KnowARC Reference Manual

Generated by Doxygen 1.4.7

Fri Dec 14 11:20:07 2007

Contents

Chapter 1

KnowARC Namespace Index

1.1 KnowARC Namespace List

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

Chapter 2

KnowARC Hierarchical Index

2.1 KnowARC Class Hierarchy

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

Chapter 3

KnowARC Class Index

3.1 KnowARC Class List

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

Chapter 4

KnowARC Namespace Documentation

4.1 Arc Namespace Reference

Classes

- class Config

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

- class RegularExpression

  A regular expression class.

- class **Base64**
- class Counter

  A class defining a common interface for counters.

- class CounterTicket

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

- class ExpirationReminder

  A class intended for internal use within counters.

- class IntraProcessCounter

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

- class **Period**
- class Time

  A class for storing and manipulating times.

- class LogMessage

  A class for log messages.

- class LogDestination

  A base class for log destinations.

- class LogStream

A class for logging to ostreams.

- class Logger

  A logger class.

- class Run
- class SimpleCondition

  Simple triggered condition.

- class URL

  Class to hold general URL's.

- class URLLocation

  Class to hold a resolved URL location.

- class **User**
- class XMLNode

  Wrapper for LibXML library Tree interface.

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

  Defines interface for variuos checksum manipulations.

- class CRC32Sum

  Implementation of CRC32 checksum.

- class MD5Sum

  Implementation of MD5 checksum.

- class CheckSumAny

  Wraper for CheckSum class.

- class DataBufferPar

  Represents set of buffers.

- class DataCache
- class DataCallback
- class DataHandle

  This class is a wrapper around the DataPoint class.

- class DataMover
- class DataPoint

  This class is an abstraction of URL.

- class DataPointDirect

  This is kind of generalized file handle.

- class DataPointIndex

  Complements DataPoint with attributes common for meta-URLs.

- class DataSpeed

  Keeps track of average and instantaneous transfer speed.

- class **DMC**
- class FileInfo

  FileInfo stores information about files (metadata).

- class **URLMap**
- class DelegationConsumer
- class DelegationProvider
- class DelegationConsumerSOAP
- class DelegationProviderSOAP
- class DelegationContainerSOAP
- class **InfoCache**
- class 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 **ClassLoader**
- class DMCFactory
- class **LoadableClass**
- class Loader

  Creator of Message Component Chains (MCC).

- class ChainContext

  Interface to chain specific functionality.

- struct loader_descriptor

  Identifier of plugin.

- class LoaderFactory

  Plugin handler.

- class MCCFactory

  MCC Plugins handler.

- class ModuleManager

  Manager of shared libraries.

- class PDPFactory

  PDP Plugins handler.

- class PlexerEntry

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

- class Plexer

  The Plexer class, used for routing messages to services.

- class SecHandlerFactory

  SecHandler Plugins handler.

- class ServiceFactory

  Service Plugins handler.

- class MCCInterface

  Interface for communication between MCC, Service and Plexer objects.

- class MCC

  Message Chain Component - base class for every MCC plugin.

- class MCC_Status

  A class for communication of MCC processing results.

- class MessagePayload

  Base class for content of message passed through chain.

- class MessageContextElement

  Top class for elements contained in message context.

- class MessageContext

  Handler for context of message context.

- class Message

  Object being passed through chain of MCCs.

- class AttributeIterator

  An iterator class for accessing multiple values of an attribute.

- class MessageAttributes

  A class for storage of attribute values.

- class MessageAuth

  Contains authencity information, authorization tokens and decisions.

- class PayloadRawInterface

  Random Access Payload for Message objects.

- struct **PayloadRawBuf**
- class PayloadRaw

  Raw byte multi-buffer.

- class PayloadSOAP

  Payload of Message with SOAP content.

- class PayloadStreamInterface

  Stream-like Payload for Message object.

- class PayloadStream

  POSIX handle as Payload.

- class Service

  Service - last component in a Message Chain.

- class SOAPFault

  Interface to SOAP Fault message.

- class SOAPEnvelope

  Extends XMLNode class to support structures of SOAP message.

- class SOAPMessage

  Message restricted to SOAP payload.

- class BaseConfig
- class **ClientInterface**
- class **ClientTCP**
- class **ClientHTTP**
- class ClientSOAP
- class **MCCConfig**
- class **DMCConfig**
- class WSAEndpointReference

  Interface for manipulation of WS-Adressing Endpoint Reference.

- class WSAHeader

  Interface for manipulation WS-Addressing information in SOAP header.

- class UsernameToken

  Interface for manipulation of WS-Security Username Token Profile.

- class PayloadWSRF

  This class combines MessagePayload with WSRF.

