified offset.

5.72.2.22 ssize_t Arc::FileAccess::read (void ∗ *buf*, size_t *size*)

Read from open file.

5.72.2.23 bool Arc::FileAccess::readdir (std::string & *name*)

Read relative name of object in open directory.

5.72.2.24 bool Arc::FileAccess::readlink (const std::string & *path*, std::string & *linkpath*)

Read content of symbolic link.

5.72.2.25 bool Arc::FileAccess::remove (const std::string & *path*)

Remove file system object.

5.72.2.26 bool Arc::FileAccess::rmdir (const std::string & *path*)

Remove directory (if empty).

5.72.2.27 bool Arc::FileAccess::rmdirr (const std::string & *path*)

Remove directory recursively.

5.72.2.28 bool Arc::FileAccess::setuid (int *uid*, int *gid*)

Modify user uid and gid. If any is set to 0 then executable is switched to original uid/gid.

5.72.2.29 bool Arc::FileAccess::softlink (const std::string & *oldpath*, const std::string & *newpath*)

Create symbolic (aka soft) link.

5.72.2.30 bool Arc::FileAccess::stat (const std::string & *path*, struct stat & *st*)

stat file.

5.72.2.31 static void Arc::FileAccess::testtune (void) `[static]`

Special method for using in unit tests.

5.72.2.32 bool Arc::FileAccess::unlink (const std::string & *path*)

Remove file.

5.72.2.33 ssize_t Arc::FileAccess::write (const void $*$ *buf*, size_t *size*)

Write to open file.

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

- FileAccess.h

5.73 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)

5.73.1 Detailed Description

FileCache 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() should be called, so that the cache file can be prepared and locked. When a transfer has finished successfully, Link() 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() 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 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 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 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 class, which creates a lock file with the '.lock' suffix next to the cache file. Calling Start() creates this lock and Stop() releases it. All processes calling Start() must wait until they successfully obtain the lock before downloading can begin or an existing cache file can be used. Once a process obtains a lock it must later release it by calling Stop() or StopAnd-Delete(). Once a cache file is successfully linked to the per-job directory in Link(), it is also unlocked, but Stop() should still be called after.

5.73.2 Constructor & Destructor Documentation

5.73.2.1 Arc::FileCache::FileCache (std::string *cache_path*, std::string *id*, uid_t *job_uid*, gid_t *job_gid*)

Create a new FileCache instance.

Parameters:

cache_path The format is "cache_dir[link_path]". path is the path to the cache directory and the optional link_path is used to create a link in case the cache directory is visible under a different name during actual usage. When linking from the session dir this path is used instead of cache_-path.

id the job id. This is used to create the per-job dir which the job's cache files will be hard linked from

job_uid owner of job. The per-job dir will only be readable by this user

job_gid owner group of job

5.73.2.2 Arc::FileCache::FileCache (std::vector< std::string > *caches*, std::string *id*, uid_t *job_uid*, gid_t *job_gid*)

Create a new FileCache instance with multiple cache dirs

Parameters:

caches a vector of strings describing caches. The format of each string is "cache_dir[link_path]".

id the job id. This is used to create the per-job dir which the job's cache files will be hard linked from

job_uid owner of job. The per-job dir will only be readable by this user

job_gid owner group of job

5.73.2.3 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 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

5.73.2.4 Arc::FileCache::FileCache () `[inline]`

Default constructor. Invalid cache.

5.73.3 Member Function Documentation

5.73.3.1 bool Arc::FileCache::AddDN (std::string *url*, std::string *DN*, Time *expiry_time*)

Add the given DN to the list of cached DNs with the given expiry time

Parameters:

url the url corresponding to the cache file to which we want to add a cached DN

DN the DN of the user

expiry_time the expiry time of this DN in the DN cache

5.73.3.2 bool Arc::FileCache::CheckCreated (std::string *url*)

Check if there is an information about creation time. Returns true if the file exists in the cache, since the creation time is the creation time of the cache file.

Parameters:

url the url corresponding to the cache file for which we want to know if the creation date exists

5.73.3.3 bool Arc::FileCache::CheckDN (std::string *url*, std::string *DN*)

Check if the given DN is cached for authorisation.

Parameters:

url the url corresponding to the cache file for which we want to check the cached DN

DN the DN of the user

5.73.3.4 bool Arc::FileCache::CheckValid (std::string *url*)

Check if there is an information about expiry time.

Parameters:

url the url corresponding to the cache file for which we want to know if the expiration time exists

5.73.3.5 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() should be used instead with copy set to true.

5.73.3.6 std::string Arc::FileCache::File (std::string *url*)

Returns the full pathname of the file in the cache which corresponds to the given url.

5.73.3.7 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

5.73.3.8 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

5.73.3.9 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().

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

5.73.3.10 Arc::FileCache::operator bool (void) `[inline]`

Returns true if object is useable.

5.73.3.11 bool Arc::FileCache::operator== (const FileCache & *a*)

Return true if all attributes are equal

5.73.3.12 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.

5.73.3.13 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

5.73.3.14 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.

5.73.3.15 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() 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()), or if it fails to delete the lock file.

Parameters:

url the url of the file that was downloaded

5.73.3.16 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().

Parameters:

url the url corresponding to the cache file that has to be released and deleted

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

- FileCache.h

5.74 FileCacheHash Class Reference

`#include <FileCacheHash.h>`

Static Public Member Functions

- static std::string getHash (std::string url)
- static int maxLength ()

5.74.1 Detailed Description

FileCacheHash provides methods to make hashes from strings. Currently the md5 hash from the openssl library is used.

5.74.2 Member Function Documentation

5.74.2.1 static std::string FileCacheHash::getHash (std::string *url*) `[static]`

Return a hash of the given URL, according to the current hash scheme.

5.74.2.2 static int FileCacheHash::maxLength () `[inline, static]`

Return the maximum length of a hash string.

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

- FileCacheHash.h

5.75 Arc::FileInfo Class Reference

FileInfo stores information about files (metadata).

```
#include <FileInfo.h>
```

5.75.1 Detailed Description

FileInfo stores information about files (metadata).

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

- FileInfo.h

5.76 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

5.76.1 Detailed Description

A general file locking class.

This class can be used when protected access is required to files which are used by multiple processes or threads. Call acquire() to obtain a lock and release() to release it when finished. check() can be used to verify if a lock is valid for the current process. Locks are independent of FileLock objects - locks are only created and destroyed through acquire() and release(), not on creation or destruction of FileLock objects.

Unless use_pid is set false, the process ID and hostname of the calling process are stored in a file filename.lock in the form pid. This information is used to determine whether a lock is still valid. It is also possible to specify a timeout on the lock.

To ensure an atomic locking operation, acquire() first creates a temporary lock file filename.lock.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.

5.76.2 Constructor & Destructor Documentation

5.76.2.1 Arc::FileLock::FileLock (const std::string & *filename*, unsigned int *timeout* = `DEFAULT_LOCK_TIMEOUT`, bool *use_pid* = `true`)

Create a new FileLock object.

Parameters:

filename The name of the file to be locked *timeout* The timeout of the lock

use_pid If true, use process id in the lock and to determine lock validity

5.76.3 Member Function Documentation

5.76.3.1 bool Arc::FileLock::acquire ()

Acquire the lock.

Callers can use this version of acquire() if they do not care whether an invalid lock was removed in the process of obtaining the lock.

5.76.3.2 bool Arc::FileLock::acquire (bool & *lock_removed*)

Acquire the lock.

Returns true if the lock was acquired successfully. Locks are acquired if no lock file currently exists, or if the current lock file is invalid. A lock is invalid if the process ID inside the lock no longer exists on the host inside the lock, or the age of the lock file is greater than the lock timeout.

Parameters:

lock_removed Set to true if an existing lock was removed due to being invalid. In this case the caller may decide to check or delete the file as it is potentially corrupted.

Returns:

True if lock is successfully acquired

5.76.3.3 bool Arc::FileLock::check ()

Check the lock is valid.

Returns true if the lock is valid for the current process

5.76.3.4 static std::string Arc::FileLock::getLockSuffix () `[static]`

Get the lock suffix used.

5.76.3.5 bool Arc::FileLock::release (bool *force* = `false`)

Release the lock.

Parameters:

force Remove the lock without checking ownership or timeout

5.76.4 Field Documentation

5.76.4.1 const int Arc::FileLock::DEFAULT_LOCK_TIMEOUT `[static]`

Default timeout for a lock.

5.76.4.2 const std::string Arc::FileLock::LOCK_SUFFIX `[static]`

Suffix added to file name to make lock file.

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

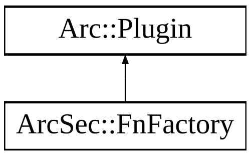
- FileLock.h

5.77 ArcSec::FnFactory Class Reference

Interface for function factory class.

`#include <FnFactory.h>`

Inheritance diagram for ArcSec::FnFactory::



Public Member Functions

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

5.77.1 Detailed Description

Interface for function factory class.

FnFactory is in charge of creating Function object according to the algorithm type given as argument of method createFn. This class can be inherited for implementing a factory class which can create some specific Function objects.

5.77.2 Member Function Documentation

5.77.2.1 virtual Function∗ ArcSec::FnFactory::createFn (const std::string & *type*) `[pure virtual]`

creat algorithm object based on the type algorithm type

Parameters:

type The type of Function

Returns:

The object of Function

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

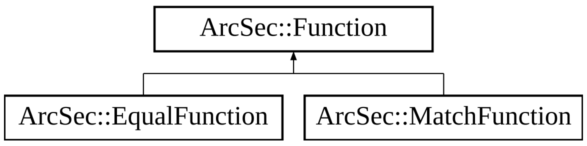
- FnFactory.h

5.78 ArcSec::Function Class Reference

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

`#include <Function.h>`

Inheritance diagram for ArcSec::Function::



Public Member Functions

- virtual AttributeValue ∗ evaluate (AttributeValue ∗arg0, AttributeValue ∗arg1, bool check_-id=true)=0
- virtual std::list< AttributeValue ∗ > evaluate (std::list< AttributeValue ∗ > args, bool check_-id=true)=0

5.78.1 Detailed Description

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

5.78.2 Member Function Documentation

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

Evaluate a list of AttributeValue objects, and return a list of Attribute objects

Implemented in ArcSec::EqualFunction, and ArcSec::MatchFunction.

5.78.2.2 virtual AttributeValue∗ ArcSec::Function::evaluate (AttributeValue ∗ *arg0*, AttributeValue ∗ *arg1*, bool *check_id* = `true`) `[pure virtual]`

Evaluate two AttributeValue objects, and return one AttributeValue object

Implemented in ArcSec::EqualFunction, and ArcSec::MatchFunction.

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

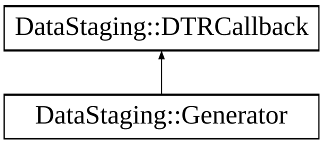
- Function.h

5.79 DataStaging::Generator Class Reference

Simple Generator implementation.

```
#include <Generator.h>
```

Inheritance diagram for DataStaging::Generator::



Public Member Functions

- virtual void receiveDTR (DTR &dtr)
- void run (const std::string &source, const std::string &destination)

5.79.1 Detailed Description

Simple Generator implementation.

This Generator implementation is included in the data staging library for for basic direct testing of the library and to show how a Generator can be written. It has one method, run(), which creates a single DTR and submits it to the Scheduler.

5.79.2 Member Function Documentation

5.79.2.1 virtual void DataStaging::Generator::receiveDTR (DTR & *dtr*) `[virtual]`

Implementation of callback from DTRCallback.

Callback method used when DTR processing is complete to pass back to the generator. The DTR is passed by value so that the scheduler can delete its copy of the object after calling this method.

Implements DataStaging::DTRCallback.

5.79.2.2 void DataStaging::Generator::run (const std::string & *source*, const std::string & *destination*)

Submit a DTR with given source and destination.

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

- Generator.h

5.80 Arc::GLUE2 Class Reference

GLUE2 parser.

```
#include <GLUE2.h>
```

5.80.1 Detailed Description

GLUE2 parser.

This class pparses GLUE2 infromation rendeed in XML and transfers information into various classes representing different types of objects which GLUE2 information model can describe. This parser uses GLUE Specification v. 2.0 (GFD-R-P.147).

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

- GLUE2.h

5.81 Arc::InfoCache Class Reference

Stores XML document in filesystem split into parts.

```
#include <InfoCache.h>
```

Public Member Functions

- InfoCache (const Config &cfg, const std::string &service_id)

5.81.1 Detailed Description

Stores XML document in filesystem split into parts.

5.81.2 Constructor & Destructor Documentation

5.81.2.1 Arc::InfoCache::InfoCache (const Config & *cfg*, const std::string & *service_id*)

Creates object according to configuration (see InfoCacheConfig.xsd).

XML configuration is passed in cfg. Argument service_id is used to distiguish between various documents stored under same path - corresponding files will be stored in subdirectory with service_id name.

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

- InfoCache.h

5.82 Arc::InfoFilter Class Reference

Filters information document according to identity of requestor.

```
#include <InfoFilter.h>
```

Public Member Functions

- InfoFilter (MessageAuth &id)
- bool Filter (XMLNode doc) const
- bool Filter (XMLNode doc, const InfoFilterPolicies &policies, const NS &ns) const

5.82.1 Detailed Description

Filters information document according to identity of requestor.

Identity is compared to policies stored inside information document and external ones. Parts of document which do not pass policy evaluation are removed.

5.82.2 Constructor & Destructor Documentation

5.82.2.1 Arc::InfoFilter::InfoFilter (MessageAuth & *id*)

Creates object and associates identity.

Associated identity is not copied, hence passed argument must not be destroyed while this method is used.

5.82.3 Member Function Documentation

5.82.3.1 bool Arc::InfoFilter::Filter (XMLNode *doc*, const InfoFilterPolicies & *policies*, const NS & *ns*) const

Filter information document according to internal and external policies.

In provided document all policies and nodes which have their policies evaluated to negative result are removed. External policies are provided in policies argument. First element of every pair is XPath defining to which XML node policy must be applied. Second element is policy itself. Argument ns defines XML namespaces for XPath evaluation.

5.82.3.2 bool Arc::InfoFilter::Filter (XMLNode *doc*) const

Filter information document according to internal policies.

In provided document all policies and nodes which have their policies evaluated to negative result are removed.

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

- InfoFilter.h

5.83 Arc::InfoRegister Class Reference

Registration to ISIS interface.

```
#include <InfoRegister.h>
```

5.83.1 Detailed Description

Registration to ISIS interface.

This class represents service registering to Information Indexing Service. It does not perform registration itself. It only collects configuration information. Configuration is as described in InfoRegisterConfig.xsd for element InfoRegistration.

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

- InfoRegister.h

5.84 Arc::InfoRegisterContainer Class Reference

```
#include <InfoRegister.h>
```

Public Member Functions

- InfoRegistrar ∗ addRegistrar (XMLNode doc)
- void addService (InfoRegister ∗reg, const std::list< std::string > &ids, XMLNode cfg=XMLNode())
- void removeService (InfoRegister ∗reg)

5.84.1 Detailed Description

Singleton class for scanning configuration and storing refernces to registration elements.

5.84.2 Member Function Documentation

5.84.2.1 InfoRegistrar∗ Arc::InfoRegisterContainer::addRegistrar (XMLNode *doc*)

Adds ISISes to list of handled services.

Supplied configuration document is scanned for InfoRegistrar elements and those are turned into InfoRegistrar classes for handling connection to ISIS service each.

5.84.2.2 void Arc::InfoRegisterContainer::addService (InfoRegister ∗ *reg*, const std::list< std::string > & *ids*, XMLNode *cfg* = `XMLNode()`)

Adds service to list of handled.

This method must be called first time after last addRegistrar was called - services will be only associated with ISISes which are already added. Argument ids contains list of ISIS identifiers to which service is associated. If ids is empty then service is associated to all ISISes currently added. If argument cfg is available and no ISISes are configured then addRegistrars is called with cfg used as configuration document.

5.84.2.3 void Arc::InfoRegisterContainer::removeService (InfoRegister ∗ *reg*)

This method must be called if service being destroyed.

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

- InfoRegister.h

5.85 Arc::InfoRegisters Class Reference

Handling multiple registrations to ISISes.

```
#include <InfoRegister.h>
```

Public Member Functions

- InfoRegisters (XMLNode &cfg, Service ∗service_)

5.85.1 Detailed Description

Handling multiple registrations to ISISes.

5.85.2 Constructor & Destructor Documentation

5.85.2.1 Arc::InfoRegisters::InfoRegisters (XMLNode & *cfg*, Service ∗ *service_*)

Constructor creates InfoRegister objects according to configuration.

Inside cfg elements InfoRegistration are found and for each corresponding InfoRegister object is created. Those objects are destroyed in destructor of this class.

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

- InfoRegister.h

5.86 Arc::InfoRegistrar Class Reference

Registration process associated with particular ISIS.

```
#include <InfoRegister.h>
```

Public Member Functions

- void registration (void)
- bool addService (InfoRegister ∗, XMLNode &)
- bool removeService (InfoRegister ∗)

5.86.1 Detailed Description

Registration process associated with particular ISIS.

Instance of this class starts thread which takes care passing information about associated services to ISIS service defined in configuration. Configuration is as described in InfoRegister.xsd for element InfoRegistrar.

5.86.2 Member Function Documentation

5.86.2.1 bool Arc::InfoRegistrar::addService (InfoRegister ∗, XMLNode &)

Adds new service to list of handled services.

Service is described by it's InfoRegister object which must be valid as long as this object is functional.

5.86.2.2 void Arc::InfoRegistrar::registration (void)

Performs registartion in a loop.

Never exits unless there is a critical error or requested by destructor.

5.86.2.3 bool Arc::InfoRegistrar::removeService (InfoRegister ∗)

Removes service from list of handled services.

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

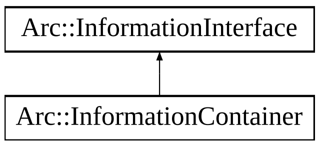
- InfoRegister.h

5.87 Arc::InformationContainer Class Reference

Information System document container and processor.

`#include <InformationInterface.h>`

Inheritance diagram for Arc::InformationContainer::



Public Member Functions

- InformationContainer (XMLNode doc, bool copy=false)
- XMLNode Acquire (void)
- void Assign (XMLNode doc, bool copy=false)

Protected Member Functions

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

Protected Attributes

- XMLNode doc_

5.87.1 Detailed Description

Information System document container and processor.

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

5.87.2 Constructor & Destructor Documentation

5.87.2.1 Arc::InformationContainer::InformationContainer (XMLNode *doc*, bool *copy* = `false`)

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

5.87.3 Member Function Documentation

5.87.3.1 XMLNode Arc::InformationContainer::Acquire (void)

Get a lock on contained XML document. To be used in multi-threaded environment. Do not forget to release it with Release()

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

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

5.87.3.3 virtual void Arc::InformationContainer::Get (const std::list< std::string > & *path*, XMLNodeContainer & *result*) `[protected, virtual]`

This method is called by this object's Process method. Real implementation of this class should return (sub)tree of XML document. This method may be called multiple times per single Process call. Here is a set on XML element names specifying how to reach requested node(s).

Reimplemented from Arc::InformationInterface.

5.87.4 Field Documentation

5.87.4.1 XMLNode Arc::InformationContainer::doc_ `[protected]`

Either link or container of XML document

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

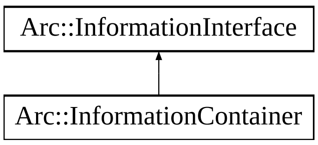
- InformationInterface.h

5.88 Arc::InformationInterface Class Reference

Information System message processor.

`#include <InformationInterface.h>`

Inheritance diagram for Arc::InformationInterface::



Public Member Functions

- InformationInterface (bool safe=true)

Protected Member Functions

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

Protected Attributes

- Glib::Mutex lock_

5.88.1 Detailed Description

Information System message processor.

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

5.88.2 Constructor & Destructor Documentation

5.88.2.1 Arc::InformationInterface::InformationInterface (bool *safe* = `true`)

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

5.88.3 Member Function Documentation

5.88.3.1 virtual void Arc::InformationInterface::Get (const std::list< std::string > & *path*, XMLNodeContainer & *result*) `[protected, virtual]`

This method is called by this object's Process method. Real implementation of this class should return (sub)tree of XML document. This method may be called multiple times per single Process call. Here is a set on XML element names specifying how to reach requested node(s).

Reimplemented in Arc::InformationContainer.

5.88.4 Field Documentation

5.88.4.1 Glib::Mutex Arc::InformationInterface::lock_ `[protected]`

Mutex used to protect access to Get methods in multi-threaded env.

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

- InformationInterface.h

5.89 Arc::InformationRequest Class Reference

Request for information in InfoSystem.

```
#include <InformationInterface.h>
```

Public Member Functions

- InformationRequest (void)
- InformationRequest (const std::list< std::string > &path)
- InformationRequest (const std::list< std::list< std::string > > &paths)
- InformationRequest (XMLNode query)
- SOAPEnvelope ∗ SOAP (void)

5.89.1 Detailed Description

Request for information in InfoSystem.

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

5.89.2 Constructor & Destructor Documentation

5.89.2.1 Arc::InformationRequest::InformationRequest (void)

Dummy constructor

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

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

5.89.2.3 Arc::InformationRequest::InformationRequest (const std::list< std::list< std::string > > & *paths*)

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

5.89.2.4 Arc::InformationRequest::InformationRequest (XMLNode *query*)

Request for attributes specified by XPath query.

5.89.3 Member Function Documentation

5.89.3.1 SOAPEnvelope∗ Arc::InformationRequest::SOAP (void)

Returns generated SOAP message

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

- InformationInterface.h

5.90 Arc::InformationResponse Class Reference

Informational response from InfoSystem.

```
#include <InformationInterface.h>
```

Public Member Functions

- InformationResponse (SOAPEnvelope &soap)
- std::list< XMLNode > Result (void)

5.90.1 Detailed Description

Informational response from InfoSystem.

This is a convenience wrapper analyzing WS-ResourceProperties response from InfoSystem interface of service.

5.90.2 Constructor & Destructor Documentation

5.90.2.1 Arc::InformationResponse::InformationResponse (SOAPEnvelope & *soap*)

Constructor parses WS-ResourceProperties ressponse. Provided SOAPEnvelope object must be valid as long as this object is in use.

5.90.3 Member Function Documentation

5.90.3.1 std::list<XMLNode> Arc::InformationResponse::Result (void)

Returns set of attributes which were in SOAP message passed to constructor.

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

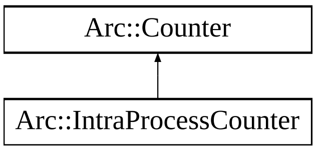
- InformationInterface.h

5.91 Arc::IntraProcessCounter Class Reference

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

`#include <IntraProcessCounter.h>`

Inheritance diagram for Arc::IntraProcessCounter::



Public Member Functions

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

Protected Member Functions

- virtual void cancel (IDType reservationID)
- virtual void extend (IDType &reservationID, Glib::TimeVal &expiryTime, Glib::TimeVal duration=ETERNAL)

5.91.1 Detailed Description

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

This is a class for shared among different threads within a single process. See the Counter class for further information about counters and examples of usage.

5.91.2 Constructor & Destructor Documentation

5.91.2.1 Arc::IntraProcessCounter::IntraProcessCounter (int *limit*, int *excess*)

Creates an IntraProcessCounter with specified limit and excess.

This constructor creates a counter with the specified limit (amount of resources available for reservation) and excess limit (an extra amount of resources that may be used for prioritized reservations).

Parameters:

limit The limit of the counter.

excess The excess limit of the counter.

5.91.2.2 virtual Arc::IntraProcessCounter::∼IntraProcessCounter () `[virtual]`

Destructor.

This is the destructor of the IntraProcessCounter class. Does not need to do anything.

5.91.3 Member Function Documentation

5.91.3.1 virtual void Arc::IntraProcessCounter::cancel (IDType *reservationID*) `[protected, virtual]`

Cancellation of a reservation.

This method cancels a reservation. It is called by the CounterTicket that corresponds to the reservation.

Parameters:

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

5.91.3.2 virtual int Arc::IntraProcessCounter::changeExcess (int *amount*) `[virtual]`

Changes the excess limit of the counter.

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

Parameters:

amount The amount by which to change the excess limit.

Returns:

The new excess limit.

Implements Arc::Counter.

5.91.3.3 virtual int Arc::IntraProcessCounter::changeLimit (int *amount*) `[virtual]`

Changes the limit of the counter.

Changes the limit of the counter by adding a certain amount to the current limit.

Parameters:

amount The amount by which to change the limit.

Returns:

The new limit.

Implements Arc::Counter.

5.91.3.4 virtual void Arc::IntraProcessCounter::extend (IDType & *reservationID*, Glib::TimeVal & *expiryTime*, Glib::TimeVal *duration* = ETERNAL) `[protected, virtual]`

Extension of a reservation.

This method extends a reservation. It is called by the CounterTicket that corresponds to the reservation.

Parameters:

reservationID Used for input as well as output. Contains the identification number of the original reservation on entry and the new identification number of the extended reservation on exit.

expiryTime Used for input as well as output. Contains the expiry time of the original reservation on entry and the new expiry time of the extended reservation on exit.

duration The time by which to extend the reservation. The new expiration time is computed based on the current time, NOT the previous expiration time.

5.91.3.5 virtual int Arc::IntraProcessCounter::getExcess () `[virtual]`

Returns the excess limit of the counter.

Returns the excess limit of the counter, i.e. by how much the usual limit may be exceeded by prioritized reservations.

Returns:

The excess limit.

Implements Arc::Counter.

5.91.3.6 virtual int Arc::IntraProcessCounter::getLimit () `[virtual]`

Returns the current limit of the counter.

This method returns the current limit of the counter, i.e. how many units can be reserved simultaneously by different threads without claiming high priority.

Returns:

The current limit of the counter.

Implements Arc::Counter.

5.91.3.7 virtual int Arc::IntraProcessCounter::getValue () `[virtual]`

Returns the current value of the counter.

Returns the current value of the counter, i.e. the number of unreserved units. Initially, the value is equal to the limit of the counter. When a reservation is made, the the value is decreased. Normally, the value should never be negative, but this may happen if there are prioritized reservations. It can also happen if the limit is decreased after some reservations have been made, since reservations are never revoked.

Returns:

The current value of the counter.

Implements Arc::Counter.

5.91.3.8 virtual CounterTicket Arc::IntraProcessCounter::reserve (int *amount* = `1`, Glib::TimeVal *duration* = `ETERNAL`, bool *prioritized* = `false`, Glib::TimeVal *timeOut* = `ETERNAL`) `[virtual]`

Makes a reservation from the counter.

This method makes a reservation from the counter. If the current value of the counter is too low to allow for the reservation, the method blocks until the reservation is possible or times out.

Parameters:

amount The amount to reserve, default value is 1.

duration The duration of a self expiring reservation, default is that it lasts forever.

prioritized Whether this reservation is prioritized and thus allowed to use the excess limit.

timeOut The maximum time to block if the value of the counter is too low, default is to allow "eternal" blocking.

Returns:

A CounterTicket that can be queried about the status of the reservation as well as for cancellations and extensions.

Implements Arc::Counter.

5.91.3.9 virtual int Arc::IntraProcessCounter::setExcess (int *newExcess*) `[virtual]`

Sets the excess limit of the counter.

This method sets a new excess limit for the counter.

Parameters:

newExcess The new excess limit, an absolute number.

Returns:

The new excess limit.

Implements Arc::Counter.

5.91.3.10 virtual int Arc::IntraProcessCounter::setLimit (int *newLimit*) `[virtual]`

Sets the limit of the counter.

This method sets a new limit for the counter.

Parameters:

newLimit The new limit, an absolute number.

Returns:

The new limit.

Implements Arc::Counter.

The documentation for this class was generated from the following file:

- IntraProcessCounter.h

5.92 Arc::Job Class Reference

Job.

```
#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)

5.92.1 Detailed Description

Job.

This class describe a Grid job. Most of the members contained in this class are directly linked to the ComputingActivity defined in the GLUE Specification v. 2.0 (GFD-R-P.147).

5.92.2 Constructor & Destructor Documentation

5.92.2.1 Arc::Job::Job ()

Create a Job object.

Default constructor. Takes no arguments.

5.92.3 Member Function Documentation

5.92.3.1 Job& Arc::Job::operator= (XMLNode *job*)

Set Job attributes from a XMLNode.

The attributes of the Job object is set to the values specified in the XMLNode. The XMLNode should be a ComputingActivity type using the GLUE2 XML hierarchical rendering, see http://forge.gridforum.org/sf/wiki/do/viewPage/projects.glue-wg/wiki/GLUE2XMLSchema for more information. Note that associations are not parsed.

Parameters:

job is a XMLNode of GLUE2 ComputingActivity type.

See also:

ToXML

5.92.3.2 void Arc::Job::Print (bool *longlist*) const

DEPRECATED: Print the Job information to std::cout.

This method is DEPRECATED, use the SaveToStream method instead. Method to print the Job attributes to std::cout

Parameters:

longlist is boolean for long listing (more details).

See also:

SaveToStream

5.92.3.3 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) method.

File locking: To avoid simultaneous use (writing and reading) of the file, reading will not be initiated before a lock on the file has been acquired. For this purpose the FileLock class is used. nTries specifies the maximal number of times the method will try to acquire a lock on the file, with an interval of tryInterval micro seconds between each attempt. If a lock is not acquired∗ this method returns false.

The method will also return false if the content of file is not in XML format. Otherwise it returns true.

Parameters:

filename is the filename of the job list to read jobs from.

jobs is a reference to a list of Job objects, which will be filled with the jobs read from file (cleared before use).

nTries specifies the maximal number of times the method will try to acquire a lock on file to read.

tryInterval specifies the interval (in micro seconds) between each attempt to acquire a lock.

Returns:

true in case of success, otherwise false.

See also:

operator=(XMLNode)
WriteJobsToTruncatedFile
WriteJobsToFile
RemoveJobsFromFile
FileLock
XMLNode::ReadFromFile

5.92.3.4 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 IDs from a file, and append them to a list.

This static method will read job IDs from the given file, and append the strings to the string list given as parameter. File locking will be done as described for the ReadAllJobsFromFile method. It returns false if the file was not readable, true otherwise, even if there were no IDs in the file. The lines of the file will be trimmed, and lines starting with # will be ignored.

Parameters:

filename is the filename of the jobidfile

jobids is a list of strings, to which the IDs read from the file will be appended

nTries specifies the maximal number of times the method will try to acquire a lock on file to read.

tryInterval specifies the interval (in micro seconds) between each attempt to acquire a lock.

Returns:

true in case of success, otherwise false.

5.92.3.5 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 objects which specifies which jobs from the file to remove.

nTries specifies the maximal number of times the method will try to acquire a lock on file to read.

tryInterval specifies the interval (in micro seconds) between each attempt to acquire a lock.

Returns:

true in case of success, otherwise false.

See also:

ReadAllJobsFromFile
WriteJobsToTruncatedFile
WriteJobsToFile
FileLock
XMLNode::ReadFromFile
XMLNode::SaveToFile

5.92.3.6 void Arc::Job::SaveToStream (std::ostream & *out*, bool *longlist*) const

Write job information to a std::ostream object.

This method will write job information to the passed std::ostream object. The longlist boolean specifies whether more (true) or less (false) information should be printed.

Parameters:

out is the std::ostream object to print the attributes to.

longlist is a boolean for switching on long listing (more details).

5.92.3.7 void Arc::Job::ToXML (XMLNode *job*) const

Add job information to a XMLNode.

Child nodes of GLUE ComputingActivity type containing job information of this object will be added to the passed XMLNode.

Parameters:

job is the XMLNode to add job information to in form of GLUE2 ComputingActivity type child nodes.

See also:

operator=

5.92.3.8 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 objects to be written to file

filename is the filename of file, where the URL objects will be appended to.

nTries specifies the maximal number of times the method will try to acquire a lock on file to read.

tryInterval specifies the interval (in micro seconds) between each attempt to acquire a lock.

Returns:

true in case of success, otherwise false.

5.92.3.9 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 object, and append it to the given file. File locking will be done as described for the ReadAllJobsFromFile method. It returns false if the file is not writable, true otherwise.

Parameters:

jobid is a jobID as a URL object

filename is the filename of the jobidfile, where the jobID will be appended

nTries specifies the maximal number of times the method will try to acquire a lock on file to read.

tryInterval specifies the interval (in micro seconds) between each attempt to acquire a lock.

Returns:

true in case of success, otherwise false.

5.92.3.10 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 objects which should be written to file.

newJobs is a reference to a list of pointers to Job 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
ReadAllJobsFromFile
WriteJobsToTruncatedFile
RemoveJobsFromFile
FileLock
XMLNode::SaveToFile

5.92.3.11 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) method, except for the information about new jobs which is disregarded.

See also:

WriteJobsToFile(const std::string&, const std::list<Job>&, std::list<const Job∗>&, unsigned, unsigned)

5.92.3.12 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 objects which should be written to file.

nTries specifies the maximal number of times the method will try to acquire a lock on file to read.

tryInterval specifies the interval (in micro seconds) between each attempt to acquire a lock.

Returns:

true in case of success, otherwise false.

See also:

ToXML

ReadAllJobsFromFile
WriteJobsToFile
RemoveJobsFromFile
FileLock
XMLNode::SaveToFile

The documentation for this class was generated from the following file:

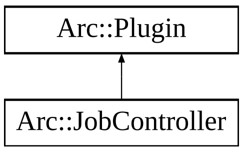
- Job.h

5.93 Arc::JobController Class Reference

Must be specialiced for each supported middleware flavour.

`#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)

5.93.1 Detailed Description

Must be specialiced for each supported middleware flavour.

The JobController is the base class for middleware specialized derived classes. The JobController 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 are private. The initialization of a (specialized) JobController object takes two steps. First the JobController specialization for the required grid flavour must be loaded by the JobControllerLoader, which sees to that the JobController 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 job pool (JobStore) which is the pool of jobs that the JobController can manage.

5.93.2 Member Function Documentation

5.93.2.1 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
GetJobInformation
JobState

5.93.2.2 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&) 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&)
GetJobInformation
JobState

5.93.2.3 void Arc::JobController::FillJobStore (const Job & *job*)

Fill jobstore.

5.93.2.4 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 with targets to migrate the job to.

broker Broker 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.

5.93.2.5 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
GetJobInformation
JobState

5.93.2.6 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::Save-ToStream method is called on each of the Job 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) 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
JobState

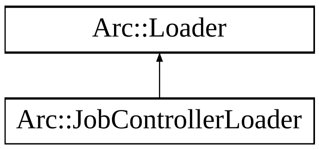
The documentation for this class was generated from the following file:

- JobController.h

5.94 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

5.94.1 Detailed Description

Class responsible for loading JobController plugins The JobController objects returned by a JobController-Loader must not be used after the JobControllerLoader goes out of scope.

5.94.2 Constructor & Destructor Documentation

5.94.2.1 Arc::JobControllerLoader::JobControllerLoader ()

Constructor Creates a new JobControllerLoader.

5.94.2.2 Arc::JobControllerLoader::∼JobControllerLoader ()

Destructor Calling the destructor destroys all JobControllers loaded by the JobControllerLoader instance.

5.94.3 Member Function Documentation

5.94.3.1 const std::list<JobController∗>& Arc::JobControllerLoader::GetJobControllers () const `[inline]`

Retrieve the list of loaded JobControllers.

Returns:

A reference to the list of JobControllers.

5.94.3.2 JobController∗ Arc::JobControllerLoader::load (const std::string & *name*, const UserConfig & *usercfg*)

Load a new JobController

Parameters:

name The name of the JobController to load.

usercfg The UserConfig object for the new JobController.

Returns:

A pointer to the new JobController (NULL on error).

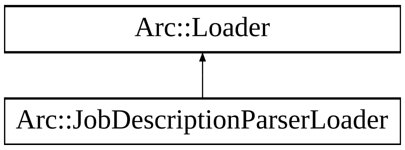
The documentation for this class was generated from the following file:

- JobController.h

5.95 Arc::JobDescriptionParserLoader Class Reference

`#include <JobDescriptionParser.h>`

Inheritance diagram for Arc::JobDescriptionParserLoader::



Public Member Functions

- JobDescriptionParserLoader ()
- ∼JobDescriptionParserLoader ()
- JobDescriptionParser ∗ load (const std::string &name)
- const std::list< JobDescriptionParser ∗ > & GetJobDescriptionParsers () const

Data Structures

- class **iterator**

5.95.1 Detailed Description

Class responsible for loading JobDescriptionParser plugins The JobDescriptionParser objects returned by a JobDescriptionParserLoader must not be used after the JobDescriptionParserLoader goes out of scope.

5.95.2 Constructor & Destructor Documentation

5.95.2.1 Arc::JobDescriptionParserLoader::JobDescriptionParserLoader ()

Constructor Creates a new JobDescriptionParserLoader.

5.95.2.2 Arc::JobDescriptionParserLoader::∼JobDescriptionParserLoader ()

Destructor Calling the destructor destroys all JobDescriptionParser object loaded by the JobDescription-ParserLoader instance.

5.95.3 Member Function Documentation

5.95.3.1 const std::list<JobDescriptionParser∗>& Arc::JobDescriptionParserLoader::GetJob-DescriptionParsers () const `[inline]`

Retrieve the list of loaded JobDescriptionParser objects.

Returns:

A reference to the list of JobDescriptionParser objects.

5.95.3.2 JobDescriptionParser∗ Arc::JobDescriptionParserLoader::load (const std::string & *name*)

Load a new JobDescriptionParser

Parameters:

name The name of the JobDescriptionParser to load.

Returns:

A pointer to the new JobDescriptionParser (NULL on error).

The documentation for this class was generated from the following file:

- JobDescriptionParser.h

5.96 Arc::JobState Class Reference

`#include <JobState.h>`

Public Member Functions

- bool IsFinished () const

5.96.1 Detailed Description

ARC general state model. The class comprise the general state model of the ARC-lib, and are herein used to compare job states from the different middlewares supported by the plugin structure of the ARC-lib. Which is why every ACC plugin should contain a class derived from this class. The derived class should consist of a constructor and a mapping function (a JobStateMap) which maps a std::string to a JobState:StateType. An example of a constructor in a plugin could be: JobStatePlugin::JobStatePluging(const std::string& state) : JobState(state, &pluginStateMap) { } where &pluginStateMap is a reference to the JobStateMap defined by the derived class.

5.96.2 Member Function Documentation

5.96.2.1 bool Arc::JobState::IsFinished () const `[inline]`

Check if state is finished.

Returns:

true is returned if the StateType is equal to FINISHED, KILLED, FAILED or DELETED, otherwise false is returned.

The documentation for this class was generated from the following file:

- JobState.h

5.97 Arc::JobSupervisor Class Reference

% JobSupervisor 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 ()

5.97.1 Detailed Description

% JobSupervisor class

The JobSupervisor class is tool for loading JobController plugins for managing Grid jobs.

5.97.2 Constructor & Destructor Documentation

5.97.2.1 Arc::JobSupervisor::JobSupervisor (const UserConfig & *usercfg*, const std::list< std::string > & *jobs*)

Create a JobSupervisor object.

Default constructor to create a JobSupervisor. Automatically loads JobController plugins based upon the input jobids.

Parameters:

usercfg Reference to UserConfig object with information about user credentials and joblistfile.

jobs List of jobs(jobid or job name) to be managed.

5.97.2.2 Arc::JobSupervisor::JobSupervisor (const UserConfig & *usercfg*, const std::list< Job > & *jobs*)

Create a JobSupervisor.

The list of Job objects passed to the constructor will be managed by this JobSupervisor, through the JobController class. It is important that the Flavour member of each Job object is set and correspond to the JobController plugin which are capable of managing that specific job. The JobController plugin will be loaded using the JobControllerLoader class, loading a plugin of type "HED:JobController" and name specified by the Flavour member, and the a reference to the UserConfig object usercfg will be passed to the plugin. Additionally a reference to the UserConfig object usercfg will be stored, thus usercfg must exist throughout the scope of the created object. If the Flavour member of a Job object is unset, a VERBOSE log message will be reported and that Job object will be ignored. If the JobController plugin for a given Flavour cannot be loaded, a WARNING log message will be reported and any Job object with that Flavour will be ignored. If loading of a specific plugin failed, that plugin will not be tried loaded for subsequent Job objects requiring that plugin. Job objects, for which the corresponding JobController plugin loaded successfully, will be added to that plugin using the JobController::FillJobStore(const Job&) method.

Parameters:

usercfg UserConfig object to pass to JobController plugins and to use in member methods.

jobs List of Job objects which will be managed by the created object.

5.97.3 Member Function Documentation

5.97.3.1 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 is equal to any in the jobids list. Only jobs corresponding to a Job object managed by this JobSupervisor will be considered for cancellation. Job objects not in a valid state (see JobState) will not be considered, and the IDFromEndpoint URLs of those objects will be appended to the notcancelled URL list. For jobs not in a finished state (see JobState::IsFinished), the JobController::Cancel method will be called, passing the corresponding Job 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 will be appended to the list to be returned. If the JobController::Cancel call fails the IDFromEndpoint URL is appended to the notkilled URL list.

Note: If there is any URL in the jobids list for which there is no corresponding Job 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 objects for which a corresponding job, managed by this JobSupervisor should be cancelled.

notcancelled List of Job::IDFromEndpoint URL objects for which the corresponding job were not cancelled.

Returns:

The list of Job::IDFromEndpoint URL objects of successfully cancelled or finished jobs is returned.

5.97.3.2 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 is equal to any in the jobids list. Only jobs corresponding to a Job object managed by this JobSupervisor will be considered for cleaning. Job objects not in a valid state (see JobState) will not be considered, and the IDFromEndpoint URLs of those objects will be appended to the notcleaned URL list, otherwise the JobController::Clean method will be called, passing the corresponding Job object, in order to clean the job. If that method fails the IDFromEndpoint URL of the Job object will be appended to the notcleaned URL list, and if it succeeds the IDFromEndpoint URL will be appended to the list of URL objects to be returned.

Note: If there is any URL in the jobids list for which there is no corresponding Job 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 objects for which a corresponding job, managed by this JobSupervisor should be cleaned.

notcleaned List of Job::IDFromEndpoint URL objects for which the corresponding job were not cleaned.

Returns:

The list of Job::IDFromEndpoint URL objects of successfully cleaned jobs is returned.

5.97.3.3 const std::list<JobController∗>& Arc::JobSupervisor::GetJobControllers () `[inline]`

Get list of JobControllers.

Method to get the list of JobControllers loaded by constructor.

5.97.3.4 bool Arc::JobSupervisor::Migrate (bool *forcemigration*, std::list< Job > & *migratedJobs*, std::list< URL > & *notmigrated*)

Migrate jobs.

Jobs managed by this JobSupervisor will be migrated when invoking this method, that is the job description of a job will be tried obtained, and if successful a job migration request will be sent, based on that job description.

Before identifying jobs to be migrated, the JobController::GetJobInformation method is called for each loaded JobController in order to retrieve the most up to date job information. Only jobs for which the State member of the Job object has the value JobState::QUEUEING, will be considered for migration. Furthermore the job description must be obtained (either locally or remote) and successfully parsed in order for a job to be migrated. If the job description cannot be obtained or parsed an ERROR log message is reported, and the IDFromEndpoint URL of the Job object is appended to the notmigrated list. If no jobs have been identified for migration, false will be returned in case ERRORs were reported, otherwise true is returned.

The execution services which can be targeted for migration are those specified in the UserConfig object of this class, as selected services. Before initiating any job migration request, resource discovery and broker∗ loading is carried out using the TargetGenerator and Broker classes, initialised by the UserConfig object of this class. If Broker loading fails, or no ExecutionTargets are found, an ERROR log message is reported and all IDFromEndpoint URLs for job considered for migration will be appended to the notmigrated list and then false will be returned.

When the above checks have been carried out successfully, the following is done for each job considered for migration. The ActivityOldId member of the Identification member in the job description will be set to that of the Job object, and the IDFromEndpoint URL will be appended to ActivityOldId member of the job description. After that the Broker object will be used to find a suitable ExecutionTarget object, and if found a migrate request will tried sent using the ExecutionTarget::Migrate method, passing the UserConfig object of this class. The passed forcemigration boolean indicates whether the migration request at the service side should ignore failures in cancelling the existing queuing job. If the request succeeds, the corresponding new Job object is appended to the migratedJobs list. If no suitable ExecutionTarget objects are found an ERROR log message is reported and the IDFromEndpoint URL of the Job object is appended to the notmigrated list. When all jobs have been processed, false is returned if any ERRORs were reported, otherwise true.

Parameters:

forcemigration indicates whether migration should succeed if service fails to cancel the existing queuing job.

migratedJobs list of Job objects which migrated jobs will be appended to.

notmigrated list of URL objects which the IDFromEndpoint URL will be appended to.

Returns:

false if any error is encountered, otherwise true.

5.97.3.5 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 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 in order to retrieve the most up to date job information. If an empty status-filter is specified, all jobs managed by this JobSupervisor will be considered for resubmission, except jobs in the undefined state (see JobState). If the status-filter is not empty, then only jobs with a general or specific state (see JobState) identical to any of the entries in the status-filter will be considered, except jobs in the undefined state. Jobs for which a job description cannot be obtained and successfully parsed will not be considered and an ERROR log message is reported, and the IDFromEndpoint URL is appended to the notresubmitted list. Job descriptions will be tried obtained either from Job object itself, or fetching them remotely. Furthermore if a Job object has the LocalInputFiles object set, then the checksum of each of the local input files specified in that object (key) will be calculated and verified to match the checksum LocalInputFiles object (value). If checksums are not matching the job will be filtered, and an ERROR log message is reported and the IDFromEndpoint URL is appended to the notresubmitted list. If no job have been identified for resubmission, false will be returned if ERRORs were reported, otherwise true is returned.

The destination for jobs is partly determined by the destination parameter. If a value of 1 is specified a job will only be targeted to the execution service (ES) on which it reside. A value of 2 indicates that a job should not be targeted to the ES it currently reside. Specifying any other value will target any ES. The ESs which can be targeted are those specified in the UserConfig object of this class, as selected services. Before initiating any job submission, resource discovery and broker loading is carried out using the TargetGenerator and Broker classes, initialised by the UserConfig object of this class. If Broker loading fails, or no ExecutionTargets are found, an ERROR log message is reported and all IDFromEndpoint URLs for job considered for resubmission will be appended to the notresubmitted list and then false will be returned.

When the above checks have been carried out successfully, then the Broker::Submit method will be invoked for each considered for resubmission. If it fails the IDFromEndpoint URL for the job is appended to the notresubmitted list, and an ERROR is reported. If submission succeeds the new job represented by a Job object will be appended to the resubmittedJobs list - it will not be added to this JobSupervisor. The method returns false if ERRORs were reported otherwise true is returned.

Parameters:

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).

resubmittedJobs list of Job objects which resubmitted jobs will be appended to.

notresubmitted list of URL objects which the IDFromEndpoint URL will be appended to.

Returns:

false if any error is encountered, otherwise true.

The documentation for this class was generated from the following file:

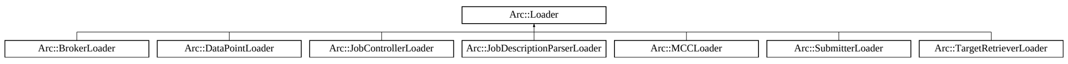
- JobSupervisor.h

5.98 Arc::Loader Class Reference

Plugins loader.

`#include <Loader.h>`

Inheritance diagram for Arc::Loader::



Public Member Functions

- Loader (XMLNode cfg)
- ~Loader ()

Protected Attributes

- PluginsFactory ∗ factory_

5.98.1 Detailed Description

Plugins loader.

This class processes XML configration and loads specified plugins. Accepted configuration is defined by XML schema mcc.xsd. "Plugins" elements are parsed by this class and corresponding libraries are loaded.

5.98.2 Constructor & Destructor Documentation

5.98.2.1 Arc::Loader::Loader (XMLNode *cfg*)

Constructor that takes whole XML configuration and performs common configuration part

5.98.2.2 Arc::Loader::~Loader ()

Destructor destroys all components created by constructor

5.98.3 Field Documentation

5.98.3.1 PluginsFactory∗ Arc::Loader::factory_ `[protected]`

Link to Factory responsible for loading and creation of Plugin and derived objects

The documentation for this class was generated from the following file:

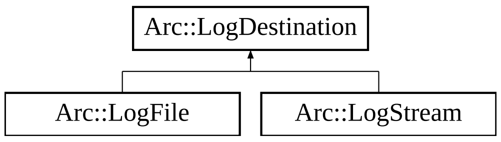
- Loader.h

5.99 Arc::LogDestination Class Reference

A base class for log destinations.

`#include <Logger.h>`

Inheritance diagram for Arc::LogDestination::



Public Member Functions

- virtual void log (const LogMessage &message)=0

Protected Member Functions

- LogDestination ()
- LogDestination (const std::string &locale)

5.99.1 Detailed Description

A base class for log destinations.

This class defines an interface for LogDestinations. LogDestination objects will typically contain synchronization mechanisms and should therefore never be copied.

5.99.2 Constructor & Destructor Documentation

5.99.2.1 Arc::LogDestination::LogDestination () `[protected]`

Default constructor.

This destination will use the default locale.

5.99.2.2 Arc::LogDestination::LogDestination (const std::string & *locale*) `[protected]`

Constructor with specific locale.

This destination will use the specified locale.

5.99.3 Member Function Documentation

5.99.3.1 virtual void Arc::LogDestination::log (const LogMessage & *message*) `[pure virtual]`

Logs a LogMessage to this LogDestination.

Implemented in Arc::LogStream, and Arc::LogFile.

The documentation for this class was generated from the following file:

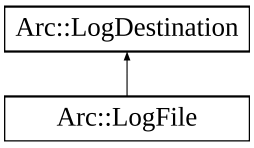
- Logger.h

5.100 Arc::LogFile Class Reference

A class for logging to files.

`#include <Logger.h>`

Inheritance diagram for Arc::LogFile::



Public Member Functions

- LogFile (const std::string &path)
- LogFile (const std::string &path, const std::string &locale)
- void setMaxSize (int newsize)
- void setBackups (int newbackup)
- void setReopen (bool newreopen)
- operator bool (void)
- bool operator! (void)
- virtual void log (const LogMessage &message)

5.100.1 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.

5.100.2 Constructor & Destructor Documentation

5.100.2.1 Arc::LogFile::LogFile (const std::string & *path*)

Creates a LogFile connected to a file.

Creates a LogFile connected to the file located at specified path. In order not to break synchronization, it is important not to connect more than one LogFile object to a certain file. If file does not exist it will be created.

Parameters:

path The path to file to which to write LogMessages.

5.100.2.2 Arc::LogFile::LogFile (const std::string & *path*, const std::string & *locale*)

Creates a LogFile connected to a file.

Creates a LogFile connected to the file located at specified path. The output will be localised to the specified locale.

5.100.3 Member Function Documentation

5.100.3.1 virtual void Arc::LogFile::log (const LogMessage & *message*) `[virtual]`

Writes a LogMessage to the file.

This method writes a LogMessage to the file that is connected to this LogFile object. If after writitng size of file exceeds one set by setMaxSize() file is moved to backup and new one is created.

Parameters:

message The LogMessage to write.

Implements Arc::LogDestination.

5.100.3.2 Arc::LogFile::operator bool (void)

Returns true if this instance is valid.

5.100.3.3 bool Arc::LogFile::operator! (void)

Returns true if this instance is invalid.

5.100.3.4 void Arc::LogFile::setBackups (int *newbackup*)

Set number of backups to store.

Set number of backups to store. When file size exceeds one specified with setMaxSize() file is closed and moved to one named path.1. If path.1 exists it is moved to path.2 and so on. Number of path.# files is one set in newbackup.

Parameters:

newbackup Number of backup files.

5.100.3.5 void Arc::LogFile::setMaxSize (int *newsize*)

Set maximal allowed size of file.

Set maximal allowed size of file. This value is not obeyed exactly. Spesified size may be exceeded by amount of one LogMessage. To disable limit specify -1.

Parameters:

newsize Max size of log file.

5.100.3.6 void Arc::LogFile::setReopen (bool *newreopen*)

Set file reopen on every write.

Set file reopen on every write. If set to true file is opened before writing every log record and closed afterward.

Parameters:

newreopen If file to be reopened for every log record.

The documentation for this class was generated from the following file:

- Logger.h

5.101 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)

5.101.1 Detailed Description

A logger class.

This class defines a Logger to which LogMessages can be sent.

Every Logger (except for the rootLogger) has a parent Logger. The domain of a Logger (a string that indicates the origin of LogMessages) is composed by adding a subdomain to the domain of its parent Logger.

A Logger also has a threshold. Every LogMessage that have a level that is greater than or equal to the threshold is forwarded to any LogDestination connected to this Logger as well as to the parent Logger.

Typical usage of the Logger class is to declare a global Logger object for each library/module/component to be used by all classes and methods there.

5.101.2 Constructor & Destructor Documentation

5.101.2.1 Arc::Logger::Logger (Logger & *parent*, const std::string & *subdomain*)

Creates a logger.

Creates a logger. The threshold is inherited from its parent Logger.

Parameters:

parent The parent Logger of the new Logger.

subdomain The subdomain of the new logger.

5.101.2.2 Arc::Logger::Logger (Logger & *parent*, const std::string & *subdomain*, LogLevel *threshold*)

Creates a logger.

Creates a logger.

Parameters:

parent The parent Logger of the new Logger.

subdomain The subdomain of the new logger.

threshold The threshold of the new logger.

5.101.2.3 Arc::Logger::∼Logger ()

Destroys a logger.

Destructor

5.101.3 Member Function Documentation

5.101.3.1 void Arc::Logger::addDestination (LogDestination & *destination*)

Adds a LogDestination.

Adds a LogDestination to which to forward LogMessages sent to this logger (if they pass the threshold). Since LogDestinatoins should not be copied, the new LogDestination is passed by reference and a pointer to it is kept for later use. It is therefore important that the LogDestination passed to this Logger exists at least as long as the Logger iteslf.

5.101.3.2 void Arc::Logger::addDestinations (const std::list< LogDestination ∗ > & *destinations*)

Adds LogDestinations.

See addDestination(LogDestination& destination).

5.101.3.3 const std::list<LogDestination∗>& Arc::Logger::getDestinations (void) const

Obtains current LogDestinations.

Returns list of pointers to LogDestination objects. Returned result refers directly to internal member of Logger intance. Hence it should not be used after this Logger is destroyed.

5.101.3.4 static Logger& Arc::Logger::getRootLogger () `[static]`

The root Logger.

This is the root Logger. It is an ancestor of any other Logger and allways exists.

5.101.3.5 LogLevel Arc::Logger::getThreshold () const

Returns the threshold.

Returns the threshold.

Returns:

The threshold of this Logger.

5.101.3.6 void Arc::Logger::msg (LogLevel *level*, const std::string & *str*) `[inline]`

Logs a message text.

Logs a message text string at the specified LogLevel. This is a convenience method to save some typing. It simply creates a LogMessage and sends it to the other msg() method.

Parameters:

level The level of the message.

str The message text.

5.101.3.7 void Arc::Logger::msg (LogMessage *message*)

Sends a LogMessage.

Sends a LogMessage.

Parameters:

The LogMessage to send.

5.101.3.8 void Arc::Logger::removeDestinations (void)

Removes all LogDestinations.

5.101.3.9 void Arc::Logger::setThreadContext (void)

Creates per-thread context.

Creates new context for this logger which becomes effective for operations initiated by this thread. All new threads started by this one will inherit new context. Context stores current threshold and pointers to destinations. Hence new context is identical to current one. One can modify new context using setThreshold(), removeDestinations() and addDestination(). All such operations will not affect old context.

5.101.3.10 void Arc::Logger::setThreshold (LogLevel *threshold*)

Sets the threshold.

This method sets the threshold of the Logger. Any message sent to this Logger that has a level below this threshold will be discarded.

Parameters:

The threshold

5.101.3.11 static void Arc::Logger::setThresholdForDomain (LogLevel *threshold*, const std::string & *domain*) `[static]`

Sets the threshold for domain.

This method sets the default threshold of the domain. All new loggers created with specified domain will have specified threshold set by default. The domain is composed of all subdomains of all loggers in chain by merging them with '.' as separator.

Parameters:

threshold The threshold

domain The domain of logger

5.101.3.12 static void Arc::Logger::setThresholdForDomain (LogLevel *threshold*, const std::list< std::string > & *subdomains*) `[static]`

Sets the threshold for domain.

This method sets the default threshold of the domain. All new loggers created with specified domain will have specified threshold set by default. The subdomains of all loggers in chain are matched against list of provided subdomains.

Parameters:

threshold The threshold

subdomains The subdomains of all loggers in chain

The documentation for this class was generated from the following file:

- Logger.h

5.102 Arc::LoggerContext Class Reference

Container for logger configuration.

```
#include <Logger.h>
```

5.102.1 Detailed Description

Container for logger configuration.

The documentation for this class was generated from the following file:

- Logger.h

5.103 Arc::LogMessage Class Reference

A class for log messages.

`#include <Logger.h>`

Public Member Functions

- LogMessage (LogLevel level, const IString &message)
- LogMessage (LogLevel level, const IString &message, const std::string &identifier)
- LogLevel getLevel () const

Protected Member Functions

- void setIdentifier (std::string identifier)

Friends

- class Logger
- std::ostream & operator<< (std::ostream &os, const LogMessage &message)

5.103.1 Detailed Description

A class for log messages.

This class is used to represent log messages internally. It contains the time the message was created, its level, from which domain it was sent, an identifier and the message text itself.

5.103.2 Constructor & Destructor Documentation

5.103.2.1 Arc::LogMessage::LogMessage (LogLevel *level*, const IString & *message*)

Creates a LogMessage with the specified level and message text.

This constructor creates a LogMessage with the specified level and message text. The time is set automatically, the domain is set by the Logger to which the LogMessage is sent and the identifier is composed from the process ID and the address of the Thread object corresponding to the calling thread.

Parameters:

level The level of the LogMessage.

message The message text.

5.103.2.2 Arc::LogMessage::LogMessage (LogLevel *level*, const IString & *message*, const std::string & *identifier*)

Creates a LogMessage with the specified attributes.

This constructor creates a LogMessage with the specified level, message text and identifier. The time is set automatically and the domain is set by the Logger to which the LogMessage is sent.

Parameters:

level The level of the LogMessage.

message The message text.

ident The identifier of the LogMessage.

5.103.3 Member Function Documentation

5.103.3.1 LogLevel Arc::LogMessage::getLevel () const

Returns the level of the LogMessage.

Returns the level of the LogMessage.

Returns:

The level of the LogMessage.

5.103.3.2 void Arc::LogMessage::setIdentifier (std::string *identifier*) `[protected]`

Sets the identifier of the LogMessage.

The purpose of this method is to allow subclasses (in case there are any) to set the identifier of a LogMessage.

Parameters:

The identifier.

5.103.4 Friends And Related Function Documentation

5.103.4.1 friend class Logger `[friend]`

The Logger class is a friend.

The Logger class must have some privileges (e.g. ability to call the setDomain() method), therefore it is a friend.

5.103.4.2 std::ostream& operator<< (std::ostream & *os*, const LogMessage & *message*) `[friend]`

Printing of LogMessages to ostreams.

Output operator so that LogMessages can be printed conveniently by LogDestinations.

The documentation for this class was generated from the following file:

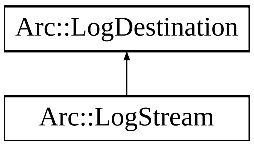
- Logger.h

5.104 Arc::LogStream Class Reference

A class for logging to ostreams.

`#include <Logger.h>`

Inheritance diagram for Arc::LogStream::



Public Member Functions

- LogStream (std::ostream &destination)
- LogStream (std::ostream &destination, const std::string &locale)
- virtual void log (const LogMessage &message)

5.104.1 Detailed Description

A class for logging to ostreams.

This class is used for logging to ostreams (cout, cerr, files). It provides synchronization in order to prevent different LogMessages to appear mixed with each other in the stream. In order not to break the synchronization, LogStreams should never be copied. Therefore the copy constructor and assignment operator are private. Furthermore, it is important to keep a LogStream object as long as the Logger to which it has been registered.

5.104.2 Constructor & Destructor Documentation

5.104.2.1 Arc::LogStream::LogStream (std::ostream & *destination*)

Creates a LogStream connected to an ostream.

Creates a LogStream connected to the specified ostream. In order not to break synchronization, it is important not to connect more than one LogStream object to a certain stream.

Parameters:

destination The ostream to which to erite LogMessages.

5.104.2.2 Arc::LogStream::LogStream (std::ostream & *destination*, const std::string & *locale*)

Creates a LogStream connected to an ostream.

Creates a LogStream connected to the specified ostream. The output will be localised to the specified locale.

5.104.3 Member Function Documentation

5.104.3.1 virtual void Arc::LogStream::log (const LogMessage & *message*) `[virtual]`

Writes a LogMessage to the stream.

This method writes a LogMessage to the ostream that is connected to this LogStream object. It is synchronized so that not more than one LogMessage can be written at a time.

Parameters:

message The LogMessage to write.

Implements Arc::LogDestination.

The documentation for this class was generated from the following file:

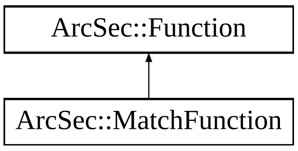
- Logger.h

5.105 ArcSec::MatchFunction Class Reference

Evaluate whether arg1 (value in regular expression) matched arg0 (lable in regular expression).

`#include <MatchFunction.h>`

Inheritance diagram for ArcSec::MatchFunction::



Public Member Functions

- virtual AttributeValue ∗ evaluate (AttributeValue ∗arg0, AttributeValue ∗arg1, bool check_id=true)
- virtual std::list< AttributeValue ∗ > evaluate (std::list< AttributeValue ∗ > args, bool check_-id=true)

Static Public Member Functions

- static std::string getFunctionName (std::string datatype)

5.105.1 Detailed Description

Evaluate whether arg1 (value in regular expression) matched arg0 (lable in regular expression).

5.105.2 Member Function Documentation

5.105.2.1 virtual std::list<AttributeValue∗> ArcSec::MatchFunction::evaluate (std::list< AttributeValue ∗ > *args*, bool *check_id* = `true`) `[virtual]`

Evaluate a list of AttributeValue objects, and return a list of Attribute objects

Implements ArcSec::Function.

5.105.2.2 virtual AttributeValue∗ ArcSec::MatchFunction::evaluate (AttributeValue ∗ *arg0*, AttributeValue ∗ *arg1*, bool *check_id* = `true`) `[virtual]`

Evaluate two AttributeValue objects, and return one AttributeValue object

Implements ArcSec::Function.

5.105.2.3 static std::string ArcSec::MatchFunction::getFunctionName (std::string *datatype*) `[static]`

help function to get the FunctionName

The documentation for this class was generated from the following file:

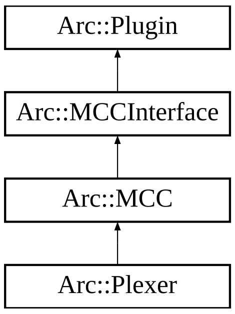
- MatchFunction.h

5.106 Arc::MCC Class Reference

Message Chain Component - base class for every MCC plugin.

`#include <MCC.h>`

Inheritance diagram for Arc::MCC::



Public Member Functions

- MCC (Config ∗)
- virtual void Next (MCCInterface ∗next, const std::string &label="")
- virtual void AddSecHandler (Config ∗cfg, ArcSec::SecHandler ∗sechandler, const std::string &label="")
- virtual void Unlink ()
- virtual MCC_Status process (Message &, Message &)

Protected Member Functions

- bool ProcessSecHandlers (Message &message, const std::string &label="") const

Protected Attributes

- std::map< std::string, MCCInterface ∗ > next_
- std::map< std::string, std::list< ArcSec::SecHandler ∗ > > sechandlers_

Static Protected Attributes

- static Logger logger

5.106.1 Detailed Description

Message Chain Component - base class for every MCC plugin.

This is partialy virtual class which defines interface and common functionality for every MCC plugin needed for managing of component in a chain.

5.106.2 Constructor & Destructor Documentation

5.106.2.1 Arc::MCC::MCC (Config ∗) `[inline]`

Example contructor - MCC takes at least it's configuration subtree

5.106.3 Member Function Documentation

5.106.3.1 virtual void Arc::MCC::AddSecHandler (Config ∗ *cfg*, ArcSec::SecHandler ∗ *sechandler*, const std::string & *label* = "") `[virtual]`

Add security components/handlers to this MCC. Security handlers are stacked into a few queues with each queue identified by its label. The queue labelled 'incoming' is executed for every 'request' message after the message is processed by the MCC on the service side and before processing on the client side. The queue labelled 'outgoing' is run for response message before it is processed by MCC algorithms on the service side and after processing on the client side. Those labels are just a matter of agreement and some MCCs may implement different queues executed at various message processing steps.

5.106.3.2 virtual void Arc::MCC::Next (MCCInterface ∗ *next*, const std::string & *label* = "") `[virtual]`

Add reference to next MCC in chain. This method is called by Loader for every potentially labeled link to next component which implements MCCInterface. If next is NULL corresponding link is removed.

Reimplemented in Arc::Plexer.

5.106.3.3 virtual MCC_Status Arc::MCC::process (Message &, Message &) `[inline, virtual]`

Dummy Message processing method. Just a placeholder.

Implements Arc::MCCInterface.

Reimplemented in Arc::Plexer.

5.106.3.4 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.

5.106.3.5 virtual void Arc::MCC::Unlink () `[virtual]`

Removing all links. Useful for destroying chains.

5.106.4 Field Documentation

5.106.4.1 Logger Arc::MCC::logger `[static, protected]`

A logger for MCCs.

A logger intended to be the parent of loggers in the different MCCs.

Reimplemented in Arc::Plexer.

5.106.4.2 std::map<std::string, MCCInterface *> Arc::MCC::next_ [protected]

Set of labeled "next" components. Each implemented MCC must call process() method of corresponding MCCInterface from this set in own process() method.

5.106.4.3 std::map<std::string, std::list<ArcSec::SecHandler *> > Arc::MCC::sechandlers_ [protected]

Set of labeled authentication and authorization handlers. MCC calls sequence of handlers at specific point depending on associated identifier. In most aces those are "in" and "out" for incoming and outgoing messages correspondingly.

The documentation for this class was generated from the following file:

- MCC.h

5.107 Arc::MCC_Status Class Reference

A class for communication of MCC processing results.

`#include <MCC_Status.h>`

Public Member Functions

- MCC_Status (StatusKind kind=STATUS_UNDEFINED, const std::string &origin="???", const std::string &explanation="No explanation.")
- bool isOk () const
- StatusKind getKind () const
- const std::string & getOrigin () const
- const std::string & getExplanation () const
- operator std::string () const
- operator bool (void) const
- bool operator! (void) const

5.107.1 Detailed Description

A class for communication of MCC processing results.

This class is used to communicate result status between MCCs. It contains a status kind, a string specifying the origin (MCC) of the status object and an explanation.

5.107.2 Constructor & Destructor Documentation

5.107.2.1 Arc::MCC_Status::MCC_Status (StatusKind *kind* = `STATUS_UNDEFINED`, const std::string & *origin* = `"???"`, const std::string & *explanation* = `"No explanation."`)

The constructor.

Creates a MCC_Status object.

Parameters:

kind The StatusKind (default: STATUS_UNDEFINED)

origin The origin MCC (default: "???")

explanation An explanation (default: "No explanation.")

5.107.3 Member Function Documentation

5.107.3.1 const std::string& Arc::MCC_Status::getExplanation () const

Returns an explanation.

This method returns an explanation of this object.

Returns:

An explanation of this object.

5.107.3.2 StatusKind Arc::MCC_Status::getKind () const

Returns the status kind.

Returns the status kind of this object.

Returns:

The status kind of this object.

5.107.3.3 const std::string& Arc::MCC_Status::getOrigin () const

Returns the origin.

This method returns a string specifying the origin MCC of this object.

Returns:

A string specifying the origin MCC of this object.

5.107.3.4 bool Arc::MCC_Status::isOk () const

Is the status kind ok?

This method returns true if the status kind of this object is STATUS_OK

Returns:

true if kind==STATUS_OK

5.107.3.5 Arc::MCC_Status::operator bool (void) const `[inline]`

Is the status kind ok?

This method returns true if the status kind of this object is STATUS_OK

Returns:

true if kind==STATUS_OK

5.107.3.6 Arc::MCC_Status::operator std::string () const

Conversion to string.

This operator converts a MCC_Status object to a string.

5.107.3.7 bool Arc::MCC_Status::operator! (void) const `[inline]`

not operator

Returns true if the status kind is not OK

Returns:

true if kind!=STATUS_OK

The documentation for this class was generated from the following file:

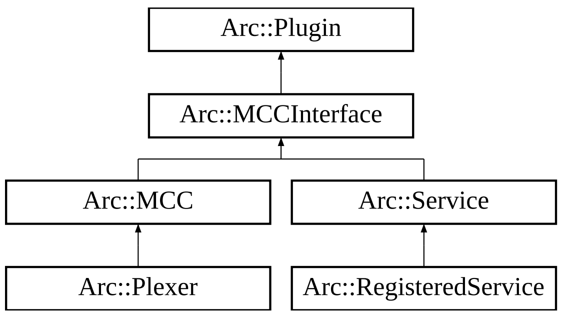
- MCC_Status.h

5.108 Arc::MCCInterface Class Reference

Interface for communication between MCC, Service and Plexer objects.

`#include <MCC.h>`

Inheritance diagram for Arc::MCCInterface::



Public Member Functions

- virtual MCC_Status process (Message &request, Message &response)=0

5.108.1 Detailed Description

Interface for communication between MCC, Service and Plexer objects.

The Interface consists of the method process() which is called by the previous MCC in the chain. For memory management policies please read the description of the Message class.

5.108.2 Member Function Documentation

5.108.2.1 virtual MCC_Status Arc::MCCInterface::process (Message & *request*, Message & *response*) `[pure virtual]`

Method for processing of requests and responses. This method is called by preceeding MCC in chain when a request needs to be processed. This method must call similar method of next MCC in chain unless any failure happens. Result returned by call to next MCC should be processed and passed back to previous MCC. In case of failure this method is expected to generate valid error response and return it back to previous MCC without calling the next one.

Parameters:

request	The request that needs to be processed.
response	A Message object that will contain the response of the request when the method returns.

Returns:

An object representing the status of the call.

Implemented in Arc::MCC, and Arc::Plexer.

The documentation for this class was generated from the following file:

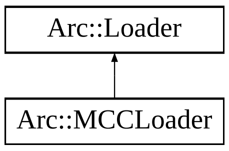
- MCC.h

5.109 Arc::MCCLoader Class Reference

Creator of Message Component Chains (MCC).

`#include <MCCLoader.h>`

Inheritance diagram for Arc::MCCLoader::



Public Member Functions

- MCCLoader (Config &cfg)
- ∼MCCLoader ()
- MCC ∗ operator[ ] (const std::string &id)

5.109.1 Detailed Description

Creator of Message Component Chains (MCC).

This class processes XML configration and creates message chains. Accepted configuration is defined by XML schema mcc.xsd. Supported components are of types MCC, Service and Plexer. MCC and Service are loaded from dynamic libraries. For Plexer only internal implementation is supported. This object is also a container for loaded componets. All components and chains are destroyed if this object is destroyed. Chains are created in 2 steps. First all components are loaded and corresponding objects are created. Constructors are supplied with corresponding configuration subtrees. During next step components are linked together by calling their Next() methods. Each call creates labeled link to next component in a chain. 2 step method has an advantage over single step because it allows loops in chains and makes loading procedure more simple. But that also means during short period of time components are only partly configured. Components in such state must produce proper error response if Message arrives. Note: Current implementation requires all components and links to be labeled. All labels must be unique. Future implementation will be able to assign labels automatically.

5.109.2 Constructor & Destructor Documentation

5.109.2.1 Arc::MCCLoader::MCCLoader (Config & *cfg*)

Constructor that takes whole XML configuration and creates component chains

5.109.2.2 Arc::MCCLoader::∼MCCLoader ()

Destructor destroys all components created by constructor

5.109.3 Member Function Documentation

5.109.3.1]

MCC∗ Arc::MCCLoader::operator[ ] (const std::string & *id*)

Access entry MCCs in chains. Those are components exposed for external access using ’entry’ attribute

The documentation for this class was generated from the following file:

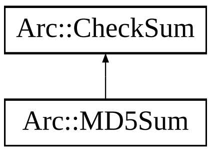
- MCCLoader.h

5.110 Arc::MD5Sum Class Reference

Implementation of MD5 checksum.

`#include <CheckSum.h>`

Inheritance diagram for Arc::MD5Sum::



5.110.1 Detailed Description

Implementation of MD5 checksum.

The documentation for this class was generated from the following file:

- CheckSum.h

5.111 Arc::Message Class Reference

Object being passed through chain of MCCs.

```
#include <Message.h>
```

Public Member Functions

- Message (void)
- Message (Message &msg)
- Message (long msg_ptr_addr)
- ∼Message (void)
- Message & operator= (Message &msg)
- MessagePayload ∗ Payload (void)
- MessagePayload ∗ Payload (MessagePayload ∗payload)
- MessageAttributes ∗ Attributes (void)
- MessageAuth ∗ Auth (void)
- MessageContext ∗ Context (void)
- MessageAuthContext ∗ AuthContext (void)
- void Context (MessageContext ∗ctx)
- void AuthContext (MessageAuthContext ∗auth_ctx)

5.111.1 Detailed Description

Object being passed through chain of MCCs.

An instance of this class refers to objects with main content (MessagePayload), authentication/authorization information (MessageAuth) and common purpose attributes (MessageAttributes). Message class does not manage pointers to objects and their content. It only serves for grouping those objects. Message objects are supposed to be processed by MCCs and Services implementing MCCInterface method process(). All objects constituting content of Message object are subject to following policies:

1. All objects created inside call to process() method using new command must be explicitely destroyed within same call using delete command with following exceptions. a) Objects which are assigned to 'response' Message. b) Objects whose management is completely acquired by objects assigned to 'response' Message.

2. All objects not created inside call to process() method are not explicitely destroyed within that call with following exception. a) Objects which are part of 'response' Method returned from call to next's process() method. Unless those objects are passed further to calling process(), of course.

3. It is not allowed to make 'response' point to same objects as 'request' does on entry to process() method. That is needed to avoid double destruction of same object. (Note: if in a future such need arises it may be solved by storing additional flags in Message object).

4. It is allowed to change content of pointers of 'request' Message. Calling process() method must not rely on that object to stay intact.

5. Called process() method should either fill 'response' Message with pointers to valid objects or to keep them intact. This makes it possible for calling process() to preload 'response' with valid error message.

5.111.2 Constructor & Destructor Documentation

5.111.2.1 Arc::Message::Message (void) `[inline]`

Dummy constructor

5.111.2.2 Arc::Message::Message (Message & *msg*) `[inline]`

Copy constructor. Ensures shallow copy.

5.111.2.3 Arc::Message::Message (long *msg_ptr_addr*)

Copy constructor. Used by language bindigs

5.111.2.4 Arc::Message::~Message (void) `[inline]`

Destructor does not affect refered objects except those created internally

5.111.3 Member Function Documentation

5.111.3.1 MessageAttributes* Arc::Message::Attributes (void) `[inline]`

Returns a pointer to the current attributes object or creates it if no attributes object has been assigned.

5.111.3.2 MessageAuth* Arc::Message::Auth (void) `[inline]`

Returns a pointer to the current authentication/authorization object or creates it if no object has been assigned.

5.111.3.3 void Arc::Message::AuthContext (MessageAuthContext * *auth_ctx*) `[inline]`

Assigns auth* context object

5.111.3.4 MessageAuthContext* Arc::Message::AuthContext (void) `[inline]`

Returns a pointer to the current auth* context object or creates it if no object has been assigned.

5.111.3.5 void Arc::Message::Context (MessageContext * *ctx*) `[inline]`

Assigns message context object

5.111.3.6 MessageContext* Arc::Message::Context (void) `[inline]`

Returns a pointer to the current context object or creates it if no object has been assigned. Last case should happen only if first MCC in a chain is connectionless like one implementing UDP protocol.

5.111.3.7 Message& Arc::Message::operator= (Message & *msg*) `[inline]`

Assignment. Ensures shallow copy.

5.111.3.8 MessagePayload* Arc::Message::Payload (MessagePayload * *payload*) `[inline]`

Replaces payload with new one. Returns the old one.

5.111.3.9 MessagePayload* Arc::Message::Payload (void) `[inline]`

Returns pointer to current payload or NULL if no payload assigned.

The documentation for this class was generated from the following file:

- Message.h

5.112 Arc::MessageAttributes Class Reference

A class for storage of attribute values.

`#include <MessageAttributes.h>`

Public Member Functions

- MessageAttributes ()
- void set (const std::string &key, const std::string &value)
- void add (const std::string &key, const std::string &value)
- void removeAll (const std::string &key)
- void remove (const std::string &key, const std::string &value)
- int count (const std::string &key) const
- const std::string & get (const std::string &key) const
- AttributeIterator getAll (const std::string &key) const
- AttributeIterator getAll (void) const

Protected Attributes

- AttrMap attributes_

5.112.1 Detailed Description

A class for storage of attribute values.

This class is used to store attributes of messages. All attribute keys and their corresponding values are stored as strings. Any key or value that is not a string must thus be represented as a string during storage. Furthermore, an attribute is usually a key-value pair with a unique key, but there may also be multiple such pairs with equal keys.

The key of an attribute is composed by the name of the Message Chain Component (MCC) which produce it and the name of the attribute itself with a colon (:) in between, i.e. MCC_Name:Attribute_Name. For example, the key of the "Content-Length" attribute of the HTTP MCC is thus "HTTP:Content-Length".

There are also "global attributes", which may be produced by different MCCs depending on the configuration. The keys of such attributes are NOT prefixed by the name of the producing MCC. Before any new global attribute is introduced, it must be agreed upon by the core development team and added below. The global attributes decided so far are:

- `Request-URI` Identifies the service to which the message shall be sent. This attribute is produced by e.g. the HTTP MCC and used by the plexer for routing the message to the appropriate service.

5.112.2 Constructor & Destructor Documentation

5.112.2.1 Arc::MessageAttributes::MessageAttributes ()

The default constructor.

This is the default constructor of the MessageAttributes class. It constructs an empty object that initially contains no attributes.

5.112.3 Member Function Documentation

5.112.3.1 void Arc::MessageAttributes::add (const std::string & *key*, const std::string & *value*)

Adds a value to an attribute.

This method adds a new value to an attribute. Any previous value will be preserved, i.e. the attribute may become multiple valued.

Parameters:

key The key of the attribute.

value The (new) value of the attribute.

5.112.3.2 int Arc::MessageAttributes::count (const std::string & *key*) const

Returns the number of values of an attribute.

Returns the number of values of an attribute that matches a certain key.

Parameters:

key The key of the attribute for which to count values.

Returns:

The number of values that corresponds to the key.

5.112.3.3 const std::string& Arc::MessageAttributes::get (const std::string & *key*) const

Returns the value of a single-valued attribute.

This method returns the value of a single-valued attribute. If the attribute is not single valued (i.e. there is no such attribute or it is a multiple-valued attribute) an empty string is returned.

Parameters:

key The key of the attribute for which to return the value.

Returns:

The value of the attribute.

5.112.3.4 AttributeIterator Arc::MessageAttributes::getAll (void) const

Access all value and attributes.

5.112.3.5 AttributeIterator Arc::MessageAttributes::getAll (const std::string & *key*) const

Access the value(s) of an attribute.

This method returns an AttributeIterator that can be used to access the values of an attribute.

Parameters:

key The key of the attribute for which to return the values.

Returns:

An AttributeIterator for access of the values of the attribute.

5.112.3.6 void Arc::MessageAttributes::remove (const std::string & *key*, const std::string & *value*)

Removes one value of an attribute.

This method removes a certain value from the attribute that matches a certain key.

Parameters:

key The key of the attribute from which the value shall be removed.

value The value to remove.

5.112.3.7 void Arc::MessageAttributes::removeAll (const std::string & *key*)

Removes all attributes with a certain key.

This method removes all attributes that match a certain key.

Parameters:

key The key of the attributes to remove.

5.112.3.8 void Arc::MessageAttributes::set (const std::string & *key*, const std::string & *value*)

Sets a unique value of an attribute.

This method removes any previous value of an attribute and sets the new value as the only value.

Parameters:

key The key of the attribute.

value The (new) value of the attribute.

5.112.4 Field Documentation

5.112.4.1 AttrMap Arc::MessageAttributes::attributes_ `[protected]`

Internal storage of attributes.

An AttrMap (multimap) in which all attributes (key-value pairs) are stored.

The documentation for this class was generated from the following file:

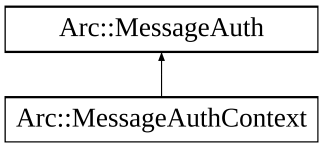
- MessageAttributes.h

5.113 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)

5.113.1 Detailed Description

Contains authencity information, authorization tokens and decisions.

This class only supports string keys and SecAttr values.

5.113.2 Member Function Documentation

5.113.2.1 bool Arc::MessageAuth::Export (SecAttrFormat *format*, XMLNode & *val*) const

Returns properly catenated attributes in specified format.

Content of XML node at is replaced with generated information if XML tree is empty. If tree at is not empty then Export() tries to merge generated information to already existing like everything would be generated inside same Export() method. If does not represent valid node then new XML tree is created.

5.113.2.2 MessageAuth∗ Arc::MessageAuth::Filter (const std::list< std::string > & *selected_keys*, const std::list< std::string > & *rejected_keys*)

Creates new instance of MessageAuth with attributes filtered.

In new instance all attributes with keys listed in are removed. If is not empty only corresponding attributes are transferred to new instance. Created instance does not own refered attributes. Hence parent instance must not be deleted as long as this one is in use.

5.113.2.3 SecAttr∗ Arc::MessageAuth::get (const std::string & *key*)

Retrieves reference to security attribute stored under specified key.

5.113.2.4]

SecAttr∗ Arc::MessageAuth::operator[ ] (const std::string & *key*) `[inline]`

Same as MessageAuth::get.

5.113.2.5 void Arc::MessageAuth::remove (const std::string & *key*)

Deletes security attribute stored under specified key.

5.113.2.6 void Arc::MessageAuth::set (const std::string & *key*, SecAttr ∗ *value*)

Adds/overwrites security attribute stored under specified key.

The documentation for this class was generated from the following file:

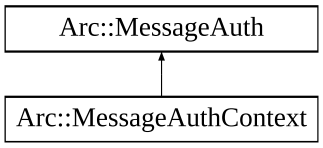
- MessageAuth.h

5.114 Arc::MessageAuthContext Class Reference

Handler for content of message auth∗ context.

`#include <Message.h>`

Inheritance diagram for Arc::MessageAuthContext::



5.114.1 Detailed Description

Handler for content of message auth∗ context.

This class is a container for authorization and authentication information. It gets associated with Message object usually by first MCC in a chain and is kept as long as connection persists.

The documentation for this class was generated from the following file:

- Message.h

5.115 Arc::MessageContext Class Reference

Handler for content of message context.

```
#include <Message.h>
```

Public Member Functions

- void Add (const std::string &name, MessageContextElement ∗element)

5.115.1 Detailed Description

Handler for content of message context.

This class is a container for objects derived from MessageContextElement. It gets associated with Message object usually by first MCC in a chain and is kept as long as connection persists.

5.115.2 Member Function Documentation

5.115.2.1 void Arc::MessageContext::Add (const std::string & *name*, MessageContextElement ∗ *element*)

Provided element is taken over by this class. It is remembered by it and destroyed when this class is destroyed.

The documentation for this class was generated from the following file:

- Message.h

5.116 Arc::MessageContextElement Class Reference

Top class for elements contained in message context.

```
#include <Message.h>
```

5.116.1 Detailed Description

Top class for elements contained in message context.

Objects of classes inherited with this one may be stored in MessageContext container.

The documentation for this class was generated from the following file:

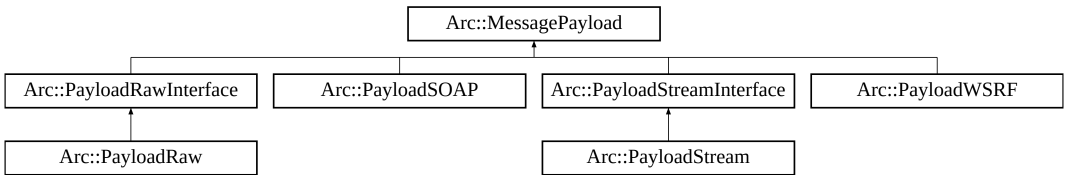
- Message.h

5.117 Arc::MessagePayload Class Reference

Base class for content of message passed through chain.

`#include <Message.h>`

Inheritance diagram for Arc::MessagePayload::



5.117.1 Detailed Description

Base class for content of message passed through chain.

It's not intended to be used directly. Instead functional classes must be derived from it.

The documentation for this class was generated from the following file:

- Message.h

5.118 Arc::ModuleDesc Class Reference

Description of loadable module.

```
#include <Plugin.h>
```

5.118.1 Detailed Description

Description of loadable module.

This class is used for reports

The documentation for this class was generated from the following file:

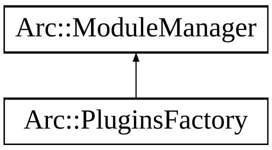
- Plugin.h

5.119 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)

5.119.1 Detailed Description

Manager of shared libraries.

This class loads shared libraries/modules. There supposed to be created one instance of it per executable. In such circumstances it would cache handles to loaded modules and not load them multiple times.

5.119.2 Constructor & Destructor Documentation

5.119.2.1 Arc::ModuleManager::ModuleManager (XMLNode *cfg*)

Constructor. It is supposed to process correponding configuration subtree and tune module loading parameters accordingly.

5.119.3 Member Function Documentation

5.119.3.1 std::string Arc::ModuleManager::find (const std::string & *name*)

Finds loadable module by 'name' looking in same places as load() does, but does not load it.

5.119.3.2 std::string Arc::ModuleManager::findLocation (const std::string & *name*)

Finds shared library corresponding to module 'name' and returns path to it

5.119.3.3 Glib::Module∗ Arc::ModuleManager::load (const std::string & *name*, bool *probe* = `false`)

Finds module 'name' in cache or loads corresponding loadable module

5.119.3.4 bool Arc::ModuleManager::makePersistent (const std::string & *name*)

Make sure this module is never unloaded. Even if unload() is called.

5.119.3.5 bool Arc::ModuleManager::makePersistent (Glib::Module ∗ *module*)

Make sure this module is never unloaded. Even if unload() is called.

5.119.3.6 Glib::Module∗ Arc::ModuleManager::reload (Glib::Module ∗ *module*)

Reload module previously loaded in probe mode. New module is loaded with all symbols resolved and old module handler is unloaded. In case of error old module is not unloaded.

5.119.3.7 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 to adopt the singleton pattern

5.119.3.8 void Arc::ModuleManager::unload (const std::string & *name*)

Unload module by its name

5.119.3.9 void Arc::ModuleManager::unload (Glib::Module ∗ *module*)

Unload module by its identifier

The documentation for this class was generated from the following file:

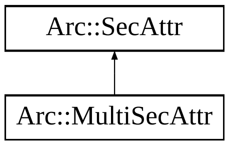
- ModuleManager.h

5.120 Arc::MultiSecAttr Class Reference

Container of multiple SecAttr attributes.

`#include <SecAttr.h>`

Inheritance diagram for Arc::MultiSecAttr::



Public Member Functions

- virtual operator bool () const
- virtual bool Export (SecAttrFormat format, XMLNode &val) const

5.120.1 Detailed Description

Container of multiple SecAttr attributes.

This class combines multiple attributes. It's export/import methods catenate results of underlying objects. Primary meaning of this class is to serve as base for classes implementing multi level hierarchical tree-like descriptions of user identity. It may also be used for collecting information of same source or kind. Like all information extracted from X509 certificate.

5.120.2 Member Function Documentation

5.120.2.1 virtual bool Arc::MultiSecAttr::Export (SecAttrFormat *format*, XMLNode & *val*) const `[virtual]`

Convert internal structure into specified format. Returns false if format is not supported/suitable for this attribute. XML node referenced by is turned into top level element of specified format.

Reimplemented from Arc::SecAttr.

5.120.2.2 virtual Arc::MultiSecAttr::operator bool () const `[virtual]`

This function should return false if the value is to be considered null, e.g. if it hasn't been set or initialized. In other cases it should return true.

Reimplemented from Arc::SecAttr.

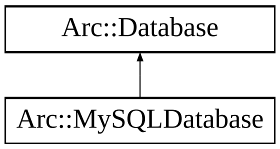
The documentation for this class was generated from the following file:

- SecAttr.h

5.121 Arc::MySQLDatabase Class Reference

`#include <MysqlWrapper.h>`

Inheritance diagram for Arc::MySQLDatabase::



Public Member Functions

- virtual bool connect (std::string &dbname, std::string &user, std::string &password)
- virtual bool isconnected () const
- virtual void close ()
- virtual bool enable_ssl (const std::string keyfile="", const std::string certfile="", const std::string cafile="", const std::string capath="")
- virtual bool shutdown ()

5.121.1 Detailed Description

Implement the database accessing interface in DBInterface.h by using mysql client library for accessing mysql database

5.121.2 Member Function Documentation

5.121.2.1 virtual void Arc::MySQLDatabase::close () `[virtual]`

Close the connection with database server

Implements Arc::Database.

5.121.2.2 virtual bool Arc::MySQLDatabase::connect (std::string & *dbname*, std::string & *user*, std::string & *password*) `[virtual]`

Do connection with database server

Parameters:

dbname The database name which will be used.

user The username which will be used to access database.

password The password which will be used to access database.

Implements Arc::Database.

5.121.2.3 virtual bool Arc::MySQLDatabase::enable_ssl (const std::string *keyfile* = "", const std::string *certfile* = "", const std::string *cafile* = "", const std::string *capath* = "") `[virtual]`

Enable ssl communication for the connection

Parameters:

keyfile The location of key file.

certfile The location of certificate file.

cafile The location of ca file.

capath The location of ca directory

Implements Arc::Database.

5.121.2.4 virtual bool Arc::MySQLDatabase::isconnected () const `[inline, virtual]`

Get the connection status

Implements Arc::Database.

5.121.2.5 virtual bool Arc::MySQLDatabase::shutdown () `[virtual]`

Ask database server to shutdown

Implements Arc::Database.

The documentation for this class was generated from the following file:

- MysqlWrapper.h

5.122 Arc::OAuthConsumer Class Reference

```
#include <OAuthConsumer.h>
```

Public Member Functions

- OAuthConsumer (const MCCConfig cfg, const URL url, std::list< std::string > idp_stack)
- MCC_Status parseDN (std::string ∗dn)
- MCC_Status approveCSR (const std::string approve_page)
- MCC_Status pushCSR (const std::string b64_pub_key, const std::string pub_key_hash, std::string ∗approve_page)
- MCC_Status storeCert (const std::string cert_path, const std::string auth_token, const std::string b64_dn)

Protected Member Functions

- MCC_Status processLogin (const std::string username="", const std::string password="")

5.122.1 Detailed Description

The OAuth functionality depends on the availability of the liboauth C-bindings library

5.122.2 Constructor & Destructor Documentation

5.122.2.1 Arc::OAuthConsumer::OAuthConsumer (const MCCConfig *cfg*, const URL *url*, std::list< std::string > *idp_stack*)

Construct an OAuth consumer with url as service provider. idp_name is currently ignored, since the idp to which the SAML2 redirect will take place is presently a hardcoded value on the SAML2 SP side. This is expected to change in the future.

5.122.3 Member Function Documentation

5.122.3.1 MCC_Status Arc::OAuthConsumer::approveCSR (const std::string *approve_page*)

Unsupported placeholder function until Confusa supports OAuth.

5.122.3.2 MCC_Status Arc::OAuthConsumer::parseDN (std::string ∗ *dn*)

Unsupported placeholder function until Confusa supports OAuth.

5.122.3.3 MCC_Status Arc::OAuthConsumer::processLogin (const std::string *username* = "", const std::string *password* = "") `[protected]`

Main function performing all the OAuth login steps. Username and password will be ignored.

5.122.3.4 MCC_Status Arc::OAuthConsumer::pushCSR (const std::string *b64_pub_key*, const std::string *pub_key_hash*, std::string $*$ *approve_page*)

Unsupported placeholder function until Confusa supports OAuth.

5.122.3.5 MCC_Status Arc::OAuthConsumer::storeCert (const std::string *cert_path*, const std::string *auth_token*, const std::string *b64_dn*)

Unsupported placeholder function until Confusa supports OAuth.

The documentation for this class was generated from the following file:

- OAuthConsumer.h

5.123 Arc::PathIterator Class Reference

Class to iterate through elements of path.

`#include <URL.h>`

Public Member Functions

- PathIterator (const std::string &path, bool end=false)
- PathIterator & operator++ ()
- PathIterator & operator– ()
- operator bool () const
- std::string operator $*$ () const
- std::string Rest () const

5.123.1 Detailed Description

Class to iterate through elements of path.

5.123.2 Constructor & Destructor Documentation

5.123.2.1 Arc::PathIterator::PathIterator (const std::string & *path*, bool *end* = `false`)

Constructor accepts path and stores it internally. If end is set to false iterator is pointing at first element in path. Otherwise selected element is one before last.

5.123.3 Member Function Documentation

5.123.3.1 std::string Arc::PathIterator::operator $*$ () const

Returns part of initial path from first till and including current

5.123.3.2 Arc::PathIterator::operator bool () const

Return false when iterator moved outside path elements

5.123.3.3 PathIterator& Arc::PathIterator::operator++ ()

Advances iterator to point at next path element

5.123.3.4 PathIterator& Arc::PathIterator::operator– ()

Moves iterator to element before current

5.123.3.5 std::string Arc::PathIterator::Rest () const

Returns part of initial path from one after current till end

The documentation for this class was generated from the following file:

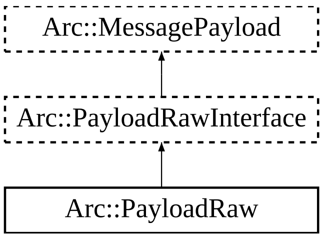
- URL.h

5.124 Arc::PayloadRaw Class Reference

Raw byte multi-buffer.

`#include <PayloadRaw.h>`

Inheritance diagram for Arc::PayloadRaw::



Public Member Functions

- PayloadRaw (void)
- virtual ∼PayloadRaw (void)
- virtual Size_t Size (void) const
- virtual char ∗ Buffer (unsigned int num=0)
- virtual Size_t BufferSize (unsigned int num=0) const
- virtual Size_t BufferPos (unsigned int num=0) const

5.124.1 Detailed Description

Raw byte multi-buffer.

This is implementation of PayloadRawInterface. Buffers are memory blocks logically placed one after another.

5.124.2 Constructor & Destructor Documentation

5.124.2.1 Arc::PayloadRaw::PayloadRaw (void) `[inline]`

Constructor. Created object contains no buffers.

5.124.2.2 virtual Arc::PayloadRaw::∼PayloadRaw (void) `[virtual]`

Destructor. Frees allocated buffers.

5.124.3 Member Function Documentation

5.124.3.1 virtual char∗ Arc::PayloadRaw::Buffer (unsigned int *num* = 0) `[virtual]`

Returns pointer to num'th buffer

Implements Arc::PayloadRawInterface.

5.124.3.2 virtual Size_t Arc::PayloadRaw::BufferPos (unsigned int *num* = 0) const [virtual]

Returns position of num'th buffer

Implements Arc::PayloadRawInterface.

5.124.3.3 virtual Size_t Arc::PayloadRaw::BufferSize (unsigned int *num* = 0) const [virtual]

Returns length of num'th buffer

Implements Arc::PayloadRawInterface.

5.124.3.4 virtual Size_t Arc::PayloadRaw::Size (void) const [virtual]

Returns logical size of whole structure.

Implements Arc::PayloadRawInterface.

The documentation for this class was generated from the following file:

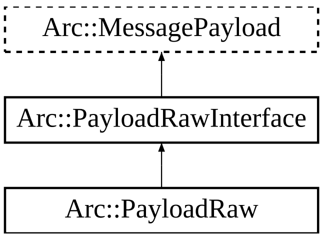
- PayloadRaw.h

5.125 Arc::PayloadRawInterface Class Reference

Random Access Payload for Message objects.

`#include <PayloadRaw.h>`

Inheritance diagram for Arc::PayloadRawInterface::



Public Member Functions

- virtual char operator[ ] (Size_t pos) const =0
- virtual char ∗ Content (Size_t pos=-1)=0
- virtual Size_t Size (void) const =0
- virtual char ∗ Insert (Size_t pos=0, Size_t size=0)=0
- virtual char ∗ Insert (const char ∗s, Size_t pos=0, Size_t size=-1)=0
- virtual char ∗ Buffer (unsigned int num)=0
- virtual Size_t BufferSize (unsigned int num) const =0
- virtual Size_t BufferPos (unsigned int num) const =0
- virtual bool Truncate (Size_t size)=0

5.125.1 Detailed Description

Random Access Payload for Message objects.

This class is a virtual interface for managing Message payload with arbitrarily accessible content. Inheriting classes are supposed to implement memory-resident or memory-mapped content made of optionally multiple chunks/buffers. Every buffer has own size and offset. This class is purely virtual.

5.125.2 Member Function Documentation

5.125.2.1 virtual char∗ Arc::PayloadRawInterface::Buffer (unsigned int *num*) `[pure virtual]`

Returns pointer to num'th buffer

Implemented in Arc::PayloadRaw.

5.125.2.2 virtual Size_t Arc::PayloadRawInterface::BufferPos (unsigned int *num*) const `[pure virtual]`

Returns position of num'th buffer

Implemented in Arc::PayloadRaw.

5.125.2.3 virtual Size_t Arc::PayloadRawInterface::BufferSize (unsigned int *num*) const `[pure virtual]`

Returns length of num'th buffer

Implemented in Arc::PayloadRaw.

5.125.2.4 virtual char* Arc::PayloadRawInterface::Content (Size_t *pos* = `-1`) `[pure virtual]`

Get pointer to buffer content at global position 'pos'. By default to beginning of main buffer whatever that means.

5.125.2.5 virtual char* Arc::PayloadRawInterface::Insert (const char * *s*, Size_t *pos* = `0`, Size_t *size* = `-1`) `[pure virtual]`

Create new buffer at global position 'pos' of size 'size'. Created buffer is filled with content of memory at 's'. If 'size' is negative content at 's' is expected to be null-terminated.

5.125.2.6 virtual char* Arc::PayloadRawInterface::Insert (Size_t *pos* = `0`, Size_t *size* = `0`) `[pure virtual]`

Create new buffer at global position 'pos' of size 'size'.

5.125.2.7]

virtual char Arc::PayloadRawInterface::operator[ ] (Size_t *pos*) const `[pure virtual]`

Returns content of byte at specified position. Specified position 'pos' is treated as global one and goes through all buffers placed one after another.

5.125.2.8 virtual Size_t Arc::PayloadRawInterface::Size (void) const `[pure virtual]`

Returns logical size of whole structure.

Implemented in Arc::PayloadRaw.

5.125.2.9 virtual bool Arc::PayloadRawInterface::Truncate (Size_t *size*) `[pure virtual]`

Change size of stored information. If size exceeds end of allocated buffer, buffers are not re-allocated, only logical size is extended. Buffers with location behind new size are deallocated.

The documentation for this class was generated from the following file:

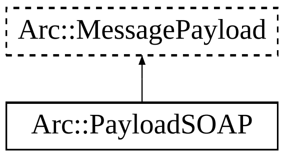
- PayloadRaw.h

5.126 Arc::PayloadSOAP Class Reference

Payload of Message with SOAP content.

```
#include <PayloadSOAP.h>
```

Inheritance diagram for Arc::PayloadSOAP::



Public Member Functions

- PayloadSOAP (const NS &ns, bool fault=false)
- PayloadSOAP (const SOAPEnvelope &soap)
- PayloadSOAP (const MessagePayload &source)

5.126.1 Detailed Description

Payload of Message with SOAP content.

This class combines MessagePayload with SOAPEnvelope to make it possible to pass SOAP messages through MCC chain.

5.126.2 Constructor & Destructor Documentation

5.126.2.1 Arc::PayloadSOAP::PayloadSOAP (const NS & *ns*, bool *fault* = `false`)

Constructor - creates new Message payload

5.126.2.2 Arc::PayloadSOAP::PayloadSOAP (const SOAPEnvelope & *soap*)

Constructor - creates Message payload from SOAP document. Provided SOAP document is copied to new object.

5.126.2.3 Arc::PayloadSOAP::PayloadSOAP (const MessagePayload & *source*)

Constructor - creates SOAP message from payload. PayloadRawInterface and derived classes are supported.

The documentation for this class was generated from the following file:

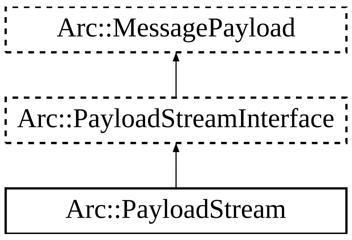
- PayloadSOAP.h

5.127 Arc::PayloadStream Class Reference

POSIX handle as Payload.

`#include <PayloadStream.h>`

Inheritance diagram for Arc::PayloadStream::



Public Member Functions

- PayloadStream (int h=-1)
- virtual ∼PayloadStream (void)
- virtual bool Get (char ∗buf, int &size)
- virtual bool Get (std::string &buf)
- virtual std::string Get (void)
- virtual bool Put (const std::string &buf)
- virtual bool Put (const char ∗buf)
- virtual operator bool (void)
- virtual bool operator! (void)
- virtual int Timeout (void) const
- virtual void Timeout (int to)
- virtual Size_t Pos (void) const
- virtual Size_t Size (void) const
- virtual Size_t Limit (void) const

Protected Attributes

- int handle_
- bool seekable_

5.127.1 Detailed Description

POSIX handle as Payload.

This is an implemetation of PayloadStreamInterface for generic POSIX handle.

5.127.2 Constructor & Destructor Documentation

5.127.2.1 Arc::PayloadStream::PayloadStream (int *h* = `-1`)

Constructor. Attaches to already open handle. Handle is not managed by this class and must be closed by external code.

5.127.2.2 virtual Arc::PayloadStream::∼PayloadStream (void) `[inline, virtual]`

Destructor.

5.127.3 Member Function Documentation

5.127.3.1 virtual std::string Arc::PayloadStream::Get (void) `[inline, virtual]`

Read as many as possible (sane amount) of bytes.

Implements Arc::PayloadStreamInterface.

5.127.3.2 virtual bool Arc::PayloadStream::Get (std::string & *buf*) `[virtual]`

Read as many as possible (sane amount) of bytes into buf.

Implements Arc::PayloadStreamInterface.

5.127.3.3 virtual bool Arc::PayloadStream::Get (char ∗ *buf*, int & *size*) `[virtual]`

Extracts information from stream up to 'size' bytes. 'size' contains number of read bytes on exit. Returns true in case of success.

Implements Arc::PayloadStreamInterface.

5.127.3.4 virtual Size_t Arc::PayloadStream::Limit (void) const `[inline, virtual]`

Returns position at which stream reading will stop if supported. That may be not same as Size() if instance is meant to provide access to only part of underlying obejct.

Implements Arc::PayloadStreamInterface.

5.127.3.5 virtual Arc::PayloadStream::operator bool (void) `[inline, virtual]`

Returns true if stream is valid.

Implements Arc::PayloadStreamInterface.

5.127.3.6 virtual bool Arc::PayloadStream::operator! (void) `[inline, virtual]`

Returns true if stream is invalid.

Implements Arc::PayloadStreamInterface.

5.127.3.7 virtual Size_t Arc::PayloadStream::Pos (void) const `[inline, virtual]`

Returns current position in stream if supported.

Implements Arc::PayloadStreamInterface.

5.127.3.8 virtual bool Arc::PayloadStream::Put (const char ∗ *buf*) [inline, virtual]

Push null terminated information from 'buf' into stream. Returns true on success.

Implements Arc::PayloadStreamInterface.

5.127.3.9 virtual bool Arc::PayloadStream::Put (const std::string & *buf*) [inline, virtual]

Push information from 'buf' into stream. Returns true on success.

Implements Arc::PayloadStreamInterface.

5.127.3.10 virtual Size_t Arc::PayloadStream::Size (void) const [inline, virtual]

Returns size of underlying object if supported.

Implements Arc::PayloadStreamInterface.

5.127.3.11 virtual void Arc::PayloadStream::Timeout (int *to*) [inline, virtual]

Set current timeout for Get() and Put() operations.

Implements Arc::PayloadStreamInterface.

5.127.3.12 virtual int Arc::PayloadStream::Timeout (void) const [inline, virtual]

Query current timeout for Get() and Put() operations.

Implements Arc::PayloadStreamInterface.

5.127.4 Field Documentation

5.127.4.1 int Arc::PayloadStream::handle_ [protected]

Timeout for read/write operations

5.127.4.2 bool Arc::PayloadStream::seekable_ [protected]

Handle for operations

The documentation for this class was generated from the following file:

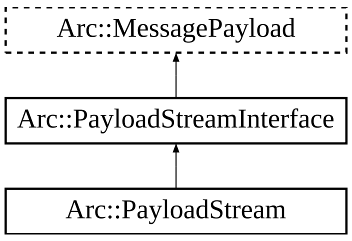
- PayloadStream.h

5.128 Arc::PayloadStreamInterface Class Reference

Stream-like Payload for Message object.

`#include <PayloadStream.h>`

Inheritance diagram for Arc::PayloadStreamInterface::



Public Member Functions

- virtual bool Get (char ∗buf, int &size)=0
- virtual bool Get (std::string &buf)=0
- virtual std::string Get (void)=0
- virtual bool Put (const char ∗buf, Size_t size)=0
- virtual bool Put (const std::string &buf)=0
- virtual bool Put (const char ∗buf)=0
- virtual operator bool (void)=0
- virtual bool operator! (void)=0
- virtual int Timeout (void) const =0
- virtual void Timeout (int to)=0
- virtual Size_t Pos (void) const =0
- virtual Size_t Size (void) const =0
- virtual Size_t Limit (void) const =0

5.128.1 Detailed Description

Stream-like Payload for Message object.

This class is a virtual interface for managing stream-like source and destination. It's supposed to be passed through MCC chain as payload of Message. It must be treated by MCCs and Services as dynamic payload. This class is purely virtual.

5.128.2 Member Function Documentation

5.128.2.1 virtual std::string Arc::PayloadStreamInterface::Get (void) `[pure virtual]`

Read as many as possible (sane amount) of bytes.

Implemented in Arc::PayloadStream.

5.128.2.2 virtual bool Arc::PayloadStreamInterface::Get (std::string & *buf*) `[pure virtual]`

Read as many as possible (sane amount) of bytes into buf.

Implemented in Arc::PayloadStream.

5.128.2.3 virtual bool Arc::PayloadStreamInterface::Get (char ∗ *buf*, int & *size*) `[pure virtual]`

Extracts information from stream up to 'size' bytes. 'size' contains number of read bytes on exit. Returns true in case of success.

Implemented in Arc::PayloadStream.

5.128.2.4 virtual Size_t Arc::PayloadStreamInterface::Limit (void) const `[pure virtual]`

Returns position at which stream reading will stop if supported. That may be not same as Size() if instance is meant to provide access to only part of underlying obejct.

Implemented in Arc::PayloadStream.

5.128.2.5 virtual Arc::PayloadStreamInterface::operator bool (void) `[pure virtual]`

Returns true if stream is valid.

Implemented in Arc::PayloadStream.

5.128.2.6 virtual bool Arc::PayloadStreamInterface::operator! (void) `[pure virtual]`

Returns true if stream is invalid.

Implemented in Arc::PayloadStream.

5.128.2.7 virtual Size_t Arc::PayloadStreamInterface::Pos (void) const `[pure virtual]`

Returns current position in stream if supported.

Implemented in Arc::PayloadStream.

5.128.2.8 virtual bool Arc::PayloadStreamInterface::Put (const char ∗ *buf*) `[pure virtual]`

Push null terminated information from 'buf' into stream. Returns true on success.

Implemented in Arc::PayloadStream.

5.128.2.9 virtual bool Arc::PayloadStreamInterface::Put (const std::string & *buf*) `[pure virtual]`

Push information from 'buf' into stream. Returns true on success.

Implemented in Arc::PayloadStream.

5.128.2.10 virtual bool Arc::PayloadStreamInterface::Put (const char ∗ *buf*, Size_t *size*) [pure virtual]

Push 'size' bytes from 'buf' into stream. Returns true on success.

5.128.2.11 virtual Size_t Arc::PayloadStreamInterface::Size (void) const [pure virtual]

Returns size of underlying object if supported.

Implemented in Arc::PayloadStream.

5.128.2.12 virtual void Arc::PayloadStreamInterface::Timeout (int *to*) [pure virtual]

Set current timeout for Get() and Put() operations.

Implemented in Arc::PayloadStream.

5.128.2.13 virtual int Arc::PayloadStreamInterface::Timeout (void) const [pure virtual]

Query current timeout for Get() and Put() operations.

Implemented in Arc::PayloadStream.

The documentation for this class was generated from the following file:

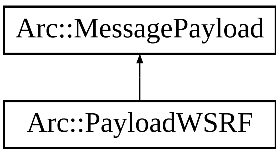
- PayloadStream.h

5.129 Arc::PayloadWSRF Class Reference

This class combines MessagePayload with WSRF.

`#include <PayloadWSRF.h>`

Inheritance diagram for Arc::PayloadWSRF::



Public Member Functions

- PayloadWSRF (const SOAPEnvelope &soap)
- PayloadWSRF (WSRF &wsrp)
- PayloadWSRF (const MessagePayload &source)

5.129.1 Detailed Description

This class combines MessagePayload with WSRF.

It's intention is to make it possible to pass WSRF messages through MCC chain as one more Payload type.

5.129.2 Constructor & Destructor Documentation

5.129.2.1 Arc::PayloadWSRF::PayloadWSRF (const SOAPEnvelope & *soap*)

Constructor - creates Message payload from SOAP message. Returns invalid WSRF if SOAP does not represent WS-ResourceProperties

5.129.2.2 Arc::PayloadWSRF::PayloadWSRF (WSRF & *wsrp*)

Constructor - creates Message payload with acquired WSRF message. WSRF message will be destroyed by destructor of this object.

5.129.2.3 Arc::PayloadWSRF::PayloadWSRF (const MessagePayload & *source*)

Constructor - creates WSRF message from payload. All classes derived from SOAPEnvelope are supported.

The documentation for this class was generated from the following file:

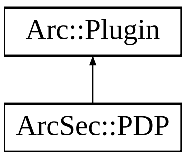
- PayloadWSRF.h

5.130 ArcSec::PDP Class Reference

Base class for Policy Decision Point plugins.

```
#include <PDP.h>
```

Inheritance diagram for ArcSec::PDP::



5.130.1 Detailed Description

Base class for Policy Decision Point plugins.

This virtual class defines method isPermitted() which processes security related information/attributes in Message and makes security decision - permit (true) or deny (false). Configuration of PDP is consumed during creation of instance through XML subtree fed to constructor.

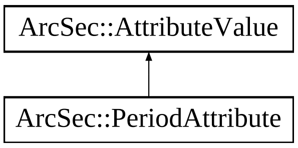
The documentation for this class was generated from the following file:

- PDP.h

5.131 ArcSec::PeriodAttribute Class Reference

`#include <DateTimeAttribute.h>`

Inheritance diagram for ArcSec::PeriodAttribute::



Public Member Functions

- virtual bool equal (AttributeValue ∗other, bool check_id=true)
- virtual std::string encode ()
- virtual std::string getType ()
- virtual std::string getId ()

5.131.1 Detailed Description

Formate: datetime"/"duration datetime"/"datetime duration"/"datetime

5.131.2 Member Function Documentation

5.131.2.1 virtual std::string ArcSec::PeriodAttribute::encode () `[virtual]`

encode the value in a string format

Implements ArcSec::AttributeValue.

5.131.2.2 virtual bool ArcSec::PeriodAttribute::equal (AttributeValue ∗ *other*, bool *check_id* = `true`) `[virtual]`

Evluate whether "this" equale to the parameter value

Implements ArcSec::AttributeValue.

5.131.2.3 virtual std::string ArcSec::PeriodAttribute::getId () `[inline, virtual]`

Get the AttributeId of the <Attribute>

Implements ArcSec::AttributeValue.

5.131.2.4 virtual std::string ArcSec::PeriodAttribute::getType () `[inline, virtual]`

Get the DataType of the <Attribute>

Implements ArcSec::AttributeValue.

The documentation for this class was generated from the following file:

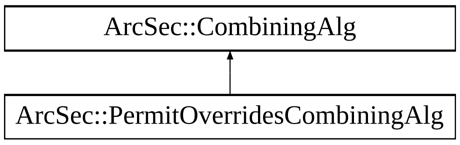
- DateTimeAttribute.h

5.132 ArcSec::PermitOverridesCombiningAlg Class Reference

Implement the "Permit-Overrides" algorithm.

`#include <PermitOverridesAlg.h>`

Inheritance diagram for ArcSec::PermitOverridesCombiningAlg::



Public Member Functions

- virtual Result combine (EvaluationCtx ∗ctx, std::list< Policy ∗ > policies)
- virtual const std::string & getalgId (void) const

5.132.1 Detailed Description

Implement the "Permit-Overrides" algorithm.

Permit-Overrides, scans the policy set which is given as the parameters of "combine" method, if gets "permit" result from any policy, then stops scanning and gives "permit" as result, otherwise gives "deny".

5.132.2 Member Function Documentation

5.132.2.1 virtual Result ArcSec::PermitOverridesCombiningAlg::combine (EvaluationCtx ∗ *ctx*, std::list< Policy ∗ > *policies*) `[virtual]`

If there is one policy which return positive evaluation result, then omit the other policies and return DECISION_PERMIT

Parameters:

ctx This object contains request information which will be used to evaluated against policy.

policlies This is a container which contains policy objects.

Returns:

The combined result according to the algorithm.

Implements ArcSec::CombiningAlg.

5.132.2.2 virtual const std::string& ArcSec::PermitOverridesCombiningAlg::getalgId (void) const `[inline, virtual]`

Get the identifier

Implements ArcSec::CombiningAlg.

The documentation for this class was generated from the following file:

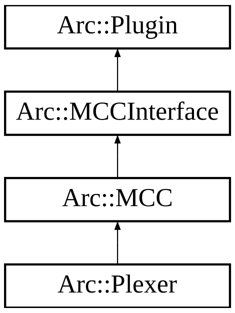
- PermitOverridesAlg.h

5.133 Arc::Plexer Class Reference

The Plexer class, used for routing messages to services.

`#include <Plexer.h>`

Inheritance diagram for Arc::Plexer::



Public Member Functions

- Plexer (Config ∗cfg)
- virtual ∼Plexer ()
- virtual void Next (MCCInterface ∗next, const std::string &label)
- virtual MCC_Status process (Message &request, Message &response)

Static Public Attributes

- static Logger logger

5.133.1 Detailed Description

The Plexer class, used for routing messages to services.

This is the Plexer class. Its purpose is to route incoming messages to appropriate Services and MCC chains.

5.133.2 Constructor & Destructor Documentation

5.133.2.1 Arc::Plexer::Plexer (Config ∗ *cfg*)

The constructor.

This is the constructor. Since all member variables are instances of "well-behaving" STL classes, nothing needs to be done.

5.133.2.2 virtual Arc::Plexer::∼Plexer () `[virtual]`

The destructor.

This is the destructor. Since all member variables are instances of "well-behaving" STL classes, nothing needs to be done.

5.133.3 Member Function Documentation

5.133.3.1 virtual void Arc::Plexer::Next (MCCInterface $*$ *next*, const std::string & *label*) `[virtual]`

Add reference to next MCC in chain.

This method is called by Loader for every potentially labeled link to next component which implements MCCInterface. If next is set NULL corresponding link is removed.

Reimplemented from Arc::MCC.

5.133.3.2 virtual MCC_Status Arc::Plexer::process (Message & *request*, Message & *response*) `[virtual]`

Route request messages to appropriate services.

Routes the request message to the appropriate service. Routing is based on the path part of value of the ENDPOINT attribute. Routed message is assigned following attributes: PLEXER:PATTERN - matched pattern, PLEXER:EXTENSION - last unmatched part of ENDPOINT path.

Reimplemented from Arc::MCC.

5.133.4 Field Documentation

5.133.4.1 Logger Arc::Plexer::logger `[static]`

A logger for MCCs.

A logger intended to be the parent of loggers in the different MCCs.

Reimplemented from Arc::MCC.

The documentation for this class was generated from the following file:

- Plexer.h

5.134 Arc::PlexerEntry Class Reference

A pair of label (regex) and pointer to MCC.

```
#include <Plexer.h>
```

5.134.1 Detailed Description

A pair of label (regex) and pointer to MCC.

A helper class that stores a label (regex) and a pointer to a service.

The documentation for this class was generated from the following file:

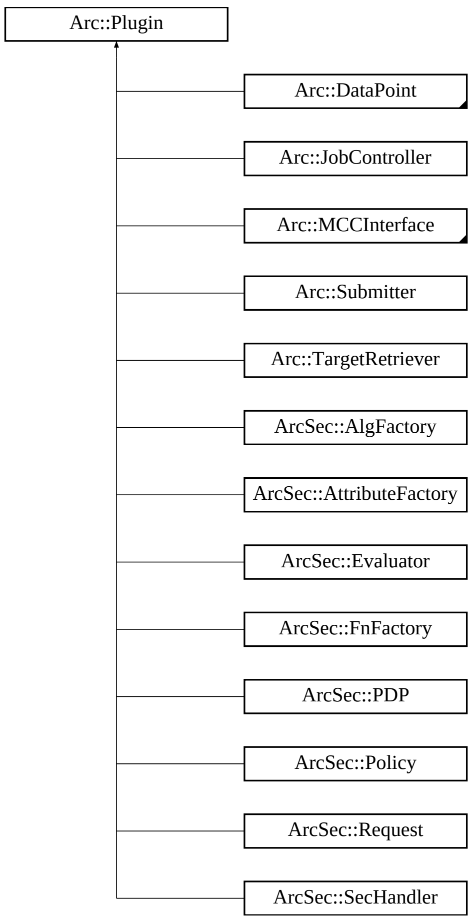
- Plexer.h

5.135 Arc::Plugin Class Reference

Base class for loadable ARC components.

`#include <Plugin.h>`

Inheritance diagram for Arc::Plugin::



5.135.1 Detailed Description

Base class for loadable ARC components.

All classes representing loadable ARC components must be either descendants of this class or be wrapped by its offspring.

The documentation for this class was generated from the following file:

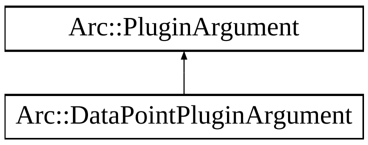
- Plugin.h

5.136 Arc::PluginArgument Class Reference

Base class for passing arguments to loadable ARC components.

`#include <Plugin.h>`

Inheritance diagram for Arc::PluginArgument::



Public Member Functions

- PluginsFactory ∗ get_factory (void)
- Glib::Module ∗ get_module (void)

5.136.1 Detailed Description

Base class for passing arguments to loadable ARC components.

During its creation constructor function of ARC loadable component expects instance of class inherited from this one or wrapped in it. Then dynamic type casting is used for obtaining class of expected kind.

5.136.2 Member Function Documentation

5.136.2.1 PluginsFactory∗ Arc::PluginArgument::get_factory (void)

Returns pointer to factory which instantiated plugin.

Because factory usually destroys/unloads plugins in its destructor it should be safe to keep this pointer inside plugin for later use. But one must always check.

5.136.2.2 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

5.137 Arc::PluginDesc Class Reference

Description of plugin.

```
#include <Plugin.h>
```

5.137.1 Detailed Description

Description of plugin.

This class is used for reports

The documentation for this class was generated from the following file:

- Plugin.h

5.138 Arc::PluginDescriptor Struct Reference

Description of ARC lodable component.

```
#include <Plugin.h>
```

5.138.1 Detailed Description

Description of ARC lodable component.

The documentation for this struct was generated from the following file:

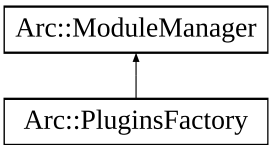
- Plugin.h

5.139 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)

5.139.1 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.

5.139.2 Constructor & Destructor Documentation

5.139.2.1 Arc::PluginsFactory::PluginsFactory (XMLNode *cfg*)

Constructor - accepts configuration (not yet used) meant to tune loading of modules.

5.139.3 Member Function Documentation

5.139.3.1 static void Arc::PluginsFactory::FilterByKind (const std::string & *kind*, std::list< ModuleDesc > & *descs*) `[static]`

Filter list of modules by kind.

5.139.3.2 bool Arc::PluginsFactory::load (const std::string & *name*)

These methods load module named lib'name' and check if it contains ARC plugin(s) of specified 'kind' and 'name'. If there are no specified plugins or module does not contain any ARC plugins it is unloaded. All loaded plugins are also registered in internal list of this instance of PluginsFactory class. Returns true if any plugin was loaded.

5.139.3.3 void Arc::PluginsFactory::report (std::list< ModuleDesc > & *descs*)

Provides information about currently loaded modules and plugins.

5.139.3.4 bool Arc::PluginsFactory::scan (const std::string & *name*, ModuleDesc & *desc*)

Collect information about plugins stored in module(s) with specified names. Returns true if any of specified modules has plugins.

5.139.3.5 void Arc::PluginsFactory::TryLoad (bool *v* = `true`) `[inline]`

Specifies if loadable module may be loaded while looking for analyzing its content. If set to false only ∗.apd files are checked. Modules without corresponding ∗.apd will be ignored. Default is true;

The documentation for this class was generated from the following file:

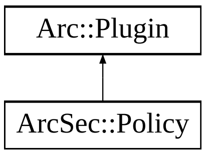
- Plugin.h

5.140 ArcSec::Policy Class Reference

Interface for containing and processing different types of policy.

`#include <Policy.h>`

Inheritance diagram for ArcSec::Policy::



Public Member Functions

- Policy ()
- Policy (const Arc::XMLNode)
- Policy (const Arc::XMLNode, EvaluatorContext ∗)
- virtual operator bool (void) const =0
- virtual MatchResult match (EvaluationCtx ∗)=0
- virtual Result eval (EvaluationCtx ∗)=0
- virtual void addPolicy (Policy ∗pl)
- virtual void setEvaluatorContext (EvaluatorContext ∗)
- virtual void make_policy ()
- virtual std::string getEffect () const =0
- virtual EvalResult & getEvalResult ()=0
- virtual void setEvalResult (EvalResult &res)=0
- virtual const char ∗ getEvalName () const =0
- virtual const char ∗ getName () const =0

5.140.1 Detailed Description

Interface for containing and processing different types of policy.

Basically, each policy object is a container which includes a few elements e.g., ArcPolicySet objects includes a few ArcPolicy objects; ArcPolicy object includes a few ArcRule objects. There is logical relationship between ArcRules or ArcPolicies, which is called combining algorithm. According to algorithm, evaluation results from the elements are combined, and then the combined evaluation result is returned to the up-level.

5.140.2 Constructor & Destructor Documentation

5.140.2.1 ArcSec::Policy::Policy () `[inline]`

Template constructor - creates empty policy.

5.140.2.2 ArcSec::Policy::Policy (const Arc::XMLNode) `[inline]`

Template constructor - creates policy based on XML document.

If XML document is empty then empty policy is created. If it is not empty then it must be valid policy document - otherwise created object should be invalid.

5.140.2.3 ArcSec::Policy::Policy (const Arc::XMLNode, EvaluatorContext ∗) `[inline]`

Template constructor - creates policy based on XML document.

If XML document is empty then empty policy is created. If it is not empty then it must be valid policy document - otherwise created object should be invalid. This constructor is based on the policy node and i the EvaluatorContext which includes the factory objects for combining algorithm and function

5.140.3 Member Function Documentation

5.140.3.1 virtual void ArcSec::Policy::addPolicy (Policy ∗ *pl*) `[inline, virtual]`

Add a policy element to into "this" object

5.140.3.2 virtual Result ArcSec::Policy::eval (EvaluationCtx ∗) `[pure virtual]`

Evaluate policy For the <Rule> of Arc, only get the "Effect" from rules; For the <Policy> of Arc, combine the evaluation result from <Rule>; For the <Rule> of XACML, evaluate the <Condition> node by using information from request, and use the "Effect" attribute of <Rule>; For the <Policy> of XACML, combine the evaluation result from <Rule>

5.140.3.3 virtual std::string ArcSec::Policy::getEffect () const `[pure virtual]`

Get the "Effect" attribute

5.140.3.4 virtual const char∗ ArcSec::Policy::getEvalName () const `[pure virtual]`

Get the name of Evaluator which can evaluate this policy

5.140.3.5 virtual EvalResult& ArcSec::Policy::getEvalResult () `[pure virtual]`

Get eveluation result

5.140.3.6 virtual const char∗ ArcSec::Policy::getName () const `[pure virtual]`

Get the name of this policy

5.140.3.7 virtual void ArcSec::Policy::make_policy () `[inline, virtual]`

Parse XMLNode, and construct the low-level Rule object

5.140.3.8 virtual MatchResult ArcSec::Policy::match (EvaluationCtx ∗) `[pure virtual]`

Evaluate whether the two targets to be evaluated match to each other.

5.140.3.9 virtual ArcSec::Policy::operator bool (void) const `[pure virtual]`

Returns true is object is valid.

5.140.3.10 virtual void ArcSec::Policy::setEvalResult (EvalResult & *res*) `[pure virtual]`

Set eveluation result

5.140.3.11 virtual void ArcSec::Policy::setEvaluatorContext (EvaluatorContext ∗) `[inline, virtual]`

Set Evaluator Context for the usage in creating low-level policy object

The documentation for this class was generated from the following file:

- Policy.h

5.141 ArcSec::PolicyParser Class Reference

A interface which will isolate the policy object from actual policy storage (files, urls, database).

`#include <PolicyParser.h>`

Public Member Functions

- virtual Policy ∗ parsePolicy (const Source &source, std::string policyclassname, EvaluatorContext ∗ctx)

5.141.1 Detailed Description

A interface which will isolate the policy object from actual policy storage (files, urls, database).

Parse the policy from policy source (e.g. files, urls, database, etc.).

5.141.2 Member Function Documentation

5.141.2.1 virtual Policy∗ ArcSec::PolicyParser::parsePolicy (const Source & *source*, std::string *policyclassname*, EvaluatorContext ∗ *ctx*) `[virtual]`

Parse policy

Parameters:

source location of the policy

policyclassname name of the policy for ClassLoader

ctx EvaluatorContext which includes the ∗∗Factory

The documentation for this class was generated from the following file:

- PolicyParser.h

5.142 ArcSec::PolicyStore Class Reference

Storage place for policy objects.

```
#include <PolicyStore.h>
```

Public Member Functions

- PolicyStore (const std::string &alg, const std::string &policyclassname, EvaluatorContext ∗ctx)

Data Structures

- class **PolicyElement**

5.142.1 Detailed Description

Storage place for policy objects.

5.142.2 Constructor & Destructor Documentation

5.142.2.1 ArcSec::PolicyStore::PolicyStore (const std::string & *alg*, const std::string & *policyclassname*, EvaluatorContext ∗ *ctx*)

Creates policy store with specified combing algorithm (alg - not used yet), policy name (policyclassname) and context (ctx)

The documentation for this class was generated from the following file:

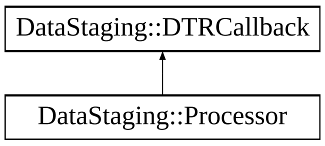
- PolicyStore.h

5.143 DataStaging::Processor Class Reference

The Processor performs pre- and post-transfer operations.

```
#include <Processor.h>
```

Inheritance diagram for DataStaging::Processor::



Public Member Functions

- Processor ()
- ∼Processor ()
- void start (void)
- void stop (void)
- virtual void receiveDTR (DTR &dtr)

Data Structures

- class **ThreadArgument**

  Class used to pass information to spawned thread.

5.143.1 Detailed Description

The Processor performs pre- and post-transfer operations.

The Processor takes care of everything that should happen before and after a transfer takes place. Calling receiveDTR() spawns a thread to perform the required operation depending on the DTR state.

5.143.2 Constructor & Destructor Documentation

5.143.2.1 DataStaging::Processor::Processor () `[inline]`

Constructor.

5.143.2.2 DataStaging::Processor::∼Processor () `[inline]`

Destructor waits for all active threads to stop.

5.143.3 Member Function Documentation

5.143.3.1 virtual void DataStaging::Processor::receiveDTR (DTR & *dtr*) `[virtual]`

Send a DTR to the Processor.

The DTR is sent to the Processor through this method when some long-latency processing is to be performed, eg contacting a remote service. The Processor spawns a thread to do the processing, and then returns. The thread notifies the scheduler when it is finished.

Implements DataStaging::DTRCallback.

5.143.3.2 void DataStaging::Processor::start (void)

Start Processor.

This method actually does nothing. It is here only to make all classes of data staging to look alike. But it is better to call it before starting to use object because it may do something in the future.

5.143.3.3 void DataStaging::Processor::stop (void)

Stop Processor.

This method sends waits for all started threads to end and exits. Since threads a short-lived it is better to wait rather than interrupt them.

The documentation for this class was generated from the following file:

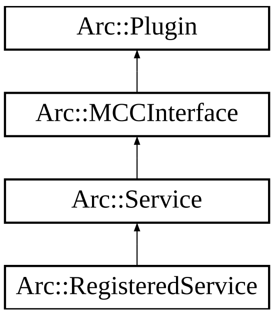
- Processor.h

5.144 Arc::RegisteredService Class Reference

RegisteredService - extension of Service performing self-registration.

`#include <RegisteredService.h>`

Inheritance diagram for Arc::RegisteredService::



Public Member Functions

- RegisteredService (Config ∗)

5.144.1 Detailed Description

RegisteredService - extension of Service performing self-registration.

5.144.2 Constructor & Destructor Documentation

5.144.2.1 Arc::RegisteredService::RegisteredService (Config ∗)

Example contructor - Server takes at least it's configuration subtree

The documentation for this class was generated from the following file:

- RegisteredService.h

5.145 Arc::RegularExpression Class Reference

A regular expression class.

`#include <ArcRegex.h>`

Public Member Functions

- RegularExpression ()
- RegularExpression (std::string pattern)
- RegularExpression (const RegularExpression &regex)
- ∼RegularExpression ()
- const RegularExpression & operator= (const RegularExpression &regex)
- bool isOk ()
- bool hasPattern (std::string str)
- bool match (const std::string &str) const
- bool match (const std::string &str, std::list< std::string > &unmatched, std::list< std::string > &matched) const
- std::string getPattern () const

5.145.1 Detailed Description

A regular expression class.

This class is a wrapper around the functions provided in regex.h.

5.145.2 Constructor & Destructor Documentation

5.145.2.1 Arc::RegularExpression::RegularExpression () `[inline]`

default constructor

5.145.2.2 Arc::RegularExpression::RegularExpression (std::string *pattern*)

Creates a reges from a pattern string.

5.145.2.3 Arc::RegularExpression::RegularExpression (const RegularExpression & *regex*)

Copy constructor.

5.145.2.4 Arc::RegularExpression::∼RegularExpression ()

Destructor.

5.145.3 Member Function Documentation

5.145.3.1 std::string Arc::RegularExpression::getPattern () const

Returns pattern.

5.145.3.2 bool Arc::RegularExpression::hasPattern (std::string *str*)

Returns true if this regex has the pattern provided.

5.145.3.3 bool Arc::RegularExpression::isOk ()

Returns true if the pattern of this regex is ok.

5.145.3.4 bool Arc::RegularExpression::match (const std::string & *str*, std::list< std::string > & *unmatched*, std::list< std::string > & *matched*) const

Returns true if this regex matches the string provided.

Unmatched parts of the string are stored in 'unmatched'. Matched parts of the string are stored in 'matched'. The first entry in matched is the string that matched the regex, and the following entries are parenthesised elements of the regex

5.145.3.5 bool Arc::RegularExpression::match (const std::string & *str*) const

Returns true if this regex matches whole string provided.

5.145.3.6 const RegularExpression& Arc::RegularExpression::operator= (const RegularExpression & *regex*)

Assignment operator.

The documentation for this class was generated from the following file:

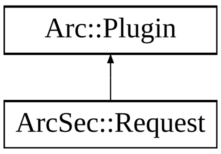
- ArcRegex.h

5.146 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 &)

5.146.1 Detailed Description

Base class/Interface for request, includes a container for RequestItems and some operations.

A Request object can has a few <subjects, actions, objects> tuples, i.e. RequestItem The Request class and any customized class which inherit from it, should be loadable, which means these classes can be dynamically loaded according to the configuration informtation, see the example configuration below: <Service name="pdp.service" id="pdp_service"> <pdp:PDPConfig> <......> <pdp:Request name="arc.request" /> <......> </pdp:PDPConfig> </Service>

There can be different types of subclass which inherit Request, such like XACMLRequest, ArcRequest, GACLRequest

5.146.2 Constructor & Destructor Documentation

5.146.2.1 ArcSec::Request::Request () `[inline]`

Default constructor

5.146.2.2 ArcSec::Request::Request (const Source &) `[inline]`

Constructor: Parse request information from a xml stucture in memory

5.146.3 Member Function Documentation

5.146.3.1 virtual void ArcSec::Request::addRequestItem (Attrs &, Attrs &, Attrs &, Attrs &) `[inline, virtual]`

Add request tuple from non-XMLNode

5.146.3.2 virtual const char∗ ArcSec::Request::getEvalName () const `[pure virtual]`

Get the name of corresponding evaulator

5.146.3.3 virtual const char∗ ArcSec::Request::getName () const `[pure virtual]`

Get the name of this request

5.146.3.4 virtual ReqItemList ArcSec::Request::getRequestItems () const `[inline, virtual]`

Get all the RequestItem inside RequestItem container

5.146.3.5 virtual void ArcSec::Request::make_request () `[pure virtual]`

Create the objects included in Request according to the node attached to the Request object

5.146.3.6 virtual void ArcSec::Request::setAttributeFactory (AttributeFactory ∗ *attributefactory*) `[pure virtual]`

Set the attribute factory for the usage of Request

5.146.3.7 virtual void ArcSec::Request::setRequestItems (ReqItemList) `[inline, virtual]`

Set the content of the container

The documentation for this class was generated from the following file:

- Request.h

5.147 ArcSec::RequestAttribute Class Reference

Wrapper which includes AttributeValue object which is generated according to date type of one spefic node in Request.xml.

```
#include <RequestAttribute.h>
```

Public Member Functions

- RequestAttribute (Arc::XMLNode &node, AttributeFactory ∗attrfactory)
- RequestAttribute & duplicate (RequestAttribute &)

5.147.1 Detailed Description

Wrapper which includes AttributeValue object which is generated according to date type of one spefic node in Request.xml.

5.147.2 Constructor & Destructor Documentation

5.147.2.1 ArcSec::RequestAttribute::RequestAttribute (Arc::XMLNode & *node*, AttributeFactory ∗ *attrfactory*)

Constructor - create attribute value object according to the "Type" in the node <Attribute attributeid="urn:arc:subject:voms-attribute" type="string">urn:mace:shibboleth:examples</Attribute>

5.147.3 Member Function Documentation

5.147.3.1 RequestAttribute& ArcSec::RequestAttribute::duplicate (RequestAttribute &)

Duplicate the parameter into "this"

The documentation for this class was generated from the following file:

- RequestAttribute.h

5.148 ArcSec::RequestItem Class Reference

Interface for request item container, <subjects, actions, objects, ctxs> tuple.

`#include <RequestItem.h>`

Public Member Functions

- RequestItem (Arc::XMLNode &, AttributeFactory ∗)

5.148.1 Detailed Description

Interface for request item container, <subjects, actions, objects, ctxs> tuple.

5.148.2 Constructor & Destructor Documentation

5.148.2.1 ArcSec::RequestItem::RequestItem (Arc::XMLNode &, AttributeFactory ∗) `[inline]`

Constructor

Parameters:

node The XMLNode structure of the request item

attributefactory The AttributeFactory which will be used to generate RequestAttribute

The documentation for this class was generated from the following file:

- RequestItem.h

5.149 ArcSec::Response Class Reference

Container for the evaluation results.

```
#include <Response.h>
```

5.149.1 Detailed Description

Container for the evaluation results.

The documentation for this class was generated from the following file:

- Response.h

5.150 ArcSec::ResponseItem Class Reference

Evaluation result concerning one RequestTuple.

```
#include <Response.h>
```

5.150.1 Detailed Description

Evaluation result concerning one RequestTuple.

Include the RequestTuple, related XMLNode, the set of policy objects which give positive evaluation result, and the related XMLNode

The documentation for this class was generated from the following file:

- Response.h

5.151 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)

5.151.1 Detailed Description

This class runs external executable. It is possible to read/write it's standard handles or to redirect then to std::string elements.

5.151.2 Constructor & Destructor Documentation

5.151.2.1 Arc::Run::Run (const std::string & *cmdline*)

Constructor preapres object to run cmdline

5.151.2.2 Arc::Run::Run (const std::list< std::string > & *argv*)

Constructor preapres object to run executable and arguments specified in argv

5.151.2.3 Arc::Run::∼Run (void)

Destructor kills running executable and releases associated resources

5.151.3 Member Function Documentation

5.151.3.1 void Arc::Run::Abandon (void)

Detach this object from running process. After calling this method instance is not associated with external process anymore. As result destructor will not kill process.

5.151.3.2 static void Arc::Run::AfterFork (void) `[static]`

Call this method after fork() in child cporocess. It will reinitialize internal structures for new environment. Do not call it in any other case than defined.

5.151.3.3 void Arc::Run::AssignStderr (std::string & *str*)

Associate stderr handle of executable with string. This method must be called before Start(). str object must be valid as long as this object exists.

5.151.3.4 void Arc::Run::AssignStdin (std::string & *str*)

Associate stdin handle of executable with string. This method must be called before Start(). str object must be valid as long as this object exists.

5.151.3.5 void Arc::Run::AssignStdout (std::string & *str*)

Associate stdout handle of executable with string. This method must be called before Start(). str object must be valid as long as this object exists.

5.151.3.6 void Arc::Run::AssignWorkingDirectory (std::string & *wd*) `[inline]`

Assign working direcotry of the running process

5.151.3.7 void Arc::Run::CloseStderr (void)

Closes pipe associated with stderr handle

5.151.3.8 void Arc::Run::CloseStdin (void)

Closes pipe associated with stdin handle

5.151.3.9 void Arc::Run::CloseStdout (void)

Closes pipe associated with stdout handle

5.151.3.10 void Arc::Run::KeepStderr (bool *keep* = `true`)

Keep stderr same as parent's if keep = true

5.151.3.11 void Arc::Run::KeepStdin (bool *keep* = `true`)

Keep stdin same as parent's if keep = true

5.151.3.12 void Arc::Run::KeepStdout (bool *keep* = `true`)

Keep stdout same as parent's if keep = true

5.151.3.13 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

5.151.3.14 Arc::Run::operator bool (void) `[inline]`

Returns true if object is valid

5.151.3.15 bool Arc::Run::operator! (void) `[inline]`

Returns true if object is invalid

5.151.3.16 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.

5.151.3.17 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.

5.151.3.18 int Arc::Run::Result (void) `[inline]`

Returns exit code of execution.

5.151.3.19 bool Arc::Run::Running (void)

Return true if execution is going on.

5.151.3.20 bool Arc::Run::Start (void)

Starts running executable. This method may be called only once.

5.151.3.21 bool Arc::Run::Wait (void)

Wait till execution finished

5.151.3.22 bool Arc::Run::Wait (int *timeout*)

Wait till execution finished or till timeout seconds expires. Returns true if execution is complete.

5.151.3.23 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

5.152 Arc::SAMLToken Class Reference

Class for manipulating SAML Token Profile.

```
#include <SAMLToken.h>
```

Public Types

- enum SAMLVersion

Public Member Functions

- SAMLToken (SOAPEnvelope &soap)
- SAMLToken (SOAPEnvelope &soap, const std::string &certfile, const std::string &keyfile, SAMLVersion saml_version=SAML2, XMLNode saml_assertion=XMLNode())
- ~SAMLToken (void)
- operator bool (void)
- bool Authenticate (const std::string &cafile, const std::string &capath)
- bool Authenticate (void)

5.152.1 Detailed Description

Class for manipulating SAML Token Profile.

This class is for generating/consuming SAML Token profile. See WS-Security SAML Token Profile v1.1 (www.oasis-open.org/committees/wss) Currently this class is used by samltoken handler (will appears in src/hed/pdc/samltokensh/) It is not a must to directly called this class. If we need to use SAML Token functionality, we only need to configure the samltoken handler into service and client. Currently, only a minor part of the specification has been implemented.

About how to identify and reference security token for signing message, currently, only the "SAML Assertion Referenced from KeyInfo" (part 3.4.2 of WS-Security SAML Token Profile v1.1 specification) is supported, which means the implementation can only process SAML assertion "referenced from KeyInfo", and also can only generate SAML Token with SAML assertion "referenced from KeyInfo". More complete support need to implement.

About subject confirmation method, the implementation can process "hold-of-key" (part 3.5.1 of WS-Security SAML Token Profile v1.1 specification) subject subject confirmation method.

About SAML vertion, the implementation can process SAML assertion with SAML version 1.1 and 2.0; can only generate SAML assertion with SAML vertion 2.0.

In the SAML Token profile, for the hold-of-key subject confirmation method, there are three interaction parts: the attesting entity, the relying party and the issuing authority. In the hold-of-key subject confirmation method, it is the attesting entity's subject identity which will be inserted into the SAML assertion.

Firstly the attesting entity authenticates to issuing authority by using some authentication scheme such as WSS x509 Token profile (Alterbatively the usename/password authentication scheme or other different authentication scheme can also be used, unless the issuing authority can retrive the key from a trusted certificate server after firmly establishing the subject's identity under the username/password scheme). So then issuing authority is able to make a definitive statement (sign a SAML assertion) about an act of authentication that has already taken place.

The attesting entity gets the SAML assertion and then signs the soap message together with the assertion by using its private key (the relevant certificate has been authenticated by issuing authority, and its relevant public key has been put into SubjectConfirmation element under saml assertion by issuing authority. Only the actual owner of the saml assertion can do this, as only the subject possesses the private key paired with the public key in the assertion. This establishes an irrefutable connection between the author of the SOAP message and the assertion describing an authentication event.)

The relying party is supposed to trust the issuing authority. When it receives a message from the asserting entity, it will check the saml assertion based on its predetermined trust relationship with the SAML issuing authority, and check the signature of the soap message based on the public key in the saml assertion without directly trust relationship with attesting entity (subject owner).

5.152.2 Member Enumeration Documentation

5.152.2.1 enum Arc::SAMLToken::SAMLVersion

Since the specfication SAMLVersion is for distinguishing two types of saml version. It is used as the parameter of constructor.

5.152.3 Constructor & Destructor Documentation

5.152.3.1 Arc::SAMLToken::SAMLToken (SOAPEnvelope & *soap*)

Constructor. Parse SAML Token information from SOAP header. SAML Token related information is extracted from SOAP header and stored in class variables. And then it the SAMLToken object will be used for authentication.

Parameters:

soap The SOAP message which contains the SAMLToken in the soap header

5.152.3.2 Arc::SAMLToken::SAMLToken (SOAPEnvelope & *soap*, const std::string & *certfile*, const std::string & *keyfile*, SAMLVersion *saml_version* = `SAML2`, XMLNode *saml_assertion* = `XMLNode()`)

Constructor. Add SAML Token information into the SOAP header. Generated token contains elements SAML token and signature, and is meant to be used for authentication on the consuming side. This constructor is for a specific SAML Token profile usage, in which the attesting entity signs the SAML assertion for itself (self-sign). This usage implicitly requires that the relying party trust the attesting entity. More general (requires issuing authority) usage will be provided by other constructor. And the under-developing SAML service will be used as the issuing authority.

Parameters:

soap The SOAP message to which the SAML Token will be inserted.

certfile The certificate file.

keyfile The key file which will be used to create signature.

samlversion The SAML version, only SAML2 is supported currently.

samlassertion The SAML assertion got from 3rd party, and used for protecting the SOAP message; If not present, then self-signed assertion will be generated.

5.152.3.3 Arc::SAMLToken::∼SAMLToken (void)

Deconstructor. Nothing to be done except finalizing the xmlsec library.

5.152.4 Member Function Documentation

5.152.4.1 bool Arc::SAMLToken::Authenticate (void)

Check signature by using the cert information in soap message

5.152.4.2 bool Arc::SAMLToken::Authenticate (const std::string & *cafile*, const std::string & *capath*)

Check signature by using the trusted certificates It is used by relying parting after calling SAMLToken(SOAPEnvelope& soap) This method will check the SAML assertion based on the trusted certificated specified as parameter cafile or capath; and also check the signature to soap message (the signature is generated by attesting entity by signing soap body together witl SAML assertion) by using the public key inside SAML assetion.

Parameters:

cafile	ca file
capath	ca directory

5.152.4.3 Arc::SAMLToken::operator bool (void)

Returns true of constructor succeeded

The documentation for this class was generated from the following file:

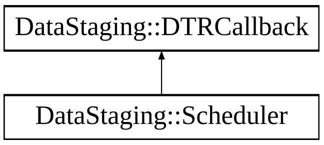
- SAMLToken.h

5.153 DataStaging::Scheduler Class Reference

The Scheduler is the control centre of the data staging framework.

`#include <Scheduler.h>`

Inheritance diagram for DataStaging::Scheduler::



Public Member Functions

- Scheduler ()
- ~Scheduler ()
- void SetSlots (int pre_processor=0, int post_processor=0, int delivery=0, int delivery_emergency=0)
- void AddURLMapping (const Arc::URL &template_url, const Arc::URL &replacement_url, const Arc::URL &access_url=Arc::URL())
- void SetURLMapping (const Arc::URLMap &mapping=Arc::URLMap())
- void SetPreferredPattern (const std::string &pattern)
- void SetTransferShares (const TransferShares &shares)
- void AddSharePriority (const std::string &name, int priority)
- void SetSharePriorities (const std::map< std::string, int > &shares)
- void SetShareType (TransferShares::ShareType share_type)
- void SetTransferParameters (const TransferParameters &params)
- void SetDumpLocation (const std::string &location)
- bool start (void)
- virtual void receiveDTR (DTR &dtr)
- bool cancelDTRs (const std::string &jobid)
- bool stop ()

5.153.1 Detailed Description

The Scheduler is the control centre of the data staging framework.

The Scheduler manages a global list of DTRs and schedules when they should go into the next state or be sent to other processes. The DTR priority is used to decide each DTR's position in a queue.

5.153.2 Constructor & Destructor Documentation

5.153.2.1 DataStaging::Scheduler::Scheduler ()

Constructor.

5.153.2.2 DataStaging::Scheduler::~Scheduler () `[inline]`

Destructor calls stop(), which cancels all DTRs and waits for them to complete.

5.153.3 Member Function Documentation

5.153.3.1 void DataStaging::Scheduler::AddSharePriority (const std::string & *name*, int *priority*)

Add share.

5.153.3.2 void DataStaging::Scheduler::AddURLMapping (const Arc::URL & *template_url*, const Arc::URL & *replacement_url*, const Arc::URL & *access_url* = `Arc::URL()`)

Add URL mapping entry.

5.153.3.3 bool DataStaging::Scheduler::cancelDTRs (const std::string & *jobid*)

Tell the Scheduler to cancel all the DTRs in the given job description.

5.153.3.4 virtual void DataStaging::Scheduler::receiveDTR (DTR & *dtr*) `[virtual]`

Callback method implemented from DTRCallback.

This method is called by the generator when it wants to pass a DTR to the scheduler.

Implements DataStaging::DTRCallback.

5.153.3.5 void DataStaging::Scheduler::SetDumpLocation (const std::string & *location*)

Set location for periodic dump of DTR state (only file paths currently supported).

5.153.3.6 void DataStaging::Scheduler::SetPreferredPattern (const std::string & *pattern*)

Set the preferred pattern.

5.153.3.7 void DataStaging::Scheduler::SetSharePriorities (const std::map< std::string, int > & *shares*)

Replace all shares.

5.153.3.8 void DataStaging::Scheduler::SetShareType (TransferShares::ShareType *share_type*)

Set share type.

5.153.3.9 void DataStaging::Scheduler::SetSlots (int *pre_processor* = `0`, int *post_processor* = `0`, int *delivery* = `0`, int *delivery_emergency* = `0`)

Set number of slots for processor and delivery stages.

5.153.3.10 void DataStaging::Scheduler::SetTransferParameters (const TransferParameters & *params*)

Set transfer limits.

5.153.3.11 void DataStaging::Scheduler::SetTransferShares (const TransferShares & *shares*)

Set TransferShares.

5.153.3.12 void DataStaging::Scheduler::SetURLMapping (const Arc::URLMap & *mapping* = `Arc::URLMap()`)

Replace all URL mapping entries.

5.153.3.13 bool DataStaging::Scheduler::start (void)

Start scheduling activity.

This method must be called after all configuration parameters are set properly. Scheduler can be stopped either by calling stop() method or by destroying its instance.

5.153.3.14 bool DataStaging::Scheduler::stop ()

Tell the Scheduler to shut down all threads and exit.

All active DTRs are cancelled and this method waits until they finish (all DTRs go to CANCELLED state)

The documentation for this class was generated from the following file:

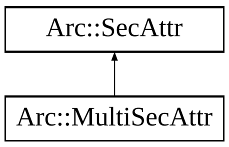
- Scheduler.h

5.154 Arc::SecAttr Class Reference

This is an abstract interface to a security attribute.

```
#include <SecAttr.h>
```

Inheritance diagram for Arc::SecAttr::



Public Member Functions

- SecAttr ()
- bool operator== (const SecAttr &b) const
- bool operator!= (const SecAttr &b) const
- virtual operator bool () const
- virtual bool Export (SecAttrFormat format, std::string &val) const
- virtual bool Export (SecAttrFormat format, XMLNode &val) const
- virtual bool Import (SecAttrFormat format, const std::string &val)
- virtual std::string get (const std::string &id) const
- virtual std::list< std::string > getAll (const std::string &id) const

Static Public Attributes

- static SecAttrFormat ARCAuth
- static SecAttrFormat XACML
- static SecAttrFormat SAML
- static SecAttrFormat GACL

5.154.1 Detailed Description

This is an abstract interface to a security attribute.

This class is meant to be inherited to implement security attributes. Depending on what data it needs to store inheriting classes may need to implement constructor and destructor. They must however override the equality and the boolean operators. The equality is meant to compare security attributes. The prototype implies that all attributes are comparable to all others. This behaviour should be modified as needed by using dynamic_cast operations. The boolean cast operation is meant to embody "nullness" if that is applicable to the particular type.

5.154.2 Constructor & Destructor Documentation

5.154.2.1 Arc::SecAttr::SecAttr () `[inline]`

representation for GACL policy

5.154.3 Member Function Documentation

5.154.3.1 virtual bool Arc::SecAttr::Export (SecAttrFormat *format*, XMLNode & *val*) const `[virtual]`

Convert internal structure into specified format. Returns false if format is not supported/suitable for this attribute. XML node referenced by is turned into top level element of specified format.

Reimplemented in Arc::MultiSecAttr.

5.154.3.2 virtual bool Arc::SecAttr::Export (SecAttrFormat *format*, std::string & *val*) const `[virtual]`

Convert internal structure into specified format. Returns false if format is not supported/suitable for this attribute.

5.154.3.3 virtual std::string Arc::SecAttr::get (const std::string & *id*) const `[virtual]`

Access to specific item of the security attribute. If there are few items of same id the first one is presented. It is meant to be used for tightly coupled SecHandlers and provides more effective interface than Export.

5.154.3.4 virtual std::list<std::string> Arc::SecAttr::getAll (const std::string & *id*) const `[virtual]`

Access to specific items of the security attribute. This method returns all items which have id assigned. It is meant to be used for tightly coupled SecHandlers and provides more effective interface than Export.

5.154.3.5 virtual bool Arc::SecAttr::Import (SecAttrFormat *format*, const std::string & *val*) `[virtual]`

Fills internal structure from external object of specified format. Returns false if failed to do. The usage pattern for this method is not defined and it is provided only to make class symmetric. Hence it's implementation is not required yet.

5.154.3.6 virtual Arc::SecAttr::operator bool () const `[virtual]`

This function should return false if the value is to be considered null, e.g. if it hasn't been set or initialized. In other cases it should return true.

Reimplemented in Arc::MultiSecAttr.

5.154.3.7 bool Arc::SecAttr::operator!= (const SecAttr & *b*) const `[inline]`

This is a convenience function to allow the usage of "not equal" conditions and need not be overridden.

5.154.3.8 bool Arc::SecAttr::operator== (const SecAttr & *b*) const `[inline]`

This function should (in inheriting classes) return true if this and b are considered to represent same content. Identifying and restricting the type of b should be done using dynamic_cast operations. Currently it is not defined how comparison methods to be used. Hence their implementation is not required.

5.154.4 Field Documentation

5.154.4.1 SecAttrFormat Arc::SecAttr::ARCAuth `[static]`

own serialization/deserialization format

5.154.4.2 SecAttrFormat Arc::SecAttr::GACL `[static]`

suitable for inclusion into SAML structures

5.154.4.3 SecAttrFormat Arc::SecAttr::SAML `[static]`

represenation for XACML policy

5.154.4.4 SecAttrFormat Arc::SecAttr::XACML `[static]`

representation for ARC authorization policy

The documentation for this class was generated from the following file:

- SecAttr.h

5.155 Arc::SecAttrFormat Class Reference

Export/import format.

```
#include <SecAttr.h>
```

5.155.1 Detailed Description

Export/import format.

Format is identified by textual identity string. Class description includes basic formats only. That list may be extended.

The documentation for this class was generated from the following file:

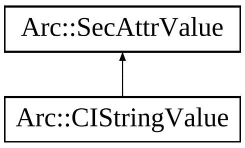
- SecAttr.h

5.156 Arc::SecAttrValue Class Reference

This is an abstract interface to a security attribute.

`#include <SecAttrValue.h>`

Inheritance diagram for Arc::SecAttrValue::



Public Member Functions

- bool operator== (SecAttrValue &b)
- bool operator!= (SecAttrValue &b)
- virtual operator bool ()

5.156.1 Detailed Description

This is an abstract interface to a security attribute.

This class is meant to be inherited to implement security attributes. Depending on what data it needs to store inheriting classes may need to implement constructor and destructor. They must however override the equality and the boolean operators. The equality is meant to compare security attributes. The prototype implies that all attributes are comparable to all others. This behaviour should be modified as needed by using dynamic_cast operations. The boolean cast operation is meant to embody "nullness" if that is applicable to the particular type.

5.156.2 Member Function Documentation

5.156.2.1 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.

5.156.2.2 bool Arc::SecAttrValue::operator!= (SecAttrValue & *b*)

This is a convenience function to allow the usage of "not equal" conditions and need not be overridden.

5.156.2.3 bool Arc::SecAttrValue::operator== (SecAttrValue & *b*)

This function should (in inheriting classes) return true if this and b are considered to be the same. Identifying and restricting the type of b should be done using dynamic_cast operations.

The documentation for this class was generated from the following file:

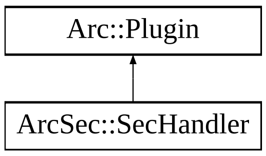
- SecAttrValue.h

5.157 ArcSec::SecHandler Class Reference

Base class for simple security handling plugins.

`#include <SecHandler.h>`

Inheritance diagram for ArcSec::SecHandler::



5.157.1 Detailed Description

Base class for simple security handling plugins.

This virtual class defines method Handle() which processes security related information/attributes in Message and optionally makes security decision. Instances of such classes are normally arranged in chains abd are called on incoming and outgoing messages in various MCC and Service plugins. Return value of Handle() defines either processing should continie (true) or stop with error (false). Configuration of SecHandler is consumed during creation of instance through XML subtree fed to constructor.

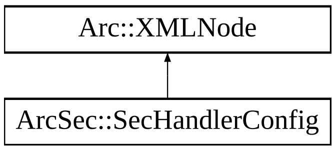
The documentation for this class was generated from the following file:

- SecHandler.h

5.158 ArcSec::SecHandlerConfig Class Reference

```
#include <SecHandler.h>
```

Inheritance diagram for ArcSec::SecHandlerConfig::



5.158.1 Detailed Description

Helper class to create Security Handler configuration

The documentation for this class was generated from the following file:

- SecHandler.h

5.159 ArcSec::Security Class Reference

Common stuff used by security related slasses.

```
#include <Security.h>
```

5.159.1 Detailed Description

Common stuff used by security related slasses.

This class is just a place where to put common stuff that is used by security related slasses. So far it only contains a logger.

The documentation for this class was generated from the following file:

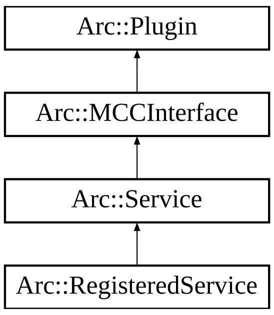
- Security.h

5.160 Arc::Service Class Reference

Service - last component in a Message 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

5.160.1 Detailed Description

Service - last component in a Message Chain.

This class which defines interface and common functionality for every Service plugin. Interface is made of method process() which is called by Plexer or MCC class. There is one Service object created for every service description processed by Loader class objects. Classes derived from Service class must implement process() method of MCCInterface. It is up to developer how internal state of service is stored and communicated to other services and external utilites. Service is free to expect any type of payload passed to it and generate any payload as well. Useful types depend on MCCs in chain which leads to that service. For example if service is expected to by linked to SOAP MCC it must accept and generate messages with

PayloadSOAP payload. Method process() of class derived from Service class may be called concurently in multiple threads. Developers must take that into account and write thread-safe implementation. Simple example of service is provided in /src/tests/echo/echo.cpp of source tree. The way to write client couterpart of corresponding service is undefined yet. For example see /src/tests/echo/test.cpp .

5.160.2 Constructor & Destructor Documentation

5.160.2.1 Arc::Service::Service (Config ∗)

Example contructor - Server takes at least it's configuration subtree

5.160.3 Member Function Documentation

5.160.3.1 virtual void Arc::Service::AddSecHandler (Config ∗ *cfg*, ArcSec::SecHandler ∗ *sechandler*, const std::string & *label* = "") `[virtual]`

Add security components/handlers to this MCC. For more information please see description of MCC::AddSecHandler

5.160.3.2 virtual std::string Arc::Service::getID () `[inline, virtual]`

Service may implement own service identitifer gathering method. This method return identifier of service which is used for registering it Information Services.

5.160.3.3 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

5.160.3.4 virtual bool Arc::Service::RegistrationCollector (XMLNode & *doc*) `[virtual]`

Service specific registartion collector, used for generate service registartions. In implemented service this method should generate GLUE2 document with part of service description which service wishes to advertise to Information Services.

5.160.4 Field Documentation

5.160.4.1 Logger Arc::Service::logger `[static, protected]`

Logger object used to print messages generated by this class.

5.160.4.2 std::map<std::string,std::list<ArcSec::SecHandler∗> > Arc::Service::sechandlers_ `[protected]`

Set of labeled authentication and authorization handlers. MCC calls sequence of handlers at specific point depending on associated identifier. in most aces those are "in" and "out" for incoming and outgoing messages correspondingly.

The documentation for this class was generated from the following file:

- Service.h

5.161 Arc::SimpleCondition Class Reference

Simple triggered condition.

`#include <Thread.h>`

Public Member Functions

- void lock (void)
- void unlock (void)
- void signal (void)
- void signal_nonblock (void)
- void broadcast (void)
- void wait (void)
- void wait_nonblock (void)
- bool wait (int t)
- void reset (void)

5.161.1 Detailed Description

Simple triggered condition.

Provides condition and semaphor objects in one element.

5.161.2 Member Function Documentation

5.161.2.1 void Arc::SimpleCondition::broadcast (void) `[inline]`

Signal about condition to all waiting threads

5.161.2.2 void Arc::SimpleCondition::lock (void) `[inline]`

Acquire semaphor

5.161.2.3 void Arc::SimpleCondition::reset (void) `[inline]`

Reset object to initial state

5.161.2.4 void Arc::SimpleCondition::signal (void) `[inline]`

Signal about condition

5.161.2.5 void Arc::SimpleCondition::signal_nonblock (void) `[inline]`

Signal about condition without using semaphor

5.161.2.6 void Arc::SimpleCondition::unlock (void) `[inline]`

Release semaphor

5.161.2.7 bool Arc::SimpleCondition::wait (int *t*) `[inline]`

Wait for condition no longer than t milliseconds

5.161.2.8 void Arc::SimpleCondition::wait (void) `[inline]`

Wait for condition

5.161.2.9 void Arc::SimpleCondition::wait_nonblock (void) `[inline]`

Wait for condition without using semaphor

The documentation for this class was generated from the following file:

- Thread.h

5.162 Arc::SimpleFIFO Class Reference

Class representing a named pipe.

`#include <SimpleFIFO.h>`

Public Member Functions

- SimpleFIFO (const std::string &path)
- bool read (std::string &data, unsigned int timeout=0)
- bool write (const std::string &data="")
- operator bool () const
- bool operator! () const

5.162.1 Detailed Description

Class representing a named pipe.

It should be used when process A wants to wait for notification from process B before continuing. Process A calls read(), which blocks until process B calls write(). A timeout can be set on read(). write() does not block - if there is no process calling read() then write() does not write data to the pipe and returns true.

5.162.2 Constructor & Destructor Documentation

5.162.2.1 Arc::SimpleFIFO::SimpleFIFO (const std::string & *path*)

Create named pipe, if it doesn't already exist.

5.162.3 Member Function Documentation

5.162.3.1 Arc::SimpleFIFO::operator bool (void) const `[inline]`

Is object valid?

5.162.3.2 bool Arc::SimpleFIFO::operator! (void) const `[inline]`

Is object not valid?

5.162.3.3 bool Arc::SimpleFIFO::read (std::string & *data*, unsigned int *timeout* = `0`)

Attempt to read from pipe into data - blocks until write() is called on the same named pipe or timeout expires. Returns true if data was successfully read from pipe or timeout expired (data is empty in this case).

Parameters:

data The data that was read from the pipe

timeout Time to wait for read, in milliseconds. If zero then read() blocks forever. NOTE: If timeout is too small then a race condition may result in read() blocking.

5.162.3.4 bool Arc::SimpleFIFO::write (const std::string & *data* = "")

Write data to pipe. Returns true if data was successfully written to pipe, or if no process was listening.

Parameters:

data The data to write to the pipe

The documentation for this class was generated from the following file:

- SimpleFIFO.h

5.163 Arc::SOAPMessage Class Reference

Message restricted to SOAP payload.

```
#include <SOAPMessage.h>
```

Public Member Functions

- SOAPMessage (void)
- SOAPMessage (long msg_ptr_addr)
- SOAPMessage (Message &msg)
- ∼SOAPMessage (void)
- SOAPEnvelope ∗ Payload (void)
- void Payload (SOAPEnvelope ∗new_payload)
- MessageAttributes ∗ Attributes (void)

5.163.1 Detailed Description

Message restricted to SOAP payload.

This is a special Message intended to be used in language bindings for programming languages which are not flexible enough to support all kinds of Payloads. It is passed through chain of MCCs and works like the Message but can carry only SOAP content.

5.163.2 Constructor & Destructor Documentation

5.163.2.1 Arc::SOAPMessage::SOAPMessage (void) `[inline]`

Dummy constructor

5.163.2.2 Arc::SOAPMessage::SOAPMessage (long *msg_ptr_addr*)

Copy constructor. Used by language bindigs

5.163.2.3 Arc::SOAPMessage::SOAPMessage (Message & *msg*)

Copy constructor. Ensures shallow copy.

5.163.2.4 Arc::SOAPMessage::∼SOAPMessage (void)

Destructor does not affect refered objects

5.163.3 Member Function Documentation

5.163.3.1 MessageAttributes∗ Arc::SOAPMessage::Attributes (void) `[inline]`

Returns a pointer to the current attributes object or NULL if no attributes object has been assigned.

5.163.3.2 void Arc::SOAPMessage::Payload (SOAPEnvelope ∗ *new_payload*)

Replace payload with a COPY of new one

5.163.3.3 SOAPEnvelope∗ Arc::SOAPMessage::Payload (void)

Returns pointer to current payload or NULL if no payload assigned.

The documentation for this class was generated from the following file:

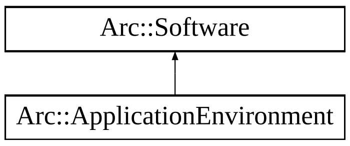
- SOAPMessage.h

5.164 Arc::Software Class Reference

Used to represent software (names and version) and comparison.

```
#include <Software.h>
```

Inheritance diagram for Arc::Software::



Public Types

- typedef bool(Software::∗) ComparisonOperator (const Software &) const
- NOTEQUAL = 0
- EQUAL = 1
- GREATERTHAN = 2
- LESSTHAN = 3
- GREATERTHANOREQUAL = 4
- LESSTHANOREQUAL = 5
- enum ComparisonOperatorEnum {
  NOTEQUAL = 0, EQUAL = 1, GREATERTHAN = 2, LESSTHAN = 3,
  GREATERTHANOREQUAL = 4, LESSTHANOREQUAL = 5 }

Public Member Functions

- Software ()
- Software (const std::string &name_version)
- Software (const std::string &name, const std::string &version)
- Software (const std::string &family, const std::string &name, const std::string &version)
- bool empty () const
- bool operator== (const Software &sw) const
- bool operator!= (const Software &sw) const
- bool operator> (const Software &sw) const
- bool operator< (const Software &sw) const
- bool operator>= (const Software &sw) const
- bool operator<= (const Software &sw) const
- std::string operator() () const
- operator std::string (void) const
- const std::string & getFamily () const
- const std::string & getName () const
- const std::string & getVersion () const

Static Public Member Functions

- static ComparisonOperator convert (const ComparisonOperatorEnum &co)
- static std::string toString (ComparisonOperator co)

Static Public Attributes

- static const std::string VERSIONTOKENS

Friends

- std::ostream & operator<< (std::ostream &out, const Software &sw)

5.164.1 Detailed Description

Used to represent software (names and version) and comparison.

The Software class is used to represent the name of a piece of software internally. Generally software are identified by a name and possibly a version number. Some software can also be categorized by type or family (compilers, operating system, etc.). A software object can be compared to other software objects using the comparison operators contained in this class. The basic usage of this class is to test if some specified software requirement (SoftwareRequirement) are fulfilled, by using the comparability of the class.

Internally the Software object is represented by a family and name identifier, and the software version is tokenized at the characters defined in VERSIONTOKENS, and stored as a list of tokens.

5.164.2 Member Typedef Documentation

5.164.2.1 typedef bool(Software::∗) Arc::Software::ComparisonOperator(const Software &) const

Definition of a comparison operator method pointer.

This `typedef` defines a comparison operator method pointer.

See also:

operator==,
operator!=,
operator>,
operator<,
operator>=,
operator<=,
ComparisonOperatorEnum.

5.164.3 Member Enumeration Documentation

5.164.3.1 enum Arc::Software::ComparisonOperatorEnum

Comparison operator enum.

The ComparisonOperatorEnum enumeration is a 1-1 correspondance between the defined comparison method operators (Software::ComparisonOperator), and can be used in circumstances where method pointers are not supported.

Enumerator:

NOTEQUAL see operator!=

EQUAL see operator==

GREATERTHAN see operator>

LESSTHAN see operator<

GREATERTHANOREQUAL see operator>=

LESSTHANOREQUAL see operator<=

5.164.4 Constructor & Destructor Documentation

5.164.4.1 Arc::Software::Software () `[inline]`

Dummy constructor.

This constructor creates a empty object.

5.164.4.2 Arc::Software::Software (const std::string & *name_version*)

Create a Software object.

Create a Software object from a single string composed of a name and a version part. The created object will contain a empty family part. The name and version part of the string will be split at the first 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.

5.164.4.3 Arc::Software::Software (const std::string & *name*, const std::string & *version*)

Create a Software object.

Create a Software object with the specified name and version. The family part will be left empty.

Parameters:

name the software name to represent.

version the software version to represent.

5.164.4.4 Arc::Software::Software (const std::string & *family*, const std::string & *name*, const std::string & *version*)

Create a Software object.

Create a Software object with the specified family, name and version.

Parameters:

family the software family to represent.

name the software name to represent.

version the software version to represent.

5.164.5 Member Function Documentation

5.164.5.1 static ComparisonOperator Arc::Software::convert (const ComparisonOperatorEnum & *co*) `[static]`

Convert a ComparisonOperatorEnum value to a comparison method pointer.

The passed ComparisonOperatorEnum will be converted to a comparison method pointer defined by the Software::ComparisonOperator typedef.

This static method is not defined in language bindings created with Swig, since method pointers are not supported by Swig.

Parameters:

co a ComparisonOperatorEnum value.

Returns:

A method pointer to a comparison method is returned.

5.164.5.2 bool Arc::Software::empty () const `[inline]`

Indicates whether the object is empty.

Returns:

`true` if the name of this object is empty, otherwise `false`.

5.164.5.3 const std::string& Arc::Software::getFamily () const `[inline]`

Get family.

Returns:

The family the represented software belongs to is returned.

5.164.5.4 const std::string& Arc::Software::getName () const `[inline]`

Get name.

Returns:

The name of the represented software is returned.

5.164.5.5 const std::string& Arc::Software::getVersion () const `[inline]`

Get version.

Returns:

The version of the represented software is returned.

5.164.5.6 Arc::Software::operator std::string (void) const `[inline]`

Cast to string.

This casting operator behaves exactly as ::operator()() does. The cast is used like (std::string) <software-object>.

See also:

operator()().

5.164.5.7 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 objects are of different versions but share the same name and family. So it should not be used to test if two Software objects differ in either name, version or family. Two Software 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 object.

Returns:

`true` when the two objects are inequal, otherwise `false`.

5.164.5.8 std::string Arc::Software::operator() () const

Get string representation.

Returns the string representation of this object, which is 'family'-'name'-'version'.

Returns:

The string representation of this object is returned.

See also:

operator std::string().

5.164.5.9 bool Arc::Software::operator< (const Software & *sw*) const `[inline]`

Less-than operator.

The behaviour of this less-than operator is equivalent to the greater-than operator (operator>()) with the LHS and RHS swapped.

Parameters:

sw is the RHS object.

Returns:

`true` if the LHS is less than the RHS, otherwise `false`.

See also:

operator>().

5.164.5.10 bool Arc::Software::operator<= (const Software & *sw*) const [inline]

Less-than or equal operator.

The LHS object is greater than or equal to the RHS object if the LHS equal the RHS (operator==()) or if the LHS is greater than the RHS (operator>()).

Parameters:

sw is the RHS object.

Returns:

`true` if the LHS is less than or equal the RHS, otherwise `false`.

See also:

operator==(),
operator<().

5.164.5.11 bool Arc::Software::operator== (const Software & *sw*) const [inline]

Equality operator.

Two Software 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 ComparisonOperatorEnum value.

Parameters:

sw is the RHS Software object.

Returns:

`true` when the two objects equals, otherwise `false`.

5.164.5.12 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`.

5.164.5.13 bool Arc::Software::operator>= (const Software & *sw*) const `[inline]`

Greater-than or equal operator.

The LHS object is greater than or equal to the RHS object if the LHS equal the RHS (operator==()) or if the LHS is greater than the RHS (operator>()).

Parameters:

sw is the RHS object.

Returns:

`true` if the LHS is greated than or equal the RHS, otherwise `false`.

See also:

operator==(),
operator>().

5.164.5.14 static std::string Arc::Software::toString (ComparisonOperator *co*) `[static]`

Convert Software::ComparisonOperator to a string.

This method is not available in language bindings created by Swig, since method pointers are not supported by Swig.

Parameters:

co is a Software::ComparisonOperator.

Returns:

The string representation of the passed Software::ComparisonOperator is returned.

5.164.6 Friends And Related Function Documentation

5.164.6.1 std::ostream& operator<< (std::ostream & *out*, const Software & *sw*) [friend]

Write Software string representation to a std::ostream.

Write the string representation of a Software object to a std::ostream.

Parameters:

out is a std::ostream to write the string representation of the Software object to.

sw is the Software object to write to the std::ostream.

Returns:

The passed std::ostream *out* is returned.

5.164.7 Field Documentation

5.164.7.1 const std::string Arc::Software::VERSIONTOKENS [static]

Tokens used to split version string.

This string constant specifies which tokens will be used to split the version string.

The documentation for this class was generated from the following file:

- Software.h

5.165 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

5.165.1 Detailed Description

Class used to express and resolve version requirements on software.

A requirement in this class is defined as a pair composed of a Software object and either a Software::ComparisonOperator method pointer or equally a Software::ComparisonOperatorEnum enum value. A SoftwareRequirement object can contain multiple of such requirements, and then it can specified if all these requirements should be satisfied, or if it is enough to satisfy only one of them. The requirements can be satisfied by a single Software object or a list of either Software or ApplicationEnvironment objects, by using the method isSatisfied(). This class also contain a number of methods (selectSoftware()) to select Software objects which are satisfying the requirements, and in this way resolving requirements.

5.165.2 Constructor & Destructor Documentation

5.165.2.1 Arc::SoftwareRequirement::SoftwareRequirement (bool *requiresAll* = `false`) `[inline]`

Create a empty SoftwareRequirement object.

The created SoftwareRequirement 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.

5.165.2.2 Arc::SoftwareRequirement::SoftwareRequirement (const Software & *sw*, Software::ComparisonOperator *swComOp* = `&Software::operator==`, bool *requiresAll* = `false`)

Create a SoftwareRequirement object.

The created SoftwareRequirement object will contain one requirement specified by the Software object *sw*, and the Software::ComparisonOperator *swComOp*.

This constructor is not available in language bindings created by Swig, since method pointers are not supported by Swig, see SoftwareRequirement(const Software&, Software::ComparisonOperatorEnum, bool) instead.

Parameters:

sw is the Software object of the requirement to add.

swComOp is the Software::ComparisonOperator 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.

5.165.2.3 Arc::SoftwareRequirement::SoftwareRequirement (const Software & *sw*, Software::ComparisonOperatorEnum *co*, bool *requiresAll* = `false`)

Create a SoftwareRequirement object.

The created SoftwareRequirement object will contain one requirement specified by the Software object *sw*, and the Software::ComparisonOperatorEnum *co*.

Parameters:

sw is the Software object of the requirement to add.

co is the Software::ComparisonOperatorEnum of the requirement to add.

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.

5.165.2.4 Arc::SoftwareRequirement::SoftwareRequirement (const SoftwareRequirement & *sr*) `[inline]`

Copy constructor.

Create a SoftwareRequirement object from another SoftwareRequirement object.

Parameters:

sr is the SoftwareRequirement object to make a copy of.

5.165.3 Member Function Documentation

5.165.3.1 void Arc::SoftwareRequirement::add (const Software & *sw*, Software::ComparisonOperatorEnum *co*)

Add a Software 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 object to add as part of a requirement.

co is the Software::ComparisonOperatorEnum value to add as part of a requirement, the default enum will be Software::EQUAL.

5.165.3.2 void Arc::SoftwareRequirement::add (const Software & *sw*, Software::ComparisonOperator *swComOp* = `&Software::operator==`)

Add a Software 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) instead.

Parameters:

sw is the Software object to add as part of a requirement.

swComOp is the Software::ComparisonOperator method pointer to add as part of a requirement, the default operator will be Software::operator==().

5.165.3.3 void Arc::SoftwareRequirement::clear () `[inline]`

Clear the object.

The requirements in this object will be cleared when invoking this method.

5.165.3.4 bool Arc::SoftwareRequirement::empty () const `[inline]`

Test if the object is empty.

Returns:

`true` if this object do no contain any requirements, otherwise `false`.

5.165.3.5 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,
getSoftwareList.

5.165.3.6 const std::list<Software>& Arc::SoftwareRequirement::getSoftwareList () const `[inline]`

Get list of Software objects.

Returns:

The list of internally stored Software objects is returned.

See also:

Software,
getComparisonOperatorList.

5.165.3.7 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.

5.165.3.8 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==).

If specified that all requirements has to be satisfied, then for this object to be resolved each requirement must have a Software object with a unique family/name composition, i.e. no other requirements have a Software object with the same family/name composition, and each requirement must use the equal operator (Software::operator==).

If this object has been resolved then `true` is returned when invoking this method, otherwise `false` is returned.

Returns:

`true` if this object have been resolved, otherwise `false`.

5.165.3.9 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 method does.

Parameters:

swList is the list of ApplicationEnvironment objects which should be used to try satisfy the requirements.

Returns:

`true` if requirements are satisfied, otherwise `false`.

See also:

isSatisfied(const Software&) const,
isSatisfied(const std::list<Software>&) const,
selectSoftware(const std::list<ApplicationEnvironment>&),
isResolved() const.

5.165.3.10 bool Arc::SoftwareRequirement::isSatisfied (const std::list< Software > & *swList*) const

Test if requirements are satisfied.

Returns `true` if stored requirements are satisfied by software specified in *swList*, otherwise `false` is returned.

Note that if all requirements must be satisfied and multiple requirements exist having identical name and family all these requirements should be satisfied by a single Software object.

Parameters:

swList is the list of Software objects which should be used to try satisfy the requirements.

Returns:

`true` if requirements are satisfied, otherwise `false`.

See also:

isSatisfied(const Software&) const,
isSatisfied(const std::list<ApplicationEnvironment>&) const,
selectSoftware(const std::list<Software>&),
isResolved() const.

5.165.3.11 bool Arc::SoftwareRequirement::isSatisfied (const Software & *sw*) const `[inline]`

Test if requirements are satisfied.

Returns `true` if the requirements are satisfied by the specified Software *sw*, otherwise `false` is returned.

Parameters:

sw is the Software which should satisfy the requirements.

Returns:

`true` if requirements are satisfied, otherwise `false`.

See also:

isSatisfied(const std::list<Software>&) const,
isSatisfied(const std::list<ApplicationEnvironment>&) const,
selectSoftware(const Software&),
isResolved() const.

5.165.3.12 SoftwareRequirement& Arc::SoftwareRequirement::operator= (const SoftwareRequirement & *sr*)

Assignment operator.

Set this object equal to that of the passed SoftwareRequirement object *sr*.

Parameters:

sr is the SoftwareRequirement object to set object equal to.

5.165.3.13 bool Arc::SoftwareRequirement::selectSoftware (const std::list< ApplicationEnvironment > & *swList*)

Select software.

This method behaves exactly as the selectSoftware(const std::list<Software>&) method does.

Parameters:

swList is a list of ApplicationEnvironment objects used to satisfy requirements.

Returns:

`true` if requirements are satisfied, otherwise `false`.

See also:

selectSoftware(const Software&),
selectSoftware(const std::list<Software>&),
isSatisfied(const std::list<ApplicationEnvironment>&) const,
isResolved() const.

5.165.3.14 bool Arc::SoftwareRequirement::selectSoftware (const std::list< Software > & *swList*)

Select software.

If the passed list of Software objects *swList* do not satisfy the requirements `false` is returned and this object is not modified. If however the list of Software objects *swList* do satisfy the requirements `true` is returned and the Software objects satisfying the requirements will replace these with the equality operator (Software::operator==) used as the comparator for the new requirements.

Note that if all requirements must be satisfied and multiple requirements exist having identical name and family all these requirements should be satisfied by a single Software object and it will replace all these requirements.

Parameters:

swList is a list of Software objects used to satisfy requirements.

Returns:

`true` if requirements are satisfied, otherwise `false`.

See also:

selectSoftware(const Software&),
selectSoftware(const std::list<ApplicationEnvironment>&),
isSatisfied(const std::list<Software>&) const,
isResolved() const.

5.165.3.15 bool Arc::SoftwareRequirement::selectSoftware (const Software & *sw*) `[inline]`

Select software.

If the passed Software *sw* do not satisfy the requirements `false` is returned and this object is not modified. If however the Software object *sw* do satisfy the requirements `true` is returned and the requirements are set to equal the *sw* Software object.

Parameters:

sw is the Software object used to satisfy requirements.

Returns:

`true` if requirements are satisfied, otherwise `false`.

See also:

selectSoftware(const std::list<Software>&),
selectSoftware(const std::list<ApplicationEnvironment>&),
isSatisfied(const Software&) const,
isResolved() const.

5.165.3.16 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().

The documentation for this class was generated from the following file:

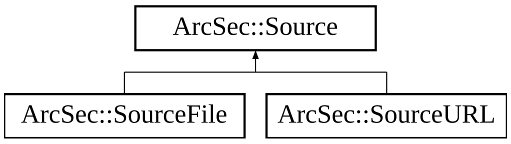
- Software.h

5.166 ArcSec::Source Class Reference

Acquires and parses XML document from specified source.

```
#include <Source.h>
```

Inheritance diagram for ArcSec::Source::



Public Member Functions

- Source (const Source &s)
- Source (Arc::XMLNode &xml)
- Source (std::istream &stream)
- Source (Arc::URL &url)
- Source (const std::string &str)
- Arc::XMLNode Get (void) const
- operator bool (void)

5.166.1 Detailed Description

Acquires and parses XML document from specified source.

This class is to be used to provide easy way to specify different sources for XML Authorization Policies and Requests.

5.166.2 Constructor & Destructor Documentation

5.166.2.1 ArcSec::Source::Source (const Source & *s*) `[inline]`

Copy constructor.

Use this constructor only for temporary objects. Parsed XML document is still owned by copied source and hence lifetime of create object should not exceed that of copied one.

5.166.2.2 ArcSec::Source::Source (Arc::XMLNode & *xml*)

Copy XML tree from XML subtree refered by xml.

5.166.2.3 ArcSec::Source::Source (std::istream & *stream*)

Read XML document from stream and parse it.

5.166.2.4 ArcSec::Source::Source (Arc::URL & *url*)

Fetch XML document from specified url and parse it.

This constructor is not implemented yet.

5.166.2.5 ArcSec::Source::Source (const std::string & *str*)

Read XML document from string.

5.166.3 Member Function Documentation

5.166.3.1 Arc::XMLNode ArcSec::Source::Get (void) const `[inline]`

Get reference to parsed document.

5.166.3.2 ArcSec::Source::operator bool (void) `[inline]`

Returns true if valid document is available.

The documentation for this class was generated from the following file:

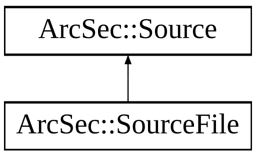
- Source.h

5.167 ArcSec::SourceFile Class Reference

Convenience class for obtaining XML document from file.

`#include <Source.h>`

Inheritance diagram for ArcSec::SourceFile::



Public Member Functions

- SourceFile (const SourceFile &s)
- SourceFile (const char ∗name)
- SourceFile (const std::string &name)

5.167.1 Detailed Description

Convenience class for obtaining XML document from file.

5.167.2 Constructor & Destructor Documentation

5.167.2.1 ArcSec::SourceFile::SourceFile (const SourceFile & *s*) `[inline]`

See corresponding constructor of Source class.

5.167.2.2 ArcSec::SourceFile::SourceFile (const char ∗ *name*)

Read XML document from file named name and store it.

5.167.2.3 ArcSec::SourceFile::SourceFile (const std::string & *name*)

Read XML document from file named name and store it.

The documentation for this class was generated from the following file:

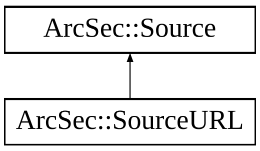
- Source.h

5.168 ArcSec::SourceURL Class Reference

Convenience class for obtaining XML document from remote URL.

`#include <Source.h>`

Inheritance diagram for ArcSec::SourceURL::



Public Member Functions

- SourceURL (const SourceURL &s)
- SourceURL (const char ∗url)
- SourceURL (const std::string &url)

5.168.1 Detailed Description

Convenience class for obtaining XML document from remote URL.

5.168.2 Constructor & Destructor Documentation

5.168.2.1 ArcSec::SourceURL::SourceURL (const SourceURL & *s*) `[inline]`

See corresponding constructor of Source class.

5.168.2.2 ArcSec::SourceURL::SourceURL (const char ∗ *url*)

Read XML document from URL url and store it.

5.168.2.3 ArcSec::SourceURL::SourceURL (const std::string & *url*)

Read XML document from URL url and store it.

The documentation for this class was generated from the following file:

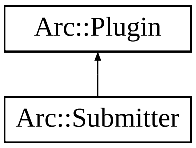
- Source.h

5.169 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

5.169.1 Detailed Description

Base class for the Submitters.

Submitter is the base class for Grid middleware specialized Submitter objects. The class submits job(s) to the computing resource it represents and uploads (needed by the job) local input files.

5.169.2 Member Function Documentation

5.169.2.1 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 JobDescription or NULL if ther is no test description defined with the requested id.

5.169.2.2 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 jobid, and is represented by the JobDescription jobdesc. The forcemigration boolean specifies if the migration should succeed if the active job cannot be terminated. The protected method AddJob can be used to save job information. This method should return the URL of the migrated job. In case migration fails an empty URL should be returned.

5.169.2.3 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 jobdesc, to the ExecutionTarget et. The protected convenience method AddJob can be used to save job information. This method should return the URL of the submitted job. In case submission fails an empty URL should be returned.

5.169.3 Field Documentation

5.169.3.1 const ExecutionTarget* Arc::Submitter::target `[protected]`

Target to submit to.

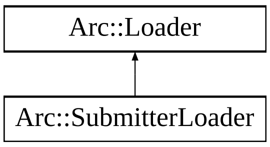
The documentation for this class was generated from the following file:

- Submitter.h

5.170 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

5.170.1 Detailed Description

Class responsible for loading Submitter plugins The Submitter objects returned by a SubmitterLoader must not be used after the SubmitterLoader goes out of scope.

5.170.2 Constructor & Destructor Documentation

5.170.2.1 Arc::SubmitterLoader::SubmitterLoader ()

Constructor Creates a new SubmitterLoader.

5.170.2.2 Arc::SubmitterLoader::∼SubmitterLoader ()

Destructor Calling the destructor destroys all Submitters loaded by the SubmitterLoader instance.

5.170.3 Member Function Documentation

5.170.3.1 const std::list<Submitter∗>& Arc::SubmitterLoader::GetSubmitters () const `[inline]`

Retrieve the list of loaded Submitters.

Returns:

A reference to the list of Submitters.

5.170.3.2 Submitter∗ Arc::SubmitterLoader::load (const std::string & *name*, const UserConfig & *usercfg*)

Load a new Submitter

Parameters:

name	The name of the Submitter to load.
usercfg	The UserConfig object for the new Submitter.

Returns:

A pointer to the new Submitter (NULL on error).

The documentation for this class was generated from the following file:

- Submitter.h

5.171 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)

5.171.1 Detailed Description

Target generation class

The TargetGenerator 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 loads TargetRetriever plugins (which implements the actual information retrieval) from URL objects found in the UserConfig object passed to its constructor using the custom TargetRetriever-Loader.

5.171.2 Constructor & Destructor Documentation

5.171.2.1 Arc::TargetGenerator::TargetGenerator (const UserConfig & *usercfg*, unsigned int *startRetrieval* = 0)

Create a TargetGenerator object.

Default constructor to create a TargeGenerator. The constructor reads the computing and index service URL objects from the passed UserConfig object using the UserConfig:GetSelectedServices method. From each URL a matching specialized TargetRetriever plugin is loaded using the TargetRetrieverLoader. 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 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.

5.171.3 Member Function Documentation

5.171.3.1 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 against the servers returned by UserConfig::GetRejectedServices and only allows to add the service if not specifically rejected.

Parameters:

flavour The flavour if the the index server.

url URL pointing to the index server.

5.171.3.2 void Arc::TargetGenerator::AddJob (const Job & *job*)

Add a new Job to this object.

Method to add a new Job (usually discovered by a TargetRetriever) to the internal list of jobs in a thread secure way.

Parameters:

job Job describing the job.

See also:

AddJob(const Job&)

5.171.3.3 void Arc::TargetGenerator::AddJob (const XMLNode & *job*)

DEPRECATED: Add a new Job to this object.

This method is DEPRECATED, use the AddJob(const Job&) method instead. Method to add a new Job (usually discovered by a TargetRetriever) to the internal list of jobs in a thread secure way.

Parameters:

job XMLNode describing the job.

5.171.3.4 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 against the services returned by UserConfig::GetRejectedServices and only allows to add the service if not specifically rejected.

Parameters:

> ***flavour*** The flavour if the the computing service.
>
> ***url*** URL pointing to the information system of the computing service.

5.171.3.5 void Arc::TargetGenerator::AddTarget (const ExecutionTarget & *target*)

Add a new ExecutionTarget to the foundTargets list.

Method to add a new ExecutionTarget (usually discovered by a TargetRetriever) to the list of foundTargets in a thread secure way.

Parameters:

> ***target*** ExecutionTarget to be added.

5.171.3.6 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.

5.171.3.7 const std::list<ExecutionTarget>& Arc::TargetGenerator::FoundTargets () const `[inline]`

DEPRECATED: Return targets found by GetTargets.

This method is DEPRECATED, use the FoundTargets() instead. Method to return the list of Execution-Target objects (currently only supported Target type) found by the GetTarget method.

5.171.3.8 const std::list<ExecutionTarget>& Arc::TargetGenerator::GetExecutionTargets () const `[inline]`

Return targets fetched by RetrieveExecutionTargets method.

Method to return a const list of ExecutionTarget objects retrieved by the RetrieveExecutionTargets method.

See also:

> RetrieveExecutionTargets
> GetExecutionTargets

5.171.3.9 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 objects is returned

See also:

RetrieveJobs

5.171.3.10 void Arc::TargetGenerator::GetTargets (int *targetType*, int *detailLevel*)

DEPRECATED: Find available targets.

This method is DEPRECATED, use the RetrieveExecutionTargets() or RetrieveJobs() method instead. Method to prepare a list of chosen Targets with a specified detail level. Current implementation supports finding computing elements (ExecutionTarget) with full detail level and jobs with limited detail level.

Parameters:

targetType	0 = ExecutionTarget, 1 = Grid jobs
detailLevel	

See also:

RetrieveExecutionsTargets()
RetrieveJobs()

5.171.3.11 std::list<ExecutionTarget>& Arc::TargetGenerator::ModifyFoundTargets ()

DEPRECATED: Return targets found by GetTargets.

This method is DEPRECATED, use the FoundTargets() instead. Method to return the list of ExecutionTarget objects (currently only supported Target type) found by the GetTarget method.

5.171.3.12 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

5.171.3.13 void Arc::TargetGenerator::RetrieveExecutionTargets ()

Retrieve available execution services.

The endpoints specified in the UserConfig object passed to this object will be used to retrieve information about execution services (ExecutionTarget 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
GetExecutionTargets

5.171.3.14 void Arc::TargetGenerator::RetrieveJobs ()

Retrieve job information from execution services.

The endpoints specified in the UserConfig 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 object will be considered (exception being services which has no user authentication). If a endpoint is a index service, each execution service registered will be queried, and searched for job information.

See also:

RetrieveExecutionTargets

5.171.3.15 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

5.171.3.16 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:

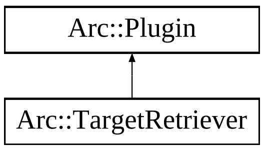
- TargetGenerator.h

5.172 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

5.172.1 Detailed Description

TargetRetriever base class

The TargetRetriever class is a pure virtual base class to be used for grid flavour specializations. It is designed to work in conjunction with the TargetGenerator.

5.172.2 Constructor & Destructor Documentation

5.172.2.1 Arc::TargetRetriever::TargetRetriever (const UserConfig & *usercfg*, const URL & *url*, ServiceType *st*, const std::string & *flavour*) `[protected]`

TargetRetriever constructor.

Default constructor to create a TargeGenerator. The constructor reads the computing and index service URL objects from the

Parameters:

usercfg

url

st

flavour

5.172.3 Member Function Documentation

5.172.3.1 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 which has loaded the TargetRetriever

detailLevel is the required level of details (1 = All details, 2 = Limited details)

5.172.3.2 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 which has loaded the TargetRetriever

detailLevel is the required level of details (1 = All details, 2 = Limited details)

5.172.3.3 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 which has loaded the TargetRetriever

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)

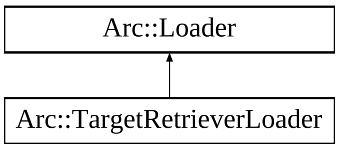
The documentation for this class was generated from the following file:

- TargetRetriever.h

5.173 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

5.173.1 Detailed Description

Class responsible for loading TargetRetriever plugins The TargetRetriever objects returned by a TargetRetrieverLoader must not be used after the TargetRetrieverLoader goes out of scope.

5.173.2 Constructor & Destructor Documentation

5.173.2.1 Arc::TargetRetrieverLoader::TargetRetrieverLoader ()

Constructor Creates a new TargetRetrieverLoader.

5.173.2.2 Arc::TargetRetrieverLoader::~TargetRetrieverLoader ()

Destructor Calling the destructor destroys all TargetRetrievers loaded by the TargetRetrieverLoader instance.

5.173.3 Member Function Documentation

5.173.3.1 const std::list<TargetRetriever∗>& Arc::TargetRetrieverLoader::GetTargetRetrievers () const `[inline]`

Retrieve the list of loaded TargetRetrievers.

Returns:

A reference to the list of TargetRetrievers.

5.173.3.2 TargetRetriever∗ Arc::TargetRetrieverLoader::load (const std::string & *name*, const UserConfig & *usercfg*, const std::string & *service*, const ServiceType & *st*)

Load a new TargetRetriever

Parameters:

name The name of the TargetRetriever to load.

usercfg The UserConfig object for the new TargetRetriever.

service The URL used to contact the target.

st specifies service type of the target.

Returns:

A pointer to the new TargetRetriever (NULL on error).

The documentation for this class was generated from the following file:

- TargetRetriever.h

5.174 Arc::ThreadDataItem Class Reference

Base class for per-thread object.

`#include <Thread.h>`

Public Member Functions

- ThreadDataItem (void)
- ThreadDataItem (std::string &key)
- ThreadDataItem (const std::string &key)
- void Attach (std::string &key)
- void Attach (const std::string &key)
- virtual void Dup (void)

Static Public Member Functions

- static ThreadDataItem ∗ Get (const std::string &key)

5.174.1 Detailed Description

Base class for per-thread object.

Classes inherited from this one are attached to current thread under specified key and destroyed only when thread ends or object is replaced by another one with same key.

5.174.2 Constructor & Destructor Documentation

5.174.2.1 Arc::ThreadDataItem::ThreadDataItem (void)

Dummy constructor which does nothing. To make object usable one of Attach(...) methods must be used.

5.174.2.2 Arc::ThreadDataItem::ThreadDataItem (std::string & *key*)

Creates instance and attaches it to current thread under key. If supplied key is empty random one is generated and stored in key variable.

5.174.2.3 Arc::ThreadDataItem::ThreadDataItem (const std::string & *key*)

Creates instance and attaches it to current thread under key.

5.174.3 Member Function Documentation

5.174.3.1 void Arc::ThreadDataItem::Attach (const std::string & *key*)

Attaches object to current thread under key. This method must be used only if object was created using dummy constructor.

5.174.3.2 void Arc::ThreadDataItem::Attach (std::string & *key*)

Attaches object to current thread under key. If supplied key is empty random one is generated and stored in key variable. This method must be used only if object was created using dummy constructor.

5.174.3.3 virtual void Arc::ThreadDataItem::Dup (void) `[virtual]`

Creates copy of object. This method is called when new thread is created from current thread. It is called in new thread, so new object - if created - gets attached to new thread. If object is not meant to be inherited by new threads then this method should do nothing.

5.174.3.4 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

5.175 Arc::ThreadRegistry Class Reference

```
#include <Thread.h>
```

Public Member Functions

- void RegisterThread (void)
- void UnregisterThread (void)
- bool WaitOrCancel (int timeout)
- bool WaitForExit (int timeout=-1)

5.175.1 Detailed Description

This class is a set of conditions, mutexes, etc. conveniently exposed to monitor running child threads and to wait till they exit. There are no protections against race conditions. So use it carefully.

5.175.2 Member Function Documentation

5.175.2.1 void Arc::ThreadRegistry::RegisterThread (void)

Register thread as started/starting into this instance.

5.175.2.2 void Arc::ThreadRegistry::UnregisterThread (void)

Report thread as exited.

5.175.2.3 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.

5.175.2.4 bool Arc::ThreadRegistry::WaitOrCancel (int *timeout*)

Wait for timeout milliseconds or cancel request. Returns true if cancel request received.

The documentation for this class was generated from the following file:

- Thread.h

5.176 Arc::Time Class Reference

A class for storing and manipulating times.

```
#include <DateTime.h>
```

Public Member Functions

- Time ()
- Time (time_t)
- Time (time_t time, uint32_t nanosec)
- Time (const std::string &)
- Time & operator= (time_t)
- Time & operator= (const Time &)
- Time & operator= (const char ∗)
- Time & operator= (const std::string &)
- void SetTime (time_t)
- void SetTime (time_t time, uint32_t nanosec)
- time_t GetTime () const
- operator std::string () const
- std::string str (const TimeFormat &=time_format) const
- bool operator< (const Time &) const
- bool operator> (const Time &) const
- bool operator<= (const Time &) const
- bool operator>= (const Time &) const
- bool operator== (const Time &) const
- bool operator!= (const Time &) const
- Time operator+ (const Period &) const
- Time operator- (const Period &) const
- Period operator- (const Time &) const

Static Public Member Functions

- static void SetFormat (const TimeFormat &)
- static TimeFormat GetFormat ()

5.176.1 Detailed Description

A class for storing and manipulating times.

5.176.2 Constructor & Destructor Documentation

5.176.2.1 Arc::Time::Time ()

Default constructor. The time is put equal the current time.

5.176.2.2 Arc::Time::Time (time_t)

Constructor that takes a time_t variable and stores it.

5.176.2.3 Arc::Time::Time (time_t *time*, uint32_t *nanosec*)

Constructor that takes a fine grained time variables and stores them.

5.176.2.4 Arc::Time::Time (const std::string &)

Constructor that tries to convert a string into a time_t.

5.176.3 Member Function Documentation

5.176.3.1 static TimeFormat Arc::Time::GetFormat () `[static]`

Gets the default format for time strings.

5.176.3.2 time_t Arc::Time::GetTime () const

gets the time

5.176.3.3 Arc::Time::operator std::string () const

Returns a string representation of the time, using the default format.

5.176.3.4 bool Arc::Time::operator!= (const Time &) const

Comparing two Time objects.

5.176.3.5 Time Arc::Time::operator+ (const Period &) const

Adding Time object with Period object.

5.176.3.6 Period Arc::Time::operator- (const Time &) const

Subtracting Time object from the other Time object.

5.176.3.7 Time Arc::Time::operator- (const Period &) const

Subtracting Period object from Time object.

5.176.3.8 bool Arc::Time::operator< (const Time &) const

Comparing two Time objects.

5.176.3.9 bool Arc::Time::operator<= (const Time &) const

Comparing two Time objects.

5.176.3.10 Time& Arc::Time::operator= (const std::string &)

Assignment operator from a string.

5.176.3.11 Time& Arc::Time::operator= (const char ∗)

Assignment operator from a char pointer.

5.176.3.12 Time& Arc::Time::operator= (const Time &)

Assignment operator from a Time.

5.176.3.13 Time& Arc::Time::operator= (time_t)

Assignment operator from a time_t.

5.176.3.14 bool Arc::Time::operator== (const Time &) const

Comparing two Time objects.

5.176.3.15 bool Arc::Time::operator> (const Time &) const

Comparing two Time objects.

5.176.3.16 bool Arc::Time::operator>= (const Time &) const

Comparing two Time objects.

5.176.3.17 static void Arc::Time::SetFormat (const TimeFormat &) `[static]`

Sets the default format for time strings.

5.176.3.18 void Arc::Time::SetTime (time_t *time*, uint32_t *nanosec*)

sets the fine grained time

5.176.3.19 void Arc::Time::SetTime (time_t)

sets the time

5.176.3.20 std::string Arc::Time::str (const TimeFormat & = `time_format`) const

Returns a string representation of the time, using the specified format.

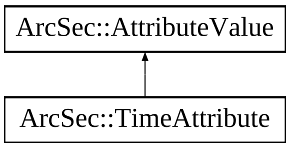
The documentation for this class was generated from the following file:

- DateTime.h

5.177 ArcSec::TimeAttribute Class Reference

`#include <DateTimeAttribute.h>`

Inheritance diagram for ArcSec::TimeAttribute::



Public Member Functions

- virtual bool equal (AttributeValue ∗other, bool check_id=true)
- virtual std::string encode ()
- virtual std::string getType ()
- virtual std::string getId ()

5.177.1 Detailed Description

Format: HHMMSSZ HH:MM:SS HH:MM:SS+HH:MM HH:MM:SSZ

5.177.2 Member Function Documentation

5.177.2.1 virtual std::string ArcSec::TimeAttribute::encode () `[virtual]`

encode the value in a string format

Implements ArcSec::AttributeValue.

5.177.2.2 virtual bool ArcSec::TimeAttribute::equal (AttributeValue ∗ *other*, bool *check_id* = `true`) `[virtual]`

Evluate whether "this" equale to the parameter value

Implements ArcSec::AttributeValue.

5.177.2.3 virtual std::string ArcSec::TimeAttribute::getId () `[inline, virtual]`

Get the AttributeId of the <Attribute>

Implements ArcSec::AttributeValue.

5.177.2.4 virtual std::string ArcSec::TimeAttribute::getType () `[inline, virtual]`

Get the DataType of the <Attribute>

Implements ArcSec::AttributeValue.

The documentation for this class was generated from the following file:

- DateTimeAttribute.h

5.178 DataStaging::TransferParameters Class Reference

Represents limits and properties of a DTR transfer.

```
#include <DTR.h>
```

Public Member Functions

- TransferParameters ()

Data Fields

- unsigned long long int min_average_bandwidth
- unsigned int max_inactivity_time
- unsigned long long int min_current_bandwidth
- unsigned int averaging_time
- unsigned long long int bytes_transferred
- Arc::Time start_time
- Arc::CheckSum $*$ checksum
- bool transfer_finished

5.178.1 Detailed Description

Represents limits and properties of a DTR transfer.

5.178.2 Constructor & Destructor Documentation

5.178.2.1 DataStaging::TransferParameters::TransferParameters () `[inline]`

Constructor. Initialises all values to zero.

5.178.3 Field Documentation

5.178.3.1 unsigned int DataStaging::TransferParameters::averaging_time

The time over which to average the calculation of min_curr_bandwidth.

5.178.3.2 unsigned long long int DataStaging::TransferParameters::bytes_transferred

Number of bytes transferred so far.

5.178.3.3 Arc::CheckSum$*$ DataStaging::TransferParameters::checksum

Pointer to checksum object.

5.178.3.4 unsigned int DataStaging::TransferParameters::max_inactivity_time

Maximum inactivity time in sec - if transfer stops for longer than this time it should be killed

5.178.3.5 unsigned long long int DataStaging::TransferParameters::min_average_bandwidth

Minimum average bandwidth in bytes/sec - if the average bandwidth used drops below this level the transfer should be killed

5.178.3.6 unsigned long long int DataStaging::TransferParameters::min_current_bandwidth

Minimum current bandwidth - if bandwidth averaged over averaging_time is less than minimum the transfer should be killed (allows transfers which slow down to be killed quicker)

5.178.3.7 Arc::Time DataStaging::TransferParameters::start_time

Time at which transfer started.

5.178.3.8 bool DataStaging::TransferParameters::transfer_finished

Flag to say whether transfer is complete (all bytes copied successfully).

The documentation for this class was generated from the following file:

- DTR.h

5.179 DataStaging::TransferShares Class Reference

TransferShares is used to implement fair-sharing and priorities.

`#include <TransferShares.h>`

Public Types

- USER
- VO
- GROUP
- ROLE
- NONE
- enum ShareType {
  USER, VO, GROUP, ROLE,
  NONE }

Public Member Functions

- TransferShares ()
- ~TransferShares ()
- TransferShares (const TransferShares &shares)
- TransferShares operator= (const TransferShares &shares)
- std::string extract_share_info (const DTR &DTRToExtract)
- void calculate_shares (int TotalNumberOfSlots)
- void increase_transfer_share (const std::string &ShareToIncrease)
- void decrease_transfer_share (const std::string &ShareToDecrease)
- void decrease_number_of_slots (const std::string &ShareToDecrease)
- bool can_start (const std::string &ShareToStart)
- bool is_configured (const std::string &ShareToCheck)
- int get_basic_priority (const std::string &ShareToCheck)
- void set_reference_share (const std::string &RefShare, int Priority)
- void set_reference_shares (const std::map< std::string, int > &shares)
- void set_share_type (ShareType Type)
- void set_share_type (const std::string &type)
- std::string conf () const

5.179.1 Detailed Description

TransferShares is used to implement fair-sharing and priorities.

TransferShares defines the algorithm used to prioritise and share transfers among different users or groups. It contains configuration configuration information on the share type and reference shares. The Scheduler uses TransferShares to determine which DTRs in the queue for Delivery go first to Delivery. The calculation is based on the configuration and the currently active shares (the DTRs already in Delivery). can_start() is the method called by the Scheduler to determine whether a particular share has an available slot in Delivery.

5.179.2 Member Enumeration Documentation

5.179.2.1 enum DataStaging::TransferShares::ShareType

The criterion for assigning a share to a DTR.

Enumerator:

USER Shares are defined per DN of the user's proxy.

VO Shares are defined per VOMS VO of the user's proxy.

GROUP Shares are defined per VOMS group of the user's proxy.

ROLE Shares are defined per VOMS role of the user's proxy.

NONE No share criterion - all DTRs will be assigned to a single share.

5.179.3 Constructor & Destructor Documentation

5.179.3.1 DataStaging::TransferShares::TransferShares ()

Create a new TransferShares with no configuration.

5.179.3.2 DataStaging::TransferShares::~TransferShares () `[inline]`

Empty destructor.

5.179.3.3 DataStaging::TransferShares::TransferShares (const TransferShares & *shares*)

Copy constructor must be defined because SimpleCondition cannot be copied.

5.179.4 Member Function Documentation

5.179.4.1 void DataStaging::TransferShares::calculate_shares (int *TotalNumberOfSlots*)

Calculate how many slots to assign to each active share.

This method is called each time the Scheduler loops to calculate the number of slots to assign to each share, based on the current number of active shares and the shares' relative priorities.

5.179.4.2 bool DataStaging::TransferShares::can_start (const std::string & *ShareToStart*)

Returns true if there is a slot available for the given share.

5.179.4.3 std::string DataStaging::TransferShares::conf () const

Return human-readable configuration of shares.

5.179.4.4 void DataStaging::TransferShares::decrease_number_of_slots (const std::string & *ShareToDecrease*)

Decrease by one the number of slots available to the given share.

Called when there is a Delivery slot already used by this share to reduce the number available.

5.179.4.5 void DataStaging::TransferShares::decrease_transfer_share (const std::string & *ShareToDecrease*)

Decrease by one the active count for the given share.

Called when a completed DTR leaves the system.

5.179.4.6 std::string DataStaging::TransferShares::extract_share_info (const DTR & *DTRToExtract*)

Get the name of the share the DTR should be assigned to.

5.179.4.7 int DataStaging::TransferShares::get_basic_priority (const std::string & *ShareToCheck*)

Get the priority of this share.

5.179.4.8 void DataStaging::TransferShares::increase_transfer_share (const std::string & *ShareToIncrease*)

Increase by one the active count for the given share.

Called when a new DTR enters the system.

5.179.4.9 bool DataStaging::TransferShares::is_configured (const std::string & *ShareToCheck*)

Returns true if the given share is a reference share.

5.179.4.10 TransferShares DataStaging::TransferShares::operator= (const TransferShares & *shares*)

Assignment operator must be defined because SimpleCondition cannot be copied.

5.179.4.11 void DataStaging::TransferShares::set_reference_share (const std::string & *RefShare*, int *Priority*)

Add a reference share.

5.179.4.12 void DataStaging::TransferShares::set_reference_shares (const std::map< std::string, int > & *shares*)

Set reference shares.

5.179.4.13 void DataStaging::TransferShares::set_share_type (const std::string & *type*)

Set the share type.

5.179.4.14 void DataStaging::TransferShares::set_share_type (ShareType *Type*)

Set the share type.

The documentation for this class was generated from the following file:

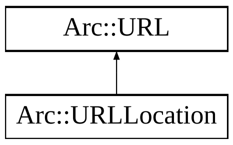
- TransferShares.h

5.180 Arc::URL Class Reference

Class to hold general URLs.

```
#include <URL.h>
```

Inheritance diagram for Arc::URL::



Public Types

- enum Scope

Public Member Functions

- URL ()
- URL (const std::string &url)
- virtual ~URL ()
- const std::string & Protocol () const
- void ChangeProtocol (const std::string &newprot)
- bool IsSecureProtocol () const
- const std::string & Username () const
- const std::string & Passwd () const
- const std::string & Host () const
- void ChangeHost (const std::string &newhost)
- int Port () const
- void ChangePort (int newport)
- const std::string & Path () const
- std::string FullPath () const
- std::string FullPathURIEncoded () const
- void ChangePath (const std::string &newpath)
- void ChangeFullPath (const std::string &newpath)
- const std::map< std::string, std::string > & HTTPOptions () const
- const std::string & HTTPOption (const std::string &option, const std::string &undefined="") const
- bool AddHTTPOption (const std::string &option, const std::string &value, bool overwrite=true)
- void RemoveHTTPOption (const std::string &option)
- const std::list< std::string > & LDAPAttributes () const
- void AddLDAPAttribute (const std::string &attribute)
- Scope LDAPScope () const
- void ChangeLDAPScope (const Scope newscope)
- const std::string & LDAPFilter () const
- void ChangeLDAPFilter (const std::string &newfilter)
- const std::map< std::string, std::string > & Options () const
- const std::string & Option (const std::string &option, const std::string &undefined="") const
- const std::map< std::string, std::string > & MetaDataOptions () const

- const std::string & MetaDataOption (const std::string &option, const std::string &undefined="") const
- bool AddOption (const std::string &option, const std::string &value, bool overwrite=true)
- bool AddOption (const std::string &option, bool overwrite=true)
- void AddMetaDataOption (const std::string &option, const std::string &value, bool overwrite=true)
- void AddLocation (const URLLocation &location)
- const std::list< URLLocation > & Locations () const
- const std::map< std::string, std::string > & CommonLocOptions () const
- const std::string & CommonLocOption (const std::string &option, const std::string &undefined="") const
- void RemoveOption (const std::string &option)
- void RemoveMetaDataOption (const std::string &option)
- virtual std::string str () const
- virtual std::string plainstr () const
- virtual std::string fullstr () const
- virtual std::string ConnectionURL () const
- bool operator< (const URL &url) const
- bool operator== (const URL &url) const
- operator bool () const
- bool StringMatches (const std::string &str) const
- std::map< std::string, std::string > ParseOptions (const std::string &optstring, char separator)

Static Public Member Functions

- static std::string OptionString (const std::map< std::string, std::string > &options, char separator)

Protected Member Functions

- void ParsePath (void)

Static Protected Member Functions

- static std::string BaseDN2Path (const std::string &)
- static std::string Path2BaseDN (const std::string &)

Protected Attributes

- std::string protocol
- std::string username
- std::string passwd
- std::string host
- bool ip6addr
- int port
- std::string path
- std::map< std::string, std::string > httpoptions
- std::map< std::string, std::string > metadataoptions
- std::list< std::string > ldapattributes
- Scope ldapscope
- std::string ldapfilter

- std::map< std::string, std::string > urloptions
- std::list< URLLocation > locations
- std::map< std::string, std::string > commonlocoptions
- bool valid

Friends

- std::ostream & operator<< (std::ostream &out, const URL &u)

5.180.1 Detailed Description

Class to hold general URLs.

The URL is split into protocol, hostname, port and path. This class tries to follow RFC 3986 for spliting URLs, at least for protocol + host part. It also accepts local file paths which are converted to file:path. The usual system dependent file paths are supported. Relative paths are converted to absolute paths by prepending them with current working directory path. A file path can't start from # symbol. If the string representation of URL starts from '@' then it is treated as path to a file containing a list of URLs.

A URL is parsed in the following way:

[protocol:][//[username:passwd@][host][:port]][;urloptions[;...]][/path[?httpoption[&...]][:metadataoption[:...]]]

The 'protocol' and 'host' parts are treated as case-insensitive and to avoid confusion are converted to lowercase in constructor. Note that 'path' is always converted to absolute path in constructor. The meaning of 'absolute' may depend upon URL type. For generic URL and local POSIX file paths that means path starts from / like

/path/to/file

For Windows paths absolute path may look like

C:\path\to\file

It is important to note that path still can be empty. For referencing local file using absolute path on POSIX filesystem one may use either

file:///path/to/file or file:/path/to/file

Relative path will look like

file:to/file

For local Windows files possible URLs are

file:C:\path\to\file or file:to\file

URLs representing LDAP resources have different structure of options following 'path' part:

ldap://host[:port][;urloptions[;...]][/path[?attributes[?scope[?filter]]]]

For LDAP URLs paths are converted from /key1=value1/.../keyN=valueN notation to keyN=valueN,...,key1=value1 and hence path does not contain leading /. If LDAP URL initially had path in second notation leading / is treated as separator only and is stripped.

URLs of indexing services optionally may have locations specified before 'host' part

protocol://[location[;location[;...]]@][host][:port]...

The structure of 'location' element is protocol specific.

5.180.2 Member Enumeration Documentation

5.180.2.1 enum Arc::URL::Scope

Scope for LDAP URLs

5.180.3 Constructor & Destructor Documentation

5.180.3.1 Arc::URL::URL ()

Empty constructor. Necessary when the class is part of another class and the like.

5.180.3.2 Arc::URL::URL (const std::string & *url*)

Constructs a new URL from a string representation.

5.180.3.3 virtual Arc::URL::~URL () `[virtual]`

URL Destructor

5.180.4 Member Function Documentation

5.180.4.1 bool Arc::URL::AddHTTPOption (const std::string & *option*, const std::string & *value*, bool *overwrite* = `true`)

Adds a HTP option with the given value. Returns false if overwrite is false and option already exists, true otherwise.

5.180.4.2 void Arc::URL::AddLDAPAttribute (const std::string & *attribute*)

Adds an LDAP attribute.

5.180.4.3 void Arc::URL::AddLocation (const URLLocation & *location*)

Adds a Location

5.180.4.4 void Arc::URL::AddMetaDataOption (const std::string & *option*, const std::string & *value*, bool *overwrite* = `true`)

Adds a metadata option

5.180.4.5 bool Arc::URL::AddOption (const std::string & *option*, bool *overwrite* = `true`)

Adds a URL option where option has the format "name=value". Returns false if overwrite is true and option already exists or if option does not have the correct format. Returns true otherwise.

5.180.4.6 bool Arc::URL::AddOption (const std::string & *option*, const std::string & *value*, bool *overwrite* = `true`)

Adds a URL option with the given value. Returns false if overwrite is false and option already exists, true otherwise. Note that some compilers may interpret AddOption("name", "value") as a call to AddOption(string, bool) so it is recommended to use explicit string types when calling this method.

5.180.4.7 static std::string Arc::URL::BaseDN2Path (const std::string &) `[static, protected]`

a private method that converts an ldap basedn to a path.

5.180.4.8 void Arc::URL::ChangeFullPath (const std::string & *newpath*)

Changes the path of the URL and all options attached.

5.180.4.9 void Arc::URL::ChangeHost (const std::string & *newhost*)

Changes the hostname of the URL.

5.180.4.10 void Arc::URL::ChangeLDAPFilter (const std::string & *newfilter*)

Changes the LDAP filter.

5.180.4.11 void Arc::URL::ChangeLDAPScope (const Scope *newscope*)

Changes the LDAP scope.

5.180.4.12 void Arc::URL::ChangePath (const std::string & *newpath*)

Changes the path of the URL.

5.180.4.13 void Arc::URL::ChangePort (int *newport*)

Changes the port of the URL.

5.180.4.14 void Arc::URL::ChangeProtocol (const std::string & *newprot*)

Changes the protocol of the URL.

5.180.4.15 const std::string& Arc::URL::CommonLocOption (const std::string & *option*, const std::string & *undefined* = "") const

Returns the value of a common location option.

Parameters:

option The option whose value is returned.

undefined This value is returned if the common location option is not defined.

5.180.4.16 const std::map<std::string, std::string>& Arc::URL::CommonLocOptions () const

Returns the common location options if any.

5.180.4.17 virtual std::string Arc::URL::ConnectionURL () const `[virtual]`

Returns a string representation with protocol, host and port only

5.180.4.18 std::string Arc::URL::FullPath () const

Returns the path of the URL with all options attached.

5.180.4.19 std::string Arc::URL::FullPathURIEncoded () const

Returns the path and all options, URI-encoded according to RFC 3986. Forward slashes ('/') in the path are not encoded but are encoded in the options.

5.180.4.20 virtual std::string Arc::URL::fullstr () const `[virtual]`

Returns a string representation including options and locations

Reimplemented in Arc::URLLocation.

5.180.4.21 const std::string& Arc::URL::Host () const

Returns the hostname of the URL.

5.180.4.22 const std::string& Arc::URL::HTTPOption (const std::string & *option*, const std::string & *undefined* = "") const

Returns the value of an HTTP option.

Parameters:

option The option whose value is returned.

undefined This value is returned if the HTTP option is not defined.

5.180.4.23 const std::map<std::string, std::string>& Arc::URL::HTTPOptions () const

Returns HTTP options if any.

5.180.4.24 bool Arc::URL::IsSecureProtocol () const

Indicates whether the protocol is secure or not.

5.180.4.25 const std::list<std::string>& Arc::URL::LDAPAttributes () const

Returns the LDAP attributes if any.

5.180.4.26 const std::string& Arc::URL::LDAPFilter () const

Returns the LDAP filter.

5.180.4.27 Scope Arc::URL::LDAPScope () const

Returns the LDAP scope.

5.180.4.28 const std::list<URLLocation>& Arc::URL::Locations () const

Returns the locations if any.

5.180.4.29 const std::string& Arc::URL::MetaDataOption (const std::string & *option*, const std::string & *undefined* = "") const

Returns the value of a metadata option.

Parameters:

option	The option whose value is returned.
undefined	This value is returned if the metadata option is not defined.

5.180.4.30 const std::map<std::string, std::string>& Arc::URL::MetaDataOptions () const

Returns metadata options if any.

5.180.4.31 Arc::URL::operator bool () const

Check if instance holds valid URL

5.180.4.32 bool Arc::URL::operator< (const URL & *url*) const

Compares one URL to another

5.180.4.33 bool Arc::URL::operator== (const URL & *url*) const

Is one URL equal to another?

5.180.4.34 const std::string& Arc::URL::Option (const std::string & *option*, const std::string & *undefined* = "") const

Returns the value of a URL option.

Parameters:

option The option whose value is returned.

undefined This value is returned if the URL option is not defined.

5.180.4.35 const std::map<std::string, std::string>& Arc::URL::Options () const

Returns URL options if any.

5.180.4.36 static std::string Arc::URL::OptionString (const std::map< std::string, std::string > & *options*, char *separator*) `[static]`

Returns a string representation of the options given in the options map

5.180.4.37 std::map<std::string, std::string> Arc::URL::ParseOptions (const std::string & *optstring*, char *separator*)

Parse a string of options separated by separator into an attribute->value map

5.180.4.38 void Arc::URL::ParsePath (void) `[protected]`

Convenience method for spliting schema specific part into path and options

5.180.4.39 const std::string& Arc::URL::Passwd () const

Returns the password of the URL.

5.180.4.40 const std::string& Arc::URL::Path () const

Returns the path of the URL.

5.180.4.41 static std::string Arc::URL::Path2BaseDN (const std::string &) `[static, protected]`

a private method that converts an ldap path to a basedn.

5.180.4.42 virtual std::string Arc::URL::plainstr () const `[virtual]`

Returns a string representation of the URL without any options

5.180.4.43 int Arc::URL::Port () const

Returns the port of the URL.

5.180.4.44 const std::string& Arc::URL::Protocol () const

Returns the protocol of the URL.

5.180.4.45 void Arc::URL::RemoveHTTPOption (const std::string & *option*)

Removes a HTTP option if exists.

Parameters:

option The option to remove.

5.180.4.46 void Arc::URL::RemoveMetaDataOption (const std::string & *option*)

Remove a metadata option if exits.

Parameters:

option The option to remove.

5.180.4.47 void Arc::URL::RemoveOption (const std::string & *option*)

Removes a URL option if exists.

Parameters:

option The option to remove.

5.180.4.48 virtual std::string Arc::URL::str () const `[virtual]`

Returns a string representation of the URL including meta-options.

Reimplemented in Arc::URLLocation.

5.180.4.49 bool Arc::URL::StringMatches (const std::string & *str*) const

Returns true if string matches url.

5.180.4.50 const std::string& Arc::URL::Username () const

Returns the username of the URL.

5.180.5 Friends And Related Function Documentation

5.180.5.1 std::ostream& operator<< (std::ostream & *out*, const URL & *u*) `[friend]`

Overloaded operator << to print a URL.

5.180.6 Field Documentation

5.180.6.1 std::map<std::string, std::string> Arc::URL::commonlocoptions `[protected]`

common location options for index server URLs.

5.180.6.2 std::string Arc::URL::host `[protected]`

hostname of the url.

5.180.6.3 std::map<std::string, std::string> Arc::URL::httpoptions `[protected]`

HTTP options of the url.

5.180.6.4 bool Arc::URL::ip6addr `[protected]`

if host is IPv6 numerical address notation.

5.180.6.5 std::list<std::string> Arc::URL::ldapattributes `[protected]`

LDAP attributes of the url.

5.180.6.6 std::string Arc::URL::ldapfilter `[protected]`

LDAP filter of the url.

5.180.6.7 Scope Arc::URL::ldapscope `[protected]`

LDAP scope of the url.

5.180.6.8 std::list<URLLocation> Arc::URL::locations `[protected]`

locations for index server URLs.

5.180.6.9 std::map<std::string, std::string> Arc::URL::metadataoptions `[protected]`

Meta data options

5.180.6.10 std::string Arc::URL::passwd `[protected]`

password of the url.

5.180.6.11 std::string Arc::URL::path `[protected]`

the url path.

5.180.6.12 int Arc::URL::port `[protected]`

portnumber of the url.

5.180.6.13 std::string Arc::URL::protocol `[protected]`

the url protocol.

5.180.6.14 std::map<std::string, std::string> Arc::URL::urloptions `[protected]`

options of the url.

5.180.6.15 std::string Arc::URL::username `[protected]`

username of the url.

5.180.6.16 bool Arc::URL::valid `[protected]`

flag to describe validity of URL

The documentation for this class was generated from the following file:

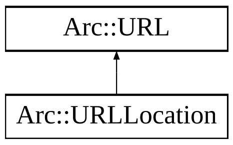
- URL.h

5.181 Arc::URLLocation Class Reference

Class to hold a resolved URL location.

```
#include <URL.h>
```

Inheritance diagram for Arc::URLLocation::



Public Member Functions

- URLLocation (const std::string &url="")
- URLLocation (const std::string &url, const std::string &name)
- URLLocation (const URL &url)
- URLLocation (const URL &url, const std::string &name)
- URLLocation (const std::map< std::string, std::string > &options, const std::string &name)
- virtual ~URLLocation ()
- const std::string & Name () const
- virtual std::string str () const
- virtual std::string fullstr () const

Protected Attributes

- std::string name

5.181.1 Detailed Description

Class to hold a resolved URL location.

It is specific to file indexing service registrations.

5.181.2 Constructor & Destructor Documentation

5.181.2.1 Arc::URLLocation::URLLocation (const std::string & *url* = " ")

Creates a URLLocation from a string representaion.

5.181.2.2 Arc::URLLocation::URLLocation (const std::string & *url*, const std::string & *name*)

Creates a URLLocation from a string representaion and a name.

5.181.2.3 Arc::URLLocation::URLLocation (const URL & *url*)

Creates a URLLocation from a URL.

5.181.2.4 Arc::URLLocation::URLLocation (const URL & *url*, const std::string & *name*)

Creates a URLLocation from a URL and a name.

5.181.2.5 Arc::URLLocation::URLLocation (const std::map< std::string, std::string > & *options*, const std::string & *name*)

Creates a URLLocation from options and a name.

5.181.2.6 virtual Arc::URLLocation::~URLLocation () `[virtual]`

URLLocation destructor.

5.181.3 Member Function Documentation

5.181.3.1 virtual std::string Arc::URLLocation::fullstr () const `[virtual]`

Returns a string representation including options and locations

Reimplemented from Arc::URL.

5.181.3.2 const std::string& Arc::URLLocation::Name () const

Returns the URLLocation name.

5.181.3.3 virtual std::string Arc::URLLocation::str () const `[virtual]`

Returns a string representation of the URLLocation.

Reimplemented from Arc::URL.

5.181.4 Field Documentation

5.181.4.1 std::string Arc::URLLocation::name `[protected]`

the URLLocation name as registered in the indexing service.

The documentation for this class was generated from the following file:

- URL.h

5.182 Arc::UserConfig Class Reference

User configuration class

`#include <UserConfig.h>`

Public Member Functions

- UserConfig (initializeCredentialsType initializeCredentials=initializeCredentialsType())
- UserConfig (const std::string &conffile, initializeCredentialsType initializeCredentials=initializeCredentialsType(), bool loadSysConfig=true)
- UserConfig (const std::string &conffile, const std::string &jfile, initializeCredentialsType initializeCredentials=initializeCredentialsType(), bool loadSysConfig=true)
- UserConfig (const long int &ptraddr)
- void InitializeCredentials ()
- bool CredentialsFound () const
- bool LoadConfigurationFile (const std::string &conffile, bool ignoreJobListFile=true)
- 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

5.182.1 Detailed Description

User configuration class

This class provides a container for a selection of various attributes/parameters which can be configured to needs of the user, and can be read by implementing instances or programs. The class can be used in two ways. One can create a object from a configuration file, or simply set the desired attributes by using the setter method, associated with every setable attribute. The list of attributes which can be configured in this class are:

- certificatepath / CertificatePath(const std::string&)

- keypath / KeyPath(const std::string&)
- proxypath / ProxyPath(const std::string&)
- cacertificatesdirectory / CACertificatesDirectory(const std::string&)
- cacertificatepath / CACertificatePath(const std::string&)
- timeout / Timeout(int)
- joblist / JobListFile(const std::string&)
- defaultservices / AddServices(const std::list<std::string>&, const std::list<std::string>&, ServiceType)
- rejectservices / AddServices(const std::list<std::string>&, const std::list<std::string>&, ServiceType)
- verbosity / Verbosity(const std::string&)
- brokername / Broker(const std::string&) or Broker(const std::string&, const std::string&)
- brokerarguments / Broker(const std::string&) or Broker(const std::string&, const std::string&)
- bartender / Bartender(const std::list<URL>&)
- vomsserverpath / VOMSServerPath(const std::string&)
- username / UserName(const std::string&)
- password / Password(const std::string&)
- keypassword / KeyPassword(const std::string&)
- keysize / KeySize(int)
- certificatelifetime / CertificateLifeTime(const Period&)
- slcs / SLCS(const URL&)
- storedirectory / StoreDirectory(const std::string&)
- jobdownloaddirectory / JobDownloadDirectory(const std::string&)
- idpname / IdPName(const std::string&)

where the first term is the name of the attribute used in the configuration file, and the second term is the associated setter method (for more information about a given attribute see the description of the setter method).

The configuration file should have a INI-style format and the IniConfig class will thus be used to parse the file. The above mentioned attributes should be placed in the common section. Another section is also valid in the configuration file, which is the alias section. Here it is possible to define aliases representing one or multiple services. These aliases can be used in the AddServices(const std::list<std::string>&, ServiceType) and AddServices(const std::list<std::string>&, const std::list<std::string>&, ServiceType) methods.

The UserConfig class also provides a method InitializeCredentials() for locating user credentials by searching in different standard locations. The CredentialsFound() method can be used to test if locating the credentials succeeded.

5.182.2 Constructor & Destructor Documentation

5.182.2.1 Arc::UserConfig::UserConfig (initializeCredentialsType *initializeCredentials* = `initializeCredentialsType()`)

Create a UserConfig object.

The UserConfig object created by this constructor initializes only default values, and if specified by the *initializeCredentials* boolean credentials will be tried initialized using the InitializeCredentials() method. The object is only non-valid if initialization of credentials fails which can be checked with the operator bool() method.

Parameters:

initializeCredentials is a optional boolean indicating if the InitializeCredentials() method should be invoked, the default is `true`.

See also:

InitializeCredentials()
operator bool()

5.182.2.2 Arc::UserConfig::UserConfig (const std::string & *conffile*, initializeCredentialsType *initializeCredentials* = `initializeCredentialsType()`, bool *loadSysConfig* = `true`)

Create a UserConfig object.

The UserConfig object created by this constructor will, if specified by the *loadSysConfig* boolean, first try to load the system configuration file by invoking the LoadConfigurationFile() method, and if this fails a WARNING is reported. Then the configuration file passed will be tried loaded using the before mentioned method, and if this fails an ERROR is reported, and the created object will be non-valid. Note that if the passed file path is empty the example configuration will be tried copied to the default configuration file path specified by DEFAULTCONFIG. If the example file cannot be copied one or more WARNING messages will be reported and no configration will be loaded. If loading the configurations file succeeded and if *initializeCredentials* is `true` then credentials will be initialized using the InitializeCredentials() method, and if no valid credentials are found the created object will be non-valid.

Parameters:

conffile is the path to a INI-configuration file.

initializeCredentials is a boolean indicating if credentials should be initialized, the default is `true`.

loadSysConfig is a boolean indicating if the system configuration file should be loaded aswell, the default is `true`.

See also:

LoadConfigurationFile(const std::string&, bool)
InitializeCredentials()
operator bool()
SYSCONFIG
EXAMPLECONFIG

5.182.2.3 Arc::UserConfig::UserConfig (const std::string & *conffile*, const std::string & *jfile*, initializeCredentialsType *initializeCredentials* = `initializeCredentialsType()`, bool *loadSysConfig* = `true`)

Create a UserConfig object.

The UserConfig object created by this constructor does only differ from the UserConfig(const std::string&, bool, bool) constructor in that it is possible to pass the path of the job list file directly to this constructor. If the job list file *joblistfile* is empty, the behaviour of this constructor is exactly the same as the before mentioned, otherwise the job list file will be initilized by invoking the setter method JobListFile(const std::string&). If it fails the created object will be non-valid, otherwise the specified configuration file *conffile* will be loaded with the *ignoreJobListFile* argument set to `true`.

Parameters:

conffile is the path to a INI-configuration file

jfile is the path to a (non-)existing job list file.

initializeCredentials is a boolean indicating if credentials should be initialized, the default is `true`.

loadSysConfig is a boolean indicating if the system configuration file should be loaded aswell, the default is `true`.

See also:

JobListFile(const std::string&)
LoadConfigurationFile(const std::string&, bool)
InitializeCredentials()
operator bool()

5.182.2.4 Arc::UserConfig::UserConfig (const long int & *ptraddr*)

Language binding constructor.

The passed long int should be a pointer address to a UserConfig object, and this address is then casted into this UserConfig object.

Parameters:

ptraddr is an memory address to a UserConfig object.

5.182.3 Member Function Documentation

5.182.3.1 void Arc::UserConfig::AddBartender (const URL & *url*) `[inline]`

Set bartenders, used to contact Chelonia.

Takes as input a Bartender URL and adds this to the list of bartenders.

Parameters:

url is a URL to be added to the list of bartenders.

See also:

Bartender(const std::list<URL>&)
Bartender() const

5.182.3.2 bool Arc::UserConfig::AddServices (const std::list< std::string > & *selected*, const std::list< std::string > & *rejected*, ServiceType *st*)

Add selected and rejected services.

The only diffence in behaviour of this method compared to the AddServices(const std::list<std::string>&, ServiceType) method is the input parameters and the format these parameters should follow. Instead of having an optional '-' in front of the string selected and rejected services should be specified in the two different arguments.

Two attributes are indirectly associated with this setter method 'defaultservices' and 'rejectservices'. The values specified with the 'defaultservices' attribute will be added to the list of selected services, and likewise with the 'rejectservices' attribute.

Parameters:

selected is a list of services which will be added to the selected services of this object.

rejected is a list of services which will be added to the rejected services of this object.

st specifies the ServiceType of the services to add.

Returns:

This method return `false` in case an alias cannot be resolved. In any other case `true` is returned.

See also:

AddServices(const std::list<std::string>&, ServiceType)
GetSelectedServices()
GetRejectedServices()
ClearSelectedServices()
ClearRejectedServices()
LoadConfigurationFile()

5.182.3.3 bool Arc::UserConfig::AddServices (const std::list< std::string > & *services*, ServiceType *st*)

Add selected and rejected services.

This method adds selected services and adds services to reject from the specified list *services*, which contains string objects. The syntax of a single element in the list must be expressed in the following two formats:

$$[-] < flavour >:< service\_url > |[-] < alias >$$

where the optional '-' indicate that the service should be added to the private list of services to reject. In the first format the <flavour> part indicates the type of ACC plugin to use when contacting the service, which is specified by the URL <service_url>, and in the second format the <alias> part specifies a alias defined in a parsed configuration file, note that the alias must not contain any of the charaters ':', '.', ' ' or '\t'. If a alias cannot be resolved an ERROR will be reported to the logger and the method will return false. If a element in the list *services* cannot be parsed an ERROR will be reported, and the element is skipped.

Two attributes are indirectly associated with this setter method 'defaultservices' and 'rejectservices'. The values specified with the 'defaultservices' attribute will be added to the list of selected services, and likewise with the 'rejectservices' attribute.

Parameters:

services is a list of services to either select or reject.

st indicates the type of the specfied services.

Returns:

This method returns `false` in case an alias cannot be resolved. In any other case `true` is returned.

See also:

AddServices(const std::string&, const std::string&, ServiceType)
GetSelectedServices()
GetRejectedServices()
ClearSelectedServices()
ClearRejectedServices()
LoadConfigurationFile()

5.182.3.4 void Arc::UserConfig::ApplyToConfig (BaseConfig & *ccfg*) const

Apply credentials to BaseConfig.

This methods sets the BaseConfig credentials to the credentials contained in this object. It also passes user defined configuration overlay if any.

See also:

InitializeCredentials()
CredentialsFound()
BaseConfig

Parameters:

ccfg a BaseConfig object which will configured with the credentials of this object.

5.182.3.5 const std::vector<URL>& Arc::UserConfig::Bartender () const `[inline]`

Get bartenders.

Returns a list of Bartender URLs

Returns:

The list of bartender URL objects is returned.

See also:

Bartender(const std::list<URL>&)
AddBartender(const URL&)

5.182.3.6 bool Arc::UserConfig::Bartender (const std::vector< URL > & *urls*) `[inline]`

Set bartenders, used to contact Chelonia.

Takes as input a vector of Bartender URLs.

The attribute associated with this setter method is 'bartender'.

Parameters:

urls is a list of URL object to be set as bartenders.

Returns:

This method always returns `true`.

See also:

AddBartender(const URL&)
Bartender() const

5.182.3.7 const std::pair<std::string, std::string>& Arc::UserConfig::Broker () const `[inline]`

Get the broker and corresponding arguments.

The returned pair contains the broker name as the first component and the argument as the second.

See also:

Broker(const std::string&)
Broker(const std::string&, const std::string&)
DEFAULT_BROKER

5.182.3.8 bool Arc::UserConfig::Broker (const std::string & *name*, const std::string & *argument*) `[inline]`

Set broker to use in target matching.

As opposed to the Broker(const std::string&) method this method sets broker name and arguments directly from the passed two arguments.

Two attributes are associated with this setter method 'brokername' and 'brokerarguments'.

Parameters:

name is the name of the broker.

argument is the arguments of the broker.

Returns:

This method always returns `true`.

See also:

Broker
Broker(const std::string&)
Broker() const
DEFAULT_BROKER

5.182.3.9 bool Arc::UserConfig::Broker (const std::string & *name*)

Set broker to use in target matching.

The string passed to this method should be in the format:

$$< name > [:< argument >]$$

where the <name> is the name of the broker and cannot contain any ':', and the optional <argument> should contain arguments which should be passed to the broker.

Two attributes are associated with this setter method 'brokername' and 'brokerarguments'.

Parameters:

name the broker name and argument specified in the format given above.

Returns:

This method allways returns `true`.

See also:

Broker
Broker(const std::string&, const std::string&)
Broker() const
DEFAULT_BROKER

5.182.3.10 const std::string& Arc::UserConfig::CACertificatePath () const `[inline]`

Get path to CA-certificate.

Retrieve the path to the file containing CA-certificate. This configuration parameter is deprecated.

Returns:

The path to the CA-certificate is returned.

See also:

CACertificatePath(const std::string&)

5.182.3.11 bool Arc::UserConfig::CACertificatePath (const std::string & *newCACertificatePath*) `[inline]`

Set CA-certificate path.

The path to the file containing CA-certificate will be set when calling this method. This configuration parameter is deprecated - use CACertificatesDirectory instead. Only arcslcs uses it.

The attribute associated with this setter method is 'cacertificatepath'.

Parameters:

newCACertificatePath is the path to the CA-certificate.

Returns:

This method always returns `true`.

See also:

CACertificatePath() const

5.182.3.12 const std::string& Arc::UserConfig::CACertificatesDirectory () const `[inline]`

Get path to CA-certificate directory.

Retrieve the path to the CA-certificate directory.

Returns:

The path to the CA-certificate directory is returned.

See also:

InitializeCredentials()
CredentialsFound() const
CACertificatesDirectory(const std::string&)

5.182.3.13 bool Arc::UserConfig::CACertificatesDirectory (const std::string & *newCACertificatesDirectory*) `[inline]`

Set path to CA-certificate directory.

The path to the directory containing CA-certificates will be set when calling this method. Note that the InitializeCredentials() method will also try to set this path, by searching in different locations.

The attribute associated with this setter method is 'cacertificatesdirectory'.

Parameters:

newCACertificatesDirectory is the path to the CA-certificate directory.

Returns:

This method always returns `true`.

See also:

InitializeCredentials()
CredentialsFound() const
CACertificatesDirectory() const

5.182.3.14 const Period& Arc::UserConfig::CertificateLifeTime () const `[inline]`

Get certificate life time.

Gets lifetime of user certificate which will be obtained from Short Lived Credentials Service.

Returns:

The certificate life time is returned as a Period object.

See also:

CertificateLifeTime(const Period&)

5.182.3.15 bool Arc::UserConfig::CertificateLifeTime (const Period & *newCertificateLifeTime*) [inline]

Set certificate life time.

Sets lifetime of user certificate which will be obtained from Short Lived Credentials Service.

The attribute associated with this setter method is 'certificatelifetime'.

Parameters:

newCertificateLifeTime is the life time of a certificate, as a Period object.

Returns:

This method always returns `true`.

See also:

CertificateLifeTime() const

5.182.3.16 const std::string& Arc::UserConfig::CertificatePath () const [inline]

Get path to certificate.

The path to the cerficate is returned when invoking this method.

Returns:

The certificate path is returned.

See also:

InitializeCredentials()
CredentialsFound() const
CertificatePath(const std::string&)
KeyPath() const

5.182.3.17 bool Arc::UserConfig::CertificatePath (const std::string & *newCertificatePath*) [inline]

Set path to certificate.

The path to user certificate will be set by this method. The path to the correcsponding key can be set with the KeyPath(const std::string&) method. Note that the InitializeCredentials() method will also try to set this path, by searching in different locations.

The attribute associated with this setter method is 'certificatepath'.

Parameters:

newCertificatePath is the path to the new certificate.

Returns:

This method always returns `true`.

See also:

InitializeCredentials()
CredentialsFound() const
CertificatePath() const
KeyPath(const std::string&)

5.182.3.18 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()
ClearSelectedServices(ServiceType)
AddServices(const std::list<std::string>&, ServiceType)
AddServices(const std::list<std::string>&, const std::list<std::string>&, ServiceType)
GetRejectedServices()

5.182.3.19 void Arc::UserConfig::ClearRejectedServices ()

Clear selected services.

Calling this method will cause the internally stored rejected services to be cleared.

See also:

ClearRejectedServices(ServiceType)
ClearSelectedServices()
AddServices(const std::list<std::string>&, ServiceType)
AddServices(const std::list<std::string>&, const std::list<std::string>&, ServiceType)
GetRejectedServices()

5.182.3.20 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()
ClearRejectedServices(ServiceType)
AddServices(const std::list<std::string>&, ServiceType)
AddServices(const std::list<std::string>&, const std::list<std::string>&, ServiceType)
GetSelectedServices()

5.182.3.21 void Arc::UserConfig::ClearSelectedServices ()

Clear selected services.

Calling this method will cause the internally stored selected services to be cleared.

See also:

ClearSelectedServices(ServiceType)
ClearRejectedServices()
AddServices(const std::list<std::string>&, ServiceType)
AddServices(const std::list<std::string>&, const std::list<std::string>&, ServiceType)
GetSelectedServices()

5.182.3.22 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()

5.182.3.23 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)
AddServices(const std::list<std::string>&, const std::list<std::string>&, ServiceType)
GetSelectedServices(ServiceType)
ClearRejectedServices()

5.182.3.24 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)
> AddServices(const std::list<std::string>&, const std::list<std::string>&, ServiceType)
> GetRejectedServices(ServiceType) const
> ClearSelectedServices()

5.182.3.25 const User& Arc::UserConfig::GetUser () const `[inline]`

Get User for filesystem access.

Returns:

> The user identity to use for file system access

See also:

> SetUser(const User&)

5.182.3.26 const std::string& Arc::UserConfig::IdPName () const `[inline]`

Get IdP name.

Gets Identity Provider name (Shibboleth) to which user belongs.

Returns:

> The IdP name

See also:

> IdPName(const std::string&)

5.182.3.27 bool Arc::UserConfig::IdPName (const std::string & *name*) `[inline]`

Set IdP name.

Sets Identity Provider name (Shibboleth) to which user belongs. It is used for contacting Short Lived Certificate Service.

The attribute associated with this setter method is 'idpname'.

Parameters:

name is the new IdP name.

Returns:

This method always returns `true`.

See also:

5.182.3.28 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&)
- Key/certificate path specified in either configuration file passed to the constructor or explicitly set using the setter methods KeyPath(const std::string&) and CertificatePath(const std::string&)
- 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().
- 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(), with 'etc' and 'certificates' separated by the directory delimeter.
- Path created by concatenating the output of ArcLocation::Get(), with 'etc', 'grid-security' and 'certificates' separated by the directory delimeter.

- Path created by concatenating the output of ArcLocation::Get(), 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()
ProxyPath(const std::string&)
KeyPath(const std::string&)
CertificatePath(const std::string&)
CACertificatesDirectory(const std::string&)

5.182.3.29 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:

5.182.3.30 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:

5.182.3.31 const std::string& Arc::UserConfig::JobListFile () const `[inline]`

Get a reference to the path of the job list file.

The job list file is used to store and fetch information about submitted computing jobs to computing services. This method will return the path to the specified job list file.

Returns:

The path to the job list file is returned.

See also:

JobListFile(const std::string&)

5.182.3.32 bool Arc::UserConfig::JobListFile (const std::string & *path*)

Set path to job list file.

The method takes a path to a file which will be used as the job list file for storing and reading job information. If the specified path *path* does not exist a empty job list file will be tried created. If creating the job list file in any way fails *false* will be returned and a ERROR message will be reported. Otherwise *true* is returned. If the directory containing the file does not exist, it will be tried created. The method will also return *false* if the file is not a regular file.

The attribute associated with this setter method is 'joblist'.

Parameters:

path the path to the job list file.

Returns:

If the job list file is a regular file or if it can be created *true* is returned, otherwise *false* is returned.

See also:

JobListFile() const

5.182.3.33 const std::string& Arc::UserConfig::KeyPassword () const `[inline]`

Get password for generated key.

Get password to be used to encode private key of credentials obtained from Short Lived Credentials Service.

Returns:

The key password is returned.

See also:

KeyPassword(const std::string&)
KeyPath() const
KeySize() const

5.182.3.34 bool Arc::UserConfig::KeyPassword (const std::string & *newKeyPassword*) `[inline]`

Set password for generated key.

Set password to be used to encode private key of credentials obtained from Short Lived Credentials Service.

The attribute associated with this setter method is 'keypassword'.

Parameters:

newKeyPassword is the new password to the key.

Returns:

This method always returns `true`.

See also:

KeyPassword() const
KeyPath(const std::string&)
KeySize(int)

5.182.3.35 const std::string& Arc::UserConfig::KeyPath () const `[inline]`

Get path to key.

The path to the key is returned when invoking this method.

Returns:

The path to the user key is returned.

See also:

InitializeCredentials()
CredentialsFound() const
KeyPath(const std::string&)
CertificatePath() const
KeyPassword() const
KeySize() const

5.182.3.36 bool Arc::UserConfig::KeyPath (const std::string & *newKeyPath*) `[inline]`

Set path to key.

The path to user key will be set by this method. The path to the corresponding certificate can be set with the CertificatePath(const std::string&) method. Note that the InitializeCredentials() method will also try to set this path, by searching in different locations.

The attribute associated with this setter method is 'keypath'.

Parameters:

newKeyPath is the path to the new key.

Returns:

This method always returns `true`.

See also:

InitializeCredentials()
CredentialsFound() const
KeyPath() const
CertificatePath(const std::string&)
KeyPassword(const std::string&)
KeySize(int)

5.182.3.37 int Arc::UserConfig::KeySize () const `[inline]`

Get key size.

Get size/strengt of private key of credentials obtained from Short Lived Credentials Service.

Returns:

The key size, as an integer, is returned.

See also:

KeySize(int)
KeyPath() const
KeyPassword() const

5.182.3.38 bool Arc::UserConfig::KeySize (int *newKeySize*) `[inline]`

Set key size.

Set size/strengt of private key of credentials obtained from Short Lived Credentials Service.

The attribute associated with this setter method is ’keysize’.

Parameters:

newKeySize is the size, an an integer, of the key.

Returns:

This method always returns `true`.

See also:

KeySize() const
KeyPath(const std::string&)
KeyPassword(const std::string&)

5.182.3.39 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 class. If the parsing is unsuccessful a WARNING is reported.

The format of the configuration file should follow that of INI, and every attribute present in the file is only allowed once, if otherwise a WARNING will be reported. The file can contain at most two sections, one named common and the other name alias. If other sections exist a WARNING will be reported. Only the following attributes is allowed in the common section of the configuration file:

- certificatepath (CertificatePath(const std::string&))
- keypath (KeyPath(const std::string&))
- proxypath (ProxyPath(const std::string&))
- cacertificatesdirectory (CACertificatesDirectory(const std::string&))
- cacertificatepath (CACertificatePath(const std::string&))
- timeout (Timeout(int))
- joblist (JobListFile(const std::string&))
- defaultservices (AddServices(const std::list<std::string>&, const std::list<std::string>&, ServiceType))
- rejectservices (AddServices(const std::list<std::string>&, const std::list<std::string>&, ServiceType))
- verbosity (Verbosity(const std::string&))
- brokername (Broker(const std::string&) or Broker(const std::string&, const std::string&))
- brokerarguments (Broker(const std::string&) or Broker(const std::string&, const std::string&))
- bartender (Bartender(const std::list<URL>&))
- vomsserverpath (VOMSServerPath(const std::string&))
- username (UserName(const std::string&))
- password (Password(const std::string&))
- keypassword (KeyPassword(const std::string&))
- keysize (KeySize(int))
- certificatelifetime (CertificateLifeTime(const Period&))
- slcs (SLCS(const URL&))
- storedirectory (StoreDirectory(const std::string&))
- jobdownloaddirectory (JobDownloadDirectory(const std::string&))
- idpname (IdPName(const std::string&))

where the 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) and AddServices(const std::list<std::string>&, const std::list<std::string>&, ServiceType) methods. An alias can not contain any of the characters '.', ':', ' ' or '\t' and should be defined as follows:

$$< alias\_name >=< service\_type >:< flavour >:< service\_url > | < alias\_ref > [...]$$

where <alias_name> is the name of the defined alias, <service_type> is the service type in lower case, <flavour> is the type of middleware plugin to use, <service_url> is the URL which should be used to contact the service and <alias_ref> is another defined alias. The parsed aliases will be stored internally and resolved when needed. If a alias already exist, and another alias with the same name is parsed then this other alias will overwrite the existing alias.

Parameters:

conffile is the path to the configuration file.

ignoreJobListFile is a optional boolean which indicates whether the joblistfile attribute in the configuration file should be ignored. Default is to ignored it (`true`).

Returns:

If loading the configuration file succeeds `true` is returned, otherwise `false` is returned.

See also:

SaveToFile()

5.182.3.40 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!()

5.182.3.41 bool Arc::UserConfig::operator! (void) const `[inline]`

Check for non-validity.

See operator bool() for a description.

See also:

operator bool()

5.182.3.42 const std::string& Arc::UserConfig::OverlayFile () const `[inline]`

Get path to configuration overlay file.

Returns:

The overlay file path

See also:

OverlayFile(const std::string&)

5.182.3.43 bool Arc::UserConfig::OverlayFile (const std::string & *path*) `[inline]`

Set path to configuration overlay file.

Content of specified file is a backdoor to configuration XML generated from information stored in this class. The content of file is passed to BaseConfig class in ApplyToConfig(BaseConfig&) then merged with internal configuration XML representation. This feature is meant for quick prototyping/testing/tuning of functionality without rewriting code. It is meant for developers and most users won't need it.

The attribute associated with this setter method is 'overlayfile'.

Parameters:

path is the new overlay file path.

Returns:

This method always returns `true`.

See also:

5.182.3.44 const std::string& Arc::UserConfig::Password () const `[inline]`

Get password.

Get password which is used for requesting credentials from Short Lived Credentials Service.

Returns:

The password is returned.

See also:

Password(const std::string&)

5.182.3.45 bool Arc::UserConfig::Password (const std::string & *newPassword*) `[inline]`

Set password.

Set password which is used for requesting credentials from Short Lived Credentials Service.

The attribute associated with this setter method is 'password'.

Parameters:

newPassword is the new password to set.

Returns:

This method always returns true.

See also:

Password() const

5.182.3.46 const std::string& Arc::UserConfig::ProxyPath () const `[inline]`

Get path to user proxy.

Retrieve path to user proxy.

Returns:

Returns the path to the user proxy.

See also:

ProxyPath(const std::string&)

5.182.3.47 bool Arc::UserConfig::ProxyPath (const std::string & *newProxyPath*) `[inline]`

Set path to user proxy.

This method will set the path of the user proxy. Note that the InitializeCredentials() method will also try to set this path, by searching in different locations.

The attribute associated with this setter method is 'proxypath'

Parameters:

newProxyPath is the path to a user proxy.

Returns:

This method always returns `true`.

See also:

InitializeCredentials()
CredentialsFound()
ProxyPath() const

5.182.3.48 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()

5.182.3.49 void Arc::UserConfig::SetUser (const User & *u*) `[inline]`

Set User for filesystem access.

Sometimes it is desirable to use the identity of another user when accessing the filesystem. This user can be specified through this method. By default this user is the same as the user running the process.

Parameters:

u User identity to use

5.182.3.50 const URL& Arc::UserConfig::SLCS () const `[inline]`

Get the URL to the Short Lived Certificate Service (SLCS).

Returns:

The SLCS is returned.

See also:

SLCS(const URL&)

5.182.3.51 bool Arc::UserConfig::SLCS (const URL & *newSLCS*) `[inline]`

Set the URL to the Short Lived Certificate Service (SLCS).

The attribute associated with this setter method is 'slcs'.

Parameters:

newSLCS is the URL to the SLCS

Returns:

This method always returns `true`.

See also:

SLCS() const

5.182.3.52 const std::string& Arc::UserConfig::StoreDirectory () const `[inline]`

Get store diretory.

Sets directory which is used to store credentials obtained from Short Lived Credential Servide.

Returns:

The path to the store directory is returned.

See also:

StoreDirectory(const std::string&)

5.182.3.53 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 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:

5.182.3.54 int Arc::UserConfig::Timeout () const `[inline]`

Get timeout.

Returns the timeout in seconds.

Returns:

timeout in seconds.

See also:

Timeout(int)
DEFAULT_TIMEOUT

5.182.3.55 bool Arc::UserConfig::Timeout (int *newTimeout*)

Set timeout.

When communicating with a service the timeout specifies how long, in seconds, the communicating instance should wait for a response. If the response have not been recieved before this period in time, the connection is typically dropped, and an error will be reported.

This method will set the timeout to the specified integer. If the passed integer is less than or equal to 0 then `false` is returned and the timeout will not be set, otherwise `true` is returned and the timeout will be set to the new value.

The attribute associated with this setter method is 'timeout'.

Parameters:

newTimeout the new timeout value in seconds.

Returns:

`false` in case *newTimeout* <= 0, otherwise `true`.

See also:

Timeout() const
DEFAULT_TIMEOUT

5.182.3.56 const std::string& Arc::UserConfig::UserName () const `[inline]`

Get user-name.

Get username which is used for requesting credentials from Short Lived Credentials Service.

Returns:

The username is returned.

See also:

UserName(const std::string&)

5.182.3.57 bool Arc::UserConfig::UserName (const std::string & *name*) `[inline]`

Set user-name for SLCS.

Set username which is used for requesting credentials from Short Lived Credentials Service.

The attribute associated with this setter method is 'username'.

Parameters:

name is the name of the user.

Returns:

This method always return true.

See also:

UserName() const

5.182.3.58 const std::string& Arc::UserConfig::UtilsDirPath () const `[inline]`

Get path to directory storing utility files for DataPoints.

Returns:

The utils dir path

See also:

UtilsDirPath(const std::string&)

5.182.3.59 bool Arc::UserConfig::UtilsDirPath (const std::string & *dir*)

Set path to directory storing utility files for DataPoints.

Some DataPoints can store information on remote services in local files. This method sets the path to the directory containing these files. For example arc∗ tools set it to ARCUSERDIRECTORY and A-REX sets it to the control directory. The directory is created if it does not exist.

Parameters:

path is the new utils dir path.

Returns:

This method always returns `true`.

5.182.3.60 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&)

5.182.3.61 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

5.182.3.62 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&)

5.182.3.63 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

5.182.4 Field Documentation

5.182.4.1 const std::string Arc::UserConfig::ARCUSERDIRECTORY `[static]`

Path to ARC user home directory.

The *ARCUSERDIRECTORY* variable is the path to the ARC home directory of the current user. This path is created using the User::Home() method.

See also:

User::Home()

5.182.4.2 const std::string Arc::UserConfig::DEFAULT_BROKER `[static]`

Default broker.

The *DEFAULT_BROKER* specifies the name of the broker which should be used in case no broker is explicitly chosen.

See also:

Broker
Broker(const std::string&)
Broker(const std::string&, const std::string&)
Broker() const

5.182.4.3 const int Arc::UserConfig::DEFAULT_TIMEOUT = 20 `[static]`

Default timeout in seconds.

The *DEFAULT_TIMEOUT* specifies interval which will be used in case no timeout interval have been explicitly specified. For a description about timeout see Timeout(int).

See also:

Timeout(int)
Timeout() const

5.182.4.4 const std::string Arc::UserConfig::DEFAULTCONFIG `[static]`

Path to default configuration file.

The *DEFAULTCONFIG* variable is the path to the default configuration file used in case no configuration file have been specified. The path is created from the ARCUSERDIRECTORY object.

5.182.4.5 const std::string Arc::UserConfig::EXAMPLECONFIG `[static]`

Path to example configuration.

The *EXAMPLECONFIG* variable is the path to the example configuration file.

5.182.4.6 const std::string Arc::UserConfig::SYSCONFIG `[static]`

Path to system configuration.

The *SYSCONFIG* variable is the path to the system configuration file. This variable is only equal to SYSCONFIGARCLOC if ARC is installed in the root (highly unlikely).

5.182.4.7 const std::string Arc::UserConfig::SYSCONFIGARCLOC `[static]`

Path to system configuration at ARC location.

The *SYSCONFIGARCLOC* variable is the path to the system configuration file which reside at the ARC installation location.

The documentation for this class was generated from the following file:

- UserConfig.h

5.183 Arc::UsernameToken Class Reference

Interface for manipulation of WS-Security according to Username Token Profile.

`#include <UsernameToken.h>`

Public Types

- enum PasswordType

Public Member Functions

- UsernameToken (SOAPEnvelope &soap)
- UsernameToken (SOAPEnvelope &soap, const std::string &username, const std::string &password, const std::string &uid, PasswordType pwdtype)
- UsernameToken (SOAPEnvelope &soap, const std::string &username, const std::string &id, bool mac, int iteration)
- operator bool (void)
- std::string Username (void)
- bool Authenticate (const std::string &password, std::string &derived_key)
- bool Authenticate (std::istream &password, std::string &derived_key)

5.183.1 Detailed Description

Interface for manipulation of WS-Security according to Username Token Profile.

5.183.2 Member Enumeration Documentation

5.183.2.1 enum Arc::UsernameToken::PasswordType

SOAP header element

5.183.3 Constructor & Destructor Documentation

5.183.3.1 Arc::UsernameToken::UsernameToken (SOAPEnvelope & *soap*)

Link to existing SOAP header and parse Username Token information. Username Token related information is extracted from SOAP header and stored in class variables.

5.183.3.2 Arc::UsernameToken::UsernameToken (SOAPEnvelope & *soap*, const std::string & *username*, const std::string & *password*, const std::string & *uid*, PasswordType *pwdtype*)

Add Username Token information into the SOAP header. Generated token contains elements Username and Password and is meant to be used for authentication.

Parameters:

soap	the SOAP message
username	<wsse:Username>...</wsse:Username> - if empty it is entered interactively from stdin

password <wsse:Password Type="...">...</wsse:Password> - if empty it is entered interactively from stdin

uid <wsse:UsernameToken wsu:ID="...">

pwdtype <wsse:Password Type="...">...</wsse:Password>

5.183.3.3 Arc::UsernameToken::UsernameToken (SOAPEnvelope & *soap*, const std::string & *username*, const std::string & *id*, bool *mac*, int *iteration*)

Add Username Token information into the SOAP header. Generated token contains elements Username and Salt and is meant to be used for deriving Key Derivation.

Parameters:

soap the SOAP message

username <wsse:Username>...</wsse:Username>

mac if derived key is meant to be used for Message Authentication Code

iteration <wsse11:Iteration>...</wsse11:Iteration>

5.183.4 Member Function Documentation

5.183.4.1 bool Arc::UsernameToken::Authenticate (std::istream & *password*, std::string & *derived_key*)

Checks parsed token against password stored in specified stream. If token is meant to be used for deriving a key then key is returned in derived_key

5.183.4.2 bool Arc::UsernameToken::Authenticate (const std::string & *password*, std::string & *derived_key*)

Checks parsed/generated token against specified password. If token is meant to be used for deriving a key then key is returned in derived_key. In that case authentication is performed outside of UsernameToken class using obtained derived_key.

5.183.4.3 Arc::UsernameToken::operator bool (void)

Returns true of constructor succeeded

5.183.4.4 std::string Arc::UsernameToken::Username (void)

Returns username associated with this instance

The documentation for this class was generated from the following file:

- UsernameToken.h

5.184 Arc::UserSwitch Class Reference

```
#include <User.h>
```

5.184.1 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 class while spawning new process.

The documentation for this class was generated from the following file:

- User.h

5.185 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< VOMSTrust-Regex > &regexs)
- VOMSTrustChain & AddChain (const VOMSTrustChain &chain)
- VOMSTrustChain & AddChain (void)
- RegularExpression & AddRegex (const VOMSTrustRegex &reg)

5.185.1 Detailed Description

Stores definitions for making decision if VOMS server is trusted

5.185.2 Constructor & Destructor Documentation

5.185.2.1 Arc::VOMSTrustList::VOMSTrustList (const std::vector< std::string > & *encoded_list*)

Creates chain lists and regexps from plain list. List is made of chunks delimited by elements containing pattern "NEXT CHAIN". Each chunk with more than one element is converted into one instance of VOMSTrustChain. Chunks with single element are converted to VOMSTrustChain if element does not have special symbols. Otherwise it is treated as regular expression. Those symbols are '^','$' and '∗'. Trusted chains can be congicured in two ways: one way is: <tls:VOMSCertTrustDNChain> <tls:VOMSCertTrustDN>/O=Grid/O=NorduGrid/CN=host/arthur.hep.lu.se</tls:VOMSCertTrustDN> <tls:VOMSCertTrustDN>/O=Grid/O=NorduGrid/CN=NorduGrid Certification Authority</tls:VOMSCertTrustDN> <tls:VOMSCertTrustDN>—-NEXT CHAIN—</tls:VOMSCertTrustDN> <tls:VOMSCertTrustDN>/DC=ch/DC=cern/OU=computers/CN=voms.cern.ch</tls:VOMSCertTrustDN> <tls:VOMSCertTrustDN>/DC=ch/DC=cern/CN=CERN Trusted Certification Authority</tls:VOMSCertTrustDN> </tls:VOMSCertTrustDNChain> the other way is: <tls:VOMSCertTrustDNChain> <tls:VOMSCertTrustDN>/O=Grid/O=NorduGrid/CN=host/arthur.hep.lu.se</tls:VOMSCertTrustDN> <tls:VOMSCertTrustDN>/O=Grid/O=NorduGrid/CN=NorduGrid Certification Authority</tls:VOMSCertTrustDN> </tls:VOMSCertTrustDNChain> <tls:VOMSCertTrustDNChain> <tls:VOMSCertTrustDN>/DC=ch/DC=cern/OU=computers/CN=voms.cern.ch</tls:VOMSCertTrustDN> <tls:VOMSCertTrustDN>/DC=ch/DC=cern/CN=CERN Trusted Certification Authority</tls:VOMSCertTrustDN> </tls:VOMSCertTrustDNChain> each chunk is supposed to contain a suit of DN of trusted certificate chain, in which the first DN is the DN of the certificate (cert0) which is used to sign the Attribute Certificate (AC), the second DN is the DN of the issuer certificate(cert1) which is used to sign cert0. So if there are one or more intermediate issuers, then there should be 3 or more than 3 DNs in this chunk (considering cert0 and the root certificate, plus the intermediate certificate) .

5.185.2.2 Arc::VOMSTrustList::VOMSTrustList (const std::vector< VOMSTrustChain > & *chains*, const std::vector< VOMSTrustRegex > & *regexs*)

Creates chain lists and regexps from those specified in arguments. See AddChain() and AddRegex() for more information.

5.185.3 Member Function Documentation

5.185.3.1 VOMSTrustChain& Arc::VOMSTrustList::AddChain (void)

Adds empty chain of trusted DNs to list.

5.185.3.2 VOMSTrustChain& Arc::VOMSTrustList::AddChain (const VOMSTrustChain & *chain*)

Adds chain of trusted DNs to list. During verification each signature of AC is checked against all stored chains. DNs of chain of certificate used for signing AC are compared against DNs stored in these chains one by one. If needed DN of issuer of last certificate is checked too. Comparison succeeds if DNs in at least one stored chain are same as those in certificate chain. Comparison stops when all DNs in stored chain are compared. If there are more DNs in stored chain than in certificate chain then comparison fails. Empty stored list matches any certificate chain. Taking into account that certificate chains are verified down to trusted CA anyway, having more than one DN in stored chain seems to be useless. But such feature may be found useful by some very strict sysadmins. ??? IMO,DN list here is not only for authentication, it is also kind of ACL, which means the AC consumer only trusts those DNs which issues AC.

5.185.3.3 RegularExpression& Arc::VOMSTrustList::AddRegex (const VOMSTrustRegex & *reg*)

Adds regular expression to list. During verification each signature of AC is checked against all stored regular expressions. DN of signing certificate must match at least one of stored regular expressions.

The documentation for this class was generated from the following file:

- VOMSUtil.h

5.186 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)

5.186.1 Detailed Description

Interface for manipulation of WS-Adressing Endpoint Reference.

It works on Endpoint Reference stored in XML tree. No information is stored in this object except reference to corresponding XML subtree.

5.186.2 Constructor & Destructor Documentation

5.186.2.1 Arc::WSAEndpointReference::WSAEndpointReference (XMLNode *epr*)

Linking to existing EPR in XML tree

5.186.2.2 Arc::WSAEndpointReference::WSAEndpointReference (const WSAEndpointReference & *wsa*)

Copy constructor

5.186.2.3 Arc::WSAEndpointReference::WSAEndpointReference (const std::string & *address*)

Creating independent EPR - not implemented

5.186.2.4 Arc::WSAEndpointReference::WSAEndpointReference (void)

Dummy constructor - creates invalid instance

5.186.2.5 Arc::WSAEndpointReference::∼WSAEndpointReference (void)

Destructor. All empty elements of EPR XML are destroyed here too

5.186.3 Member Function Documentation

5.186.3.1 void Arc::WSAEndpointReference::Address (const std::string & *uri*)

Assigns new Address value. If EPR had no Address element it is created.

5.186.3.2 std::string Arc::WSAEndpointReference::Address (void) const

Returns Address (URL) encoded in EPR

5.186.3.3 XMLNode Arc::WSAEndpointReference::MetaData (void)

Access to MetaData element of EPR. Obtained XML element should be manipulated directly in application-dependent way. If EPR had no MetaData element it is created.

5.186.3.4 Arc::WSAEndpointReference::operator XMLNode (void)

Returns reference to EPR top XML node

5.186.3.5 WSAEndpointReference& Arc::WSAEndpointReference::operator= (const std::string & *address*)

Same as Address(uri)

5.186.3.6 XMLNode Arc::WSAEndpointReference::ReferenceParameters (void)

Access to ReferenceParameters element of EPR. Obtained XML element should be manipulated directly in application-dependent way. If EPR had no ReferenceParameters element it is created.

The documentation for this class was generated from the following file:

- WSA.h

5.187 Arc::WSAHeader Class Reference

Interface for manipulation WS-Addressing information in SOAP header.

`#include <WSA.h>`

Public Member Functions

- WSAHeader (SOAPEnvelope &soap)
- WSAHeader (const std::string &action)
- std::string To (void) const
- void To (const std::string &uri)
- WSAEndpointReference From (void)
- WSAEndpointReference ReplyTo (void)
- WSAEndpointReference FaultTo (void)
- std::string Action (void) const
- void Action (const std::string &uri)
- std::string MessageID (void) const
- void MessageID (const std::string &uri)
- std::string RelatesTo (void) const
- void RelatesTo (const std::string &uri)
- std::string RelationshipType (void) const
- void RelationshipType (const std::string &uri)
- XMLNode ReferenceParameter (int n)
- XMLNode ReferenceParameter (const std::string &name)
- XMLNode NewReferenceParameter (const std::string &name)
- operator XMLNode (void)

Static Public Member Functions

- static bool Check (SOAPEnvelope &soap)

Protected Attributes

- bool header_allocated_

5.187.1 Detailed Description

Interface for manipulation WS-Addressing information in SOAP header.

It works on Endpoint Reference stored in XML tree. No information is stored in this object except reference to corresponding XML subtree.

5.187.2 Constructor & Destructor Documentation

5.187.2.1 Arc::WSAHeader::WSAHeader (SOAPEnvelope & *soap*)

Linking to a header of existing SOAP message

5.187.2.2 Arc::WSAHeader::WSAHeader (const std::string & *action*)

Creating independent SOAP header - not implemented

5.187.3 Member Function Documentation

5.187.3.1 void Arc::WSAHeader::Action (const std::string & *uri*)

Set content of Action element of SOAP Header. If such element does not exist it's created.

5.187.3.2 std::string Arc::WSAHeader::Action (void) const

Returns content of Action element of SOAP Header.

5.187.3.3 static bool Arc::WSAHeader::Check (SOAPEnvelope & *soap*) `[static]`

Tells if specified SOAP message has WSA header

5.187.3.4 WSAEndpointReference Arc::WSAHeader::FaultTo (void)

Returns FaultTo element of SOAP Header. If such element does not exist it's created. Obtained element may be manipulted.

5.187.3.5 WSAEndpointReference Arc::WSAHeader::From (void)

Returns From element of SOAP Header. If such element does not exist it's created. Obtained element may be manipulted.

5.187.3.6 void Arc::WSAHeader::MessageID (const std::string & *uri*)

Set content of MessageID element of SOAP Header. If such element does not exist it's created.

5.187.3.7 std::string Arc::WSAHeader::MessageID (void) const

Returns content of MessageID element of SOAP Header.

5.187.3.8 XMLNode Arc::WSAHeader::NewReferenceParameter (const std::string & *name*)

Creates new ReferenceParameter element with specified name. Returns reference to created element.

5.187.3.9 Arc::WSAHeader::operator XMLNode (void)

Returns reference to SOAP Header - not implemented

5.187.3.10 XMLNode Arc::WSAHeader::ReferenceParameter (const std::string & *name*)

Returns first ReferenceParameter element with specified name

5.187.3.11 XMLNode Arc::WSAHeader::ReferenceParameter (int *n*)

Return n-th ReferenceParameter element

5.187.3.12 void Arc::WSAHeader::RelatesTo (const std::string & *uri*)

Set content of RelatesTo element of SOAP Header. If such element does not exist it's created.

5.187.3.13 std::string Arc::WSAHeader::RelatesTo (void) const

Returns content of RelatesTo element of SOAP Header.

5.187.3.14 void Arc::WSAHeader::RelationshipType (const std::string & *uri*)

Set content of RelationshipType element of SOAP Header. If such element does not exist it's created.

5.187.3.15 std::string Arc::WSAHeader::RelationshipType (void) const

Returns content of RelationshipType element of SOAP Header.

5.187.3.16 WSAEndpointReference Arc::WSAHeader::ReplyTo (void)

Returns ReplyTo element of SOAP Header. If such element does not exist it's created. Obtained element may be manipulted.

5.187.3.17 void Arc::WSAHeader::To (const std::string & *uri*)

Set content of To element of SOAP Header. If such element does not exist it's created.

5.187.3.18 std::string Arc::WSAHeader::To (void) const

Returns content of To element of SOAP Header.

5.187.4 Field Documentation

5.187.4.1 bool Arc::WSAHeader::header_allocated_ `[protected]`

SOAP header element

The documentation for this class was generated from the following file:

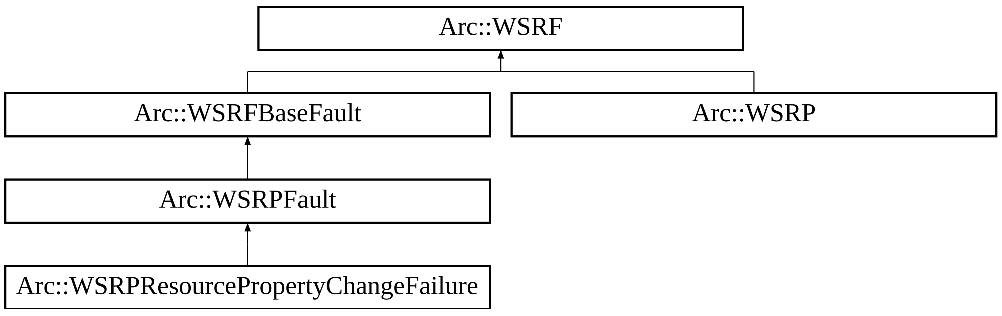
- WSA.h

5.188 Arc::WSRF Class Reference

Base class for every WSRF message.

`#include <WSRF.h>`

Inheritance diagram for Arc::WSRF::



Public Member Functions

- WSRF (SOAPEnvelope &soap, const std::string &action="")
- WSRF (bool fault=false, const std::string &action="")
- virtual SOAPEnvelope & SOAP (void)
- virtual operator bool (void)

Protected Member Functions

- void set_namespaces (void)

Protected Attributes

- bool allocated_
- bool valid_

5.188.1 Detailed Description

Base class for every WSRF message.

This class is not intended to be used directly. Use it like reference while passing through unknown WSRF message or use classes derived from it.

5.188.2 Constructor & Destructor Documentation

5.188.2.1 Arc::WSRF::WSRF (SOAPEnvelope & *soap*, const std::string & *action* = "")

Constructor - creates object out of supplied SOAP tree.

5.188.2.2 Arc::WSRF::WSRF (bool *fault* = `false`, const std::string & *action* = `""`)

Constructor - creates new WSRF object

5.188.3 Member Function Documentation

5.188.3.1 virtual Arc::WSRF::operator bool (void) `[inline, virtual]`

Returns true if instance is valid

5.188.3.2 void Arc::WSRF::set_namespaces (void) `[protected]`

set WS Resource namespaces and default prefixes in SOAP message

Reimplemented in Arc::WSRP, and Arc::WSRFBaseFault.

5.188.3.3 virtual SOAPEnvelope& Arc::WSRF::SOAP (void) `[inline, virtual]`

Direct access to underlying SOAP element

5.188.4 Field Documentation

5.188.4.1 bool Arc::WSRF::allocated_ `[protected]`

Associated SOAP message - it's SOAP message after all

5.188.4.2 bool Arc::WSRF::valid_ `[protected]`

true if soap_ needs to be deleted in destructor

The documentation for this class was generated from the following file:

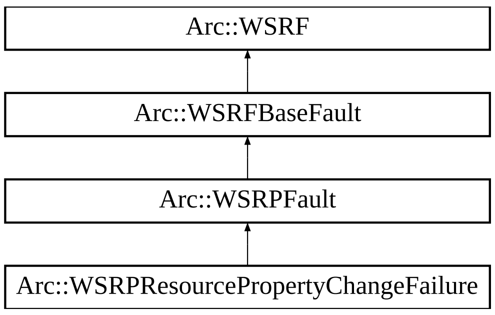
- WSRF.h

5.189 Arc::WSRFBaseFault Class Reference

Base class for WSRF fault messages.

`#include <WSRFBaseFault.h>`

Inheritance diagram for Arc::WSRFBaseFault::



Public Member Functions

- WSRFBaseFault (SOAPEnvelope &soap)
- WSRFBaseFault (const std::string &type)

Protected Member Functions

- void set_namespaces (void)

5.189.1 Detailed Description

Base class for WSRF fault messages.

Use classes inherited from it for specific faults.

5.189.2 Constructor & Destructor Documentation

5.189.2.1 Arc::WSRFBaseFault::WSRFBaseFault (SOAPEnvelope & *soap*)

Constructor - creates object out of supplied SOAP tree.

5.189.2.2 Arc::WSRFBaseFault::WSRFBaseFault (const std::string & *type*)

Constructor - creates new WSRF fault

5.189.3 Member Function Documentation

5.189.3.1 void Arc::WSRFBaseFault::set_namespaces (void) `[protected]`

set WS-ResourceProperties namespaces and default prefixes in SOAP message

Reimplemented from Arc::WSRF.

The documentation for this class was generated from the following file:

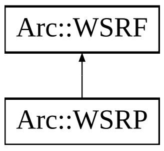
- WSRFBaseFault.h

5.190 Arc::WSRP Class Reference

Base class for WS-ResourceProperties structures.

`#include <WSResourceProperties.h>`

Inheritance diagram for Arc::WSRP::



Public Member Functions

- WSRP (bool fault=false, const std::string &action="")
- WSRP (SOAPEnvelope &soap, const std::string &action="")

Protected Member Functions

- void set_namespaces (void)

5.190.1 Detailed Description

Base class for WS-ResourceProperties structures.

Inheriting classes implement specific WS-ResourceProperties messages and their properties/elements. Refer to WS-ResourceProperties specifications for things specific to every message.

5.190.2 Constructor & Destructor Documentation

5.190.2.1 Arc::WSRP::WSRP (bool *fault* = `false`, const std::string & *action* = `""`)

Constructor - prepares object for creation of new WSRP request/response/fault

5.190.2.2 Arc::WSRP::WSRP (SOAPEnvelope & *soap*, const std::string & *action* = `""`)

Constructor - creates object out of supplied SOAP tree. It does not check if 'soap' represents valid WS-ResourceProperties structure. Actual check for validity of structure has to be done by derived class.

5.190.3 Member Function Documentation

5.190.3.1 void Arc::WSRP::set_namespaces (void) `[protected]`

set WS-ResourceProperties namespaces and default prefixes in SOAP message

Reimplemented from Arc::WSRF.

The documentation for this class was generated from the following file:

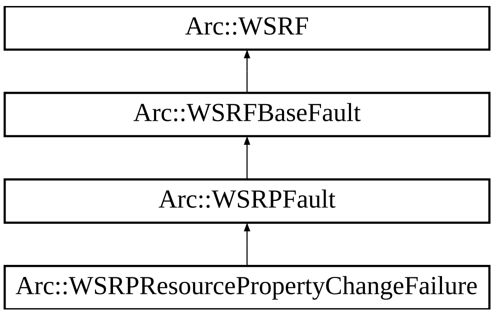
- WSResourceProperties.h

5.191 Arc::WSRPFault Class Reference

Base class for WS-ResourceProperties faults.

`#include <WSResourceProperties.h>`

Inheritance diagram for Arc::WSRPFault::



Public Member Functions

- WSRPFault (SOAPEnvelope &soap)
- WSRPFault (const std::string &type)

5.191.1 Detailed Description

Base class for WS-ResourceProperties faults.

5.191.2 Constructor & Destructor Documentation

5.191.2.1 Arc::WSRPFault::WSRPFault (SOAPEnvelope & *soap*)

Constructor - creates object out of supplied SOAP tree.

5.191.2.2 Arc::WSRPFault::WSRPFault (const std::string & *type*)

Constructor - creates new WSRP fault

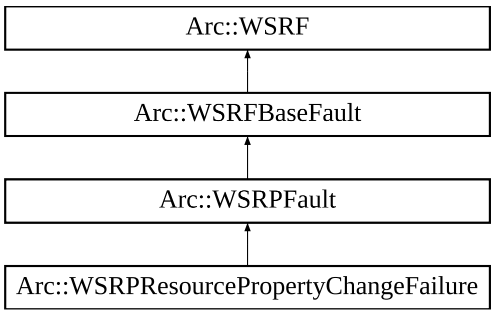
The documentation for this class was generated from the following file:

- WSResourceProperties.h

5.192 Arc::WSRPResourcePropertyChangeFailure Class Reference

`#include <WSResourceProperties.h>`

Inheritance diagram for Arc::WSRPResourcePropertyChangeFailure::



Public Member Functions

- WSRPResourcePropertyChangeFailure (SOAPEnvelope &soap)
- WSRPResourcePropertyChangeFailure (const std::string &type)

5.192.1 Detailed Description

Base class for WS-ResourceProperties faults which contain ResourcePropertyChangeFailure

5.192.2 Constructor & Destructor Documentation

5.192.2.1 Arc::WSRPResourcePropertyChangeFailure::WSRPResourcePropertyChangeFailure (SOAPEnvelope & *soap*) `[inline]`

Constructor - creates object out of supplied SOAP tree.

5.192.2.2 Arc::WSRPResourcePropertyChangeFailure::WSRPResourcePropertyChangeFailure (const std::string & *type*) `[inline]`

Constructor - creates new WSRP fault

The documentation for this class was generated from the following file:

- WSResourceProperties.h

5.193 Arc::X509Token Class Reference

Class for manipulating X.509 Token Profile.

`#include <X509Token.h>`

Public Types

- enum X509TokenType

Public Member Functions

- X509Token (SOAPEnvelope &soap, const std::string &keyfile="")
- X509Token (SOAPEnvelope &soap, const std::string &certfile, const std::string &keyfile, X509TokenType token_type=Signature)
- ~X509Token (void)
- operator bool (void)
- bool Authenticate (const std::string &cafile, const std::string &capath)
- bool Authenticate (void)

5.193.1 Detailed Description

Class for manipulating X.509 Token Profile.

This class is for generating/consuming X.509 Token profile. Currently it is used by x509token handler (src/hed/pdc/x509tokensh/) It is not necessary to directly called this class. If we need to use X.509 Token functionality, we only need to configure the x509token handler into service and client.

5.193.2 Member Enumeration Documentation

5.193.2.1 enum Arc::X509Token::X509TokenType

X509TokeType is for distinguishing two types of operation. It is used as the parameter of constuctor.

5.193.3 Constructor & Destructor Documentation

5.193.3.1 Arc::X509Token::X509Token (SOAPEnvelope & *soap*, const std::string & *keyfile* = `""`)

Constructor.Parse X509 Token information from SOAP header. X509 Token related information is extracted from SOAP header and stored in class variables. And then it the X509Token object will be used for authentication if the tokentype is Signature; otherwise if the tokentype is Encryption, the encrypted soap body will be decrypted and replaced by decrypted message. keyfile is only needed when the X509Token is encryption token

5.193.3.2 Arc::X509Token::X509Token (SOAPEnvelope & *soap*, const std::string & *certfile*, const std::string & *keyfile*, X509TokenType *token_type* = `Signature`)

Constructor. Add X509 Token information into the SOAP header. Generated token contains elements X509 token and signature, and is meant to be used for authentication on the consuming side.

Parameters:

soap The SOAP message to which the X509 Token will be inserted

certfile The certificate file which will be used to encrypt the SOAP body (if parameter tokentype is Encryption), or be used as <wsse:BinarySecurityToken/> (if parameter tokentype is Signature).

keyfile The key file which will be used to create signature. Not needed when create encryption.

tokentype Token type: Signature or Encryption.

5.193.3.3 Arc::X509Token::~X509Token (void)

Deconstructor. Nothing to be done except finalizing the xmlsec library.

5.193.4 Member Function Documentation

5.193.4.1 bool Arc::X509Token::Authenticate (void)

Check signature by using the cert information in soap message. Only the signature itself is checked, and it is not guranteed that the certificate which is supposed to check the signature is trusted.

5.193.4.2 bool Arc::X509Token::Authenticate (const std::string & *cafile*, const std::string & *capath*)

Check signature by using the certificare information in X509Token which is parsed by the constructor, and the trusted certificates specified as one of the two parameters. Not only the signature (in the X509Token) itself is checked, but also the certificate which is supposed to check the signature needs to be trused (which means the certificate is issued by the ca certificate from CA file or CA directory). At least one the the two parameters should be set.

Parameters:

cafile The CA file

capath The CA directory

Returns:

true if authentication passes; otherwise false

5.193.4.3 Arc::X509Token::operator bool (void)

Returns true of constructor succeeded

The documentation for this class was generated from the following file:

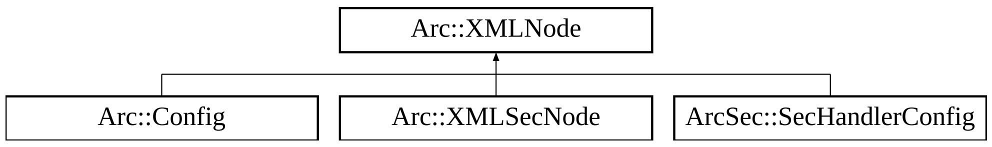
- X509Token.h

5.194 Arc::XMLNode Class Reference

Wrapper for LibXML library Tree interface.

`#include <XMLNode.h>`

Inheritance diagram for Arc::XMLNode::



Public Member Functions

- XMLNode (void)
- XMLNode (const XMLNode &node)
- XMLNode (const std::string &xml)
- XMLNode (const char ∗xml, int len=-1)
- XMLNode (long ptr_addr)
- XMLNode (const NS &ns, const char ∗name)
- ~XMLNode (void)
- void New (XMLNode &node) const
- void Exchange (XMLNode &node)
- void Move (XMLNode &node)
- void Swap (XMLNode &node)
- operator bool (void) const
- bool operator! (void) const
- bool operator== (const XMLNode &node)
- bool operator!= (const XMLNode &node)
- bool Same (const XMLNode &node)
- bool operator== (bool val)
- bool operator!= (bool val)
- bool operator== (const std::string &str)
- bool operator!= (const std::string &str)
- bool operator== (const char ∗str)
- bool operator!= (const char ∗str)
- XMLNode Child (int n=0)
- XMLNode operator[ ] (const char ∗name) const
- XMLNode operator[ ] (const std::string &name) const
- XMLNode operator[ ] (int n) const
- void operator++ (void)
- void operator– (void)
- int Size (void) const
- XMLNode Get (const std::string &name) const
- std::string Name (void) const
- std::string Prefix (void) const
- std::string FullName (void) const
- std::string Namespace (void) const

- void Name (const char ∗name)
- void Name (const std::string &name)
- void GetXML (std::string &out_xml_str, bool user_friendly=false) const
- void GetXML (std::string &out_xml_str, const std::string &encoding, bool user_friendly=false) const
- void GetDoc (std::string &out_xml_str, bool user_friendly=false) const
- operator std::string (void) const
- XMLNode & operator= (const char ∗content)
- XMLNode & operator= (const std::string &content)
- void Set (const std::string &content)
- XMLNode & operator= (const XMLNode &node)
- XMLNode Attribute (int n=0)
- XMLNode Attribute (const char ∗name)
- XMLNode Attribute (const std::string &name)
- XMLNode NewAttribute (const char ∗name)
- XMLNode NewAttribute (const std::string &name)
- int AttributesSize (void) const
- void Namespaces (const NS &namespaces, bool keep=false, int recursion=-1)
- NS Namespaces (void)
- std::string NamespacePrefix (const char ∗urn)
- XMLNode NewChild (const char ∗name, int n=-1, bool global_order=false)
- XMLNode NewChild (const std::string &name, int n=-1, bool global_order=false)
- XMLNode NewChild (const char ∗name, const NS &namespaces, int n=-1, bool global_order=false)
- XMLNode NewChild (const std::string &name, const NS &namespaces, int n=-1, bool global_-order=false)
- XMLNode NewChild (const XMLNode &node, int n=-1, bool global_order=false)
- void Replace (const XMLNode &node)
- void Destroy (void)
- XMLNodeList Path (const std::string &path)
- XMLNodeList XPathLookup (const std::string &xpathExpr, const NS &nsList)
- XMLNode GetRoot (void)
- XMLNode Parent (void)
- bool SaveToFile (const std::string &file_name) const
- bool SaveToStream (std::ostream &out) const
- bool ReadFromFile (const std::string &file_name)
- bool ReadFromStream (std::istream &in)
- bool Validate (const std::string &schema_file, std::string &err_msg)

Protected Member Functions

- XMLNode (xmlNodePtr node)

Protected Attributes

- bool is_owner_
- bool is_temporary_

Friends

- bool MatchXMLName (const XMLNode &node1, const XMLNode &node2)
- bool MatchXMLName (const XMLNode &node, const char ∗name)
- bool MatchXMLName (const XMLNode &node, const std::string &name)
- bool MatchXMLNamespace (const XMLNode &node1, const XMLNode &node2)
- bool MatchXMLNamespace (const XMLNode &node, const char ∗uri)
- bool MatchXMLNamespace (const XMLNode &node, const std::string &uri)

5.194.1 Detailed Description

Wrapper for LibXML library Tree interface.

This class wraps XML Node, Document and Property/Attribute structures. Each instance serves as pointer to actual LibXML element and provides convenient (for chosen purpose) methods for manipulating it. This class has no special ties to LibXML library and may be easily rewritten for any XML parser which provides interface similar to LibXML Tree. It implements only small subset of XML capabilities, which is probably enough for performing most of useful actions. This class also filters out (usually) useless textual nodes which are often used to make XML documents human-readable.

5.194.2 Constructor & Destructor Documentation

5.194.2.1 Arc::XMLNode::XMLNode (xmlNodePtr *node*) `[inline, protected]`

Private constructor for inherited classes Creates instance and links to existing LibXML structure. Acquired structure is not owned by class instance. If there is need to completely pass control of LibXML document to then instance's is_owner_ variable has to be set to true.

5.194.2.2 Arc::XMLNode::XMLNode (void) `[inline]`

Constructor of invalid node Created instance does not point to XML element. All methods are still allowed for such instance but produce no results.

5.194.2.3 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++.

5.194.2.4 Arc::XMLNode::XMLNode (const std::string & *xml*)

Creates XML document structure from textual representation of XML document. Created structure is pointed and owned by constructed instance

5.194.2.5 Arc::XMLNode::XMLNode (const char ∗ *xml*, int *len* = `-1`)

Same as previous

5.194.2.6 Arc::XMLNode::XMLNode (long *ptr_addr*)

Copy constructor. Used by language bindigs

5.194.2.7 Arc::XMLNode::XMLNode (const NS & *ns*, const char ∗ *name*)

Creates empty XML document structure with specified namespaces. Created XML contains only root element named 'name'. Created structure is pointed and owned by constructed instance

5.194.2.8 Arc::XMLNode::∼XMLNode (void)

Destructor Also destroys underlying XML document if owned by this instance

5.194.3 Member Function Documentation

5.194.3.1 XMLNode Arc::XMLNode::Attribute (const std::string & *name*) `[inline]`

Returns XMLNode instance representing first attribute of node with specified by name

5.194.3.2 XMLNode Arc::XMLNode::Attribute (const char ∗ *name*)

Returns XMLNode instance representing first attribute of node with specified by name

5.194.3.3 XMLNode Arc::XMLNode::Attribute (int *n* = `0`)

Returns list of all attributes of node.

Returns XMLNode instance reresenting n-th attribute of node.

5.194.3.4 int Arc::XMLNode::AttributesSize (void) const

Returns number of attributes of node

5.194.3.5 XMLNode Arc::XMLNode::Child (int *n* = `0`)

Returns XMLNode instance representing n-th child of XML element. If such does not exist invalid XMLN-ode instance is returned

5.194.3.6 void Arc::XMLNode::Destroy (void)

Destroys underlying XML element. XML element is unlinked from XML tree and destroyed. After this operation XMLNode instance becomes invalid

5.194.3.7 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 docummments. 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().

5.194.3.8 std::string Arc::XMLNode::FullName (void) const `[inline]`

Returns prefix:name of XML node

5.194.3.9 XMLNode Arc::XMLNode::Get (const std::string & *name*) const `[inline]`

Same as operator[ ]

5.194.3.10 void Arc::XMLNode::GetDoc (std::string & *out_xml_str*, bool *user_friendly* = `false`) const

Fills argument with whole XML document textual representation

5.194.3.11 XMLNode Arc::XMLNode::GetRoot (void)

Get the root node from any child node of the tree

5.194.3.12 void Arc::XMLNode::GetXML (std::string & *out_xml_str*, const std::string & *encoding*, bool *user_friendly* = `false`) const

Fills argument with this instance XML subtree textual representation if the XML subtree is corresponding to the encoding format specified in the argument, e.g. utf-8

5.194.3.13 void Arc::XMLNode::GetXML (std::string & *out_xml_str*, bool *user_friendly* = `false`) const

Fills argument with this instance XML subtree textual representation

5.194.3.14 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() and Destroy() because internally it is optimized not to copy data if not needed. The main purpose of this is to effectively extract part of XML document.

5.194.3.15 void Arc::XMLNode::Name (const std::string & *name*) `[inline]`

Assigns new name to XML node

5.194.3.16 void Arc::XMLNode::Name (const char ∗ *name*)

Assigns new name to XML node

5.194.3.17 std::string Arc::XMLNode::Name (void) const

Returns name of XML node

5.194.3.18 std::string Arc::XMLNode::Namespace (void) const

Returns namespace URI of XML node

5.194.3.19 std::string Arc::XMLNode::NamespacePrefix (const char ∗ *urn*)

Returns prefix of specified namespace. Empty string if no such namespace.

5.194.3.20 NS Arc::XMLNode::Namespaces (void)

Returns namespaces known at this node

5.194.3.21 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.

5.194.3.22 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.

5.194.3.23 XMLNode Arc::XMLNode::NewAttribute (const std::string & *name*) `[inline]`

Creates new attribute with specified name.

5.194.3.24 XMLNode Arc::XMLNode::NewAttribute (const char ∗ *name*)

Creates new attribute with specified name.

5.194.3.25 XMLNode Arc::XMLNode::NewChild (const XMLNode & *node*, int *n* = `-1`, bool *global_order* = `false`)

Link a copy of supplied XML node as child. Returns instance refering to new child. XML element is a copy of supplied one but not owned by returned instance

5.194.3.26 XMLNode Arc::XMLNode::NewChild (const std::string & *name*, const NS & *namespaces*, int *n* = `-1`, bool *global_order* = `false`) `[inline]`

Same as NewChild(const char∗,const NS&,int,bool)

5.194.3.27 XMLNode Arc::XMLNode::NewChild (const char ∗ *name*, const NS & *namespaces*, int *n* = `-1`, bool *global_order* = `false`)

Creates new child XML element at specified position with specified name and namespaces. For more information look at NewChild(const char∗,int,bool)

5.194.3.28 XMLNode Arc::XMLNode::NewChild (const std::string & *name*, int *n* = `-1`, bool *global_order* = `false`) `[inline]`

Same as NewChild(const char∗,int,bool)

5.194.3.29 XMLNode Arc::XMLNode::NewChild (const char ∗ *name*, int *n* = `-1`, bool *global_order* = `false`)

Creates new child XML element at specified position with specified name. Default is to put it at end of list. If global order is true position applies to whole set of children, otherwise only to children of same name. Returns created node.

5.194.3.30 Arc::XMLNode::operator bool (void) const `[inline]`

Returns true if instance points to XML element - valid instance

5.194.3.31 Arc::XMLNode::operator std::string (void) const

Returns textual content of node excluding content of children nodes

5.194.3.32 bool Arc::XMLNode::operator! (void) const `[inline]`

Returns true if instance does not point to XML element - invalid instance

5.194.3.33 bool Arc::XMLNode::operator!= (const char ∗ *str*) `[inline]`

This operator is needed to avoid ambiguity

5.194.3.34 bool Arc::XMLNode::operator!= (const std::string & *str*) `[inline]`

This operator is needed to avoid ambiguity

5.194.3.35 bool Arc::XMLNode::operator!= (bool *val*) `[inline]`

This operator is needed to avoid ambiguity

5.194.3.36 bool Arc::XMLNode::operator!= (const XMLNode & *node*) [inline]

Returns false if 'node' represents same XML element

5.194.3.37 void Arc::XMLNode::operator++ (void)

Convenience operator to switch to next element of same name. If there is no such node this object becomes invalid.

5.194.3.38 void Arc::XMLNode::operator– (void)

Convenience operator to switch to previous element of same name. If there is no such node this object becomes invalid.

5.194.3.39 XMLNode& Arc::XMLNode::operator= (const XMLNode & *node*)

Make instance refer to another XML node. Ownership is not inherited. Due to nature of XMLNode there should be no const here, but that does not fit into C++.

5.194.3.40 XMLNode& Arc::XMLNode::operator= (const std::string & *content*) [inline]

Sets textual content of node. All existing children nodes are discarded.

5.194.3.41 XMLNode& Arc::XMLNode::operator= (const char ∗ *content*)

Sets textual content of node. All existing children nodes are discarded.

5.194.3.42 bool Arc::XMLNode::operator== (const char ∗ *str*) [inline]

This operator is needed to avoid ambiguity

5.194.3.43 bool Arc::XMLNode::operator== (const std::string & *str*) [inline]

This operator is needed to avoid ambiguity

5.194.3.44 bool Arc::XMLNode::operator== (bool *val*) [inline]

This operator is needed to avoid ambiguity

5.194.3.45 bool Arc::XMLNode::operator== (const XMLNode & *node*) [inline]

Returns true if 'node' represents same XML element

5.194.3.46]

XMLNode Arc::XMLNode::operator[ ] (int *n*) const

Returns XMLNode instance representing n-th node in sequence of siblings of same name. It's main purpose is to be used to retrieve element in array of children of same name like node["name"][5]. This method should not be marked const because obtaining unrestricted XMLNode of child element allows modification of underlying XML tree. But in order to keep const in other places non-const-handling is passed to programmer. Otherwise C++ compiler goes nuts.

5.194.3.47]

XMLNode Arc::XMLNode::operator[ ] (const std::string & *name*) const `[inline]`

Similar to previous method

5.194.3.48]

XMLNode Arc::XMLNode::operator[ ] (const char ∗ *name*) const

Returns XMLNode instance representing first child element with specified name. Name may be "namespace_prefix:name", "namespace_uri:name" or simply "name". In last case namespace is ignored. If such node does not exist invalid XMLNode instance is returned. This method should not be marked const because obtaining unrestricted XMLNode of child element allows modification of underlying XML tree. But in order to keep const in other places non-const-handling is passed to programmer. Otherwise C++ compiler goes nuts.

5.194.3.49 XMLNode Arc::XMLNode::Parent (void)

Get the parent node from any child node of the tree

5.194.3.50 XMLNodeList Arc::XMLNode::Path (const std::string & *path*)

Collects nodes corresponding to specified path. This is a convenience function to cover common use of XPath but without performance hit. Path is made of node_name[/node_name[...]] and is relative to current node. node_names are treated in same way as in operator[ ]. Returns all nodes which are represented by path.

5.194.3.51 std::string Arc::XMLNode::Prefix (void) const

Returns namespace prefix of XML node

5.194.3.52 bool Arc::XMLNode::ReadFromFile (const std::string & *file_name*)

Read XML document from file and associate it with this node

5.194.3.53 bool Arc::XMLNode::ReadFromStream (std::istream & *in*)

Read XML document from stream and associate it with this node

5.194.3.54 void Arc::XMLNode::Replace (const XMLNode & *node*)

Makes a copy of supplied XML node and makes this instance refere to it

5.194.3.55 bool Arc::XMLNode::Same (const XMLNode & *node*) `[inline]`

Returns true if 'node' represents same XML element - for bindings

5.194.3.56 bool Arc::XMLNode::SaveToFile (const std::string & *file_name*) const

Save string representation of node to file

5.194.3.57 bool Arc::XMLNode::SaveToStream (std::ostream & *out*) const

Save string representation of node to stream

5.194.3.58 void Arc::XMLNode::Set (const std::string & *content*) `[inline]`

Same as operator=. Used for bindings.

5.194.3.59 int Arc::XMLNode::Size (void) const

Returns number of children nodes

5.194.3.60 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 tmp=∗this; ∗this=node; node=tmp; But in case of either this or node owning XML document ownership is swapped too. And this is a main purpose of Swap() method.

5.194.3.61 bool Arc::XMLNode::Validate (const std::string & *schema_file*, std::string & *err_msg*)

XML schema validation against the schema file defined as argument

5.194.3.62 XMLNodeList Arc::XMLNode::XPathLookup (const std::string & *xpathExpr*, const NS & *nsList*)

Uses xPath to look up the whole xml structure, Returns a list of XMLNode points. The xpathExpr should be like "//xx:child1/" which indicates the namespace and node that you would like to find; The nsList is the namespace the result should belong to (e.g. xx="uri:test"). Query is run on whole XML document but only the elements belonging to this XML subtree are returned.

5.194.4 Friends And Related Function Documentation

5.194.4.1 bool MatchXMLName (const XMLNode & *node*, const std::string & *name*) `[friend]`

Returns true if 'name' matches name of 'node'. If name contains prefix it's checked too

5.194.4.2 bool MatchXMLName (const XMLNode & *node*, const char ∗ *name*) [friend]

Returns true if 'name' matches name of 'node'. If name contains prefix it's checked too

5.194.4.3 bool MatchXMLName (const XMLNode & *node1*, const XMLNode & *node2*) [friend]

Returns true if underlying XML elements have same names

5.194.4.4 bool MatchXMLNamespace (const XMLNode & *node*, const std::string & *uri*) [friend]

Returns true if 'namespace' matches 'node's namespace.

5.194.4.5 bool MatchXMLNamespace (const XMLNode & *node*, const char ∗ *uri*) [friend]

Returns true if 'namespace' matches 'node's namespace.

5.194.4.6 bool MatchXMLNamespace (const XMLNode & *node1*, const XMLNode & *node2*) [friend]

Returns true if underlying XML elements belong to same namespaces

5.194.5 Field Documentation

5.194.5.1 bool Arc::XMLNode::is_owner_ [protected]

If true node is owned by this instance - hence released in destructor. Normally that may be true only for top level node of XML document.

5.194.5.2 bool Arc::XMLNode::is_temporary_ [protected]

This variable is for future

The documentation for this class was generated from the following file:

- XMLNode.h

5.195 Arc::XMLNodeContainer Class Reference

```
#include <XMLNode.h>
```

Public Member Functions

- XMLNodeContainer (void)
- XMLNodeContainer (const XMLNodeContainer &)
- XMLNodeContainer & operator= (const XMLNodeContainer &)
- void Add (const XMLNode &)
- void Add (const std::list< XMLNode > &)
- void AddNew (const XMLNode &)
- void AddNew (const std::list< XMLNode > &)
- int Size (void) const
- XMLNode operator[ ] (int)
- std::list< XMLNode > Nodes (void)

5.195.1 Detailed Description

Container for multiple XMLNode elements

5.195.2 Constructor & Destructor Documentation

5.195.2.1 Arc::XMLNodeContainer::XMLNodeContainer (void)

Default constructor

5.195.2.2 Arc::XMLNodeContainer::XMLNodeContainer (const XMLNodeContainer &)

Copy constructor. Add nodes from argument. Nodes owning XML document are copied using AddNew(). Not owning nodes are linked using Add() method.

5.195.3 Member Function Documentation

5.195.3.1 void Arc::XMLNodeContainer::Add (const std::list< XMLNode > &)

Link multiple XML subtrees to container.

5.195.3.2 void Arc::XMLNodeContainer::Add (const XMLNode &)

Link XML subtree refered by node to container. XML tree must be available as long as this object is used.

5.195.3.3 void Arc::XMLNodeContainer::AddNew (const std::list< XMLNode > &)

Copy multiple XML subtrees to container.

5.195.3.4 void Arc::XMLNodeContainer::AddNew (const XMLNode &)

Copy XML subtree referenced by node to container. After this operation container refers to independent XML document. This document is deleted when container is destroyed.

5.195.3.5 std::list<XMLNode> Arc::XMLNodeContainer::Nodes (void)

Returns all stored nodes.

5.195.3.6 XMLNodeContainer& Arc::XMLNodeContainer::operator= (const XMLNodeContainer &)

Same as copy constructor with current nodes being deleted first.

5.195.3.7]

XMLNode Arc::XMLNodeContainer::operator[ ] (int)

Returns n-th node in a store.

5.195.3.8 int Arc::XMLNodeContainer::Size (void) const

Return number of refered/stored nodes.

The documentation for this class was generated from the following file:

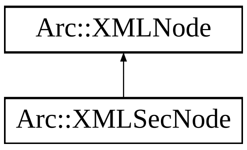
- XMLNode.h

5.196 Arc::XMLSecNode Class Reference

Extends XMLNode class to support XML security operation.

`#include <XMLSecNode.h>`

Inheritance diagram for Arc::XMLSecNode::



Public Member Functions

- XMLSecNode (XMLNode &node)
- void AddSignatureTemplate (const std::string &id_name, const SignatureMethod sign_method, const std::string &incl_namespaces="")
- bool SignNode (const std::string &privkey_file, const std::string &cert_file)
- bool VerifyNode (const std::string &id_name, const std::string &ca_file, const std::string &ca_path, bool verify_trusted=true)
- bool EncryptNode (const std::string &cert_file, const SymEncryptionType encrpt_type)
- bool DecryptNode (const std::string &privkey_file, XMLNode &decrypted_node)

5.196.1 Detailed Description

Extends XMLNode class to support XML security operation.

All XMLNode methods are exposed by inheriting from XMLNode. XMLSecNode itself does not own node, instead it uses the node from the base class XMLNode.

5.196.2 Constructor & Destructor Documentation

5.196.2.1 Arc::XMLSecNode::XMLSecNode (XMLNode & *node*)

Create a object based on an XMLNode instance.

5.196.3 Member Function Documentation

5.196.3.1 void Arc::XMLSecNode::AddSignatureTemplate (const std::string & *id_name*, const SignatureMethod *sign_method*, const std::string & *incl_namespaces* = "")

Add the signature template for later signing.

Parameters:

id_name The identifier name under this node which will be used for the <Signature> to refer to.

sign_method The sign method for signing. Two options now, RSA_SHA1, DSA_SHA1

5.196.3.2 bool Arc::XMLSecNode::DecryptNode (const std::string & *privkey_file*, XMLNode & *decrypted_node*)

Decrypt the <xenc:EncryptedData/> under this node, the decrypted node will be output in the second argument of DecryptNode method. And the <xenc:EncryptedData/> under this node will be removed after decryption.

Parameters:

privkey_file The private key file, which is used for decrypting

decrypted_node Output the decrypted node

5.196.3.3 bool Arc::XMLSecNode::EncryptNode (const std::string & *cert_file*, const SymEncryptionType *encrpt_type*)

Encrypt this node, after encryption, this node will be replaced by the encrypted node

Parameters:

cert_file The certificate file, the public key parsed from this certificate is used to encrypted the symmetric key, and then the symmetric key is used to encrypted the node

encrpt_type The encryption type when encrypting the node, four option in SymEncryptionType

verify_trusted Verify trusted certificates or not. If set to false, then only the signature will be checked (by using the public key from KeyInfo).

5.196.3.4 bool Arc::XMLSecNode::SignNode (const std::string & *privkey_file*, const std::string & *cert_file*)

Sign this node (identified by id_name).

Parameters:

privkey_file The private key file. The private key is used for signing

cert_file The certificate file. The certificate is used as the <KeyInfo> part of the <Signature>; <KeyInfo> will be used for the other end to verify this <Signature>

incl_namespaces InclusiveNamespaces for Tranform in Signature

5.196.3.5 bool Arc::XMLSecNode::VerifyNode (const std::string & *id_name*, const std::string & *ca_file*, const std::string & *ca_path*, bool *verify_trusted* = `true`)

Verify the signature under this node

Parameters:

id_name The id of this node, which is used for identifying the node

ca_file The CA file which used as trused certificate when verify the certificate in the <KeyInfo> part of <Signature>

ca_path The CA directory; either ca_file or ca_path should be set.

The documentation for this class was generated from the following file:

- XMLSecNode.h

Index