- class WSRP

  Base class for WS-ResourceProperties structures.

- class WSRPFault

  Base class for WS-ResourceProperties faults.

- class **WSRPInvalidResourcePropertyQNameFault**
- class WSRPResourcePropertyChangeFailure
- class **WSRPUnableToPutResourcePropertyDocumentFault**
- class **WSRPInvalidModificationFault**

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

  Base class for every WSRF message.

- class WSRFBaseFault

  Base class for WSRF fault messages.

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

Typedefs

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

Enumerations

- enum TimeFormat {

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

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

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

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

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

  ERROR = 16, **FATAL** = 32 }
- enum StatusKind {

  STATUS_UNDEFINED = 0, STATUS_OK = 1, GENERIC_ERROR = 2, PARSING_ERROR = 4,

  PROTOCOL_RECOGNIZED_ERROR = 8, UNKNOWN_SERVICE_ERROR = 16, BUSY_-ERROR = 32, SESSION_CLOSE = 64 }
- enum WSAFault {

  WSAFaultNone, WSAFaultUnknown, WSAFaultInvalidAddressingHeader, **WSAFaultInvalid-Address**,

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

  WSAFaultActionMismatch, **WSAFaultOnlyAnonymousAddressSupported**, **WSAFaultOnly-NonAnonymousAddressSupported**, **WSAFaultMessageAddressingHeaderRequired**,

  WSAFaultDestinationUnreachable, **WSAFaultActionNotSupported**, **WSAFaultEndpoint-Unavailable** }

Functions

- std::ostream & operator<< (std::ostream &, const Period &)
- std::ostream & operator<< (std::ostream &, const Time &)
- std::string TimeStamp (const TimeFormat &=Time::GetFormat())
- std::string TimeStamp (Time, const TimeFormat &=Time::GetFormat())
- void GUID (std::string &guid)
- std::ostream & operator<< (std::ostream &os, LogLevel level)
- LogLevel **string_to_level** (const std::string &str)
- template<typename T> T stringto (const std::string &s)
- template<typename T> bool stringto (const std::string &s, T &t)
- template<typename T> std::string tostring (T t, const int width=0, const int precision=0)
- bool CreateThreadFunction (void(*func)(void *), void *arg)
- std::list< URL > ReadURLList (const URL &urllist)
- bool MatchXMLName (const XMLNode &node1, const XMLNode &node2)
- bool MatchXMLName (const XMLNode &node, const char *name)
- bool MatchXMLName (const XMLNode &node, const std::string &name)
- bool MatchXMLNamespace (const XMLNode &node1, const XMLNode &node2)
- bool MatchXMLNamespace (const XMLNode &node, const char *uri)
- bool MatchXMLNamespace (const XMLNode &node, const std::string &uri)
- int **cache_download_url_start** (const std::string &cache_path, const std::string &cache_data_path, uid_t cache_uid, gid_t cache_gid, const std::string &url, const std::string &id, cache_download_-handler &handler)

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

Variables

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

4.1.1 Detailed Description

Platform independent represnetation of system user

4.1.2 Typedef Documentation

4.1.2.1 typedef loader_descriptor Arc::loader_descriptors[ ]

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

4.1.2.2 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.

4.1.3.3 enum Arc::StatusKind

Status kinds (types).

This enum defines a set of possible status kinds.

Enumerator:

STATUS_OK Default status - undefined error.

GENERIC_ERROR No error.

PARSING_ERROR Error does not fit any class.

PROTOCOL_RECOGNIZED_ERROR Error detected while parsing request/response.

UNKNOWN_SERVICE_ERROR Message does not fit into expected protocol.

BUSY_ERROR There is no destination configured for this message.

SESSION_CLOSE Message can't be processed now.

4.1.3.4 enum Arc::WSAFault

WS-Addressing possible faults.

Enumerator:

WSAFaultUnknown This is not a fault

WSAFaultInvalidAddressingHeader This is not a WS-Addressing fault

4.1.4 Function Documentation

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

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

4.1.4.6 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.

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

This method converts a string to any type.

4.1.4.8 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.

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

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

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

Helper function to create simple thread.

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

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

Reads a list of URLs from a file.

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

Returns true if underlying XML elements have same names

4.1.4.13 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

4.1.4.14 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

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

Returns true if underlying XML elements belong to same namespaces

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

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

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

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

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

Conversion to string.

Conversion from StatusKind to string.

Parameters:

kind The StatusKind to convert.

4.1.4.19 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.

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

Makes WS-Addressing fault.

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

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

Gets WS-addressing fault.

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

4.1.5 Variable Documentation

4.1.5.1 const Glib::TimeVal Arc::ETERNAL

A time very far in the future.

4.1.5.2 const Glib::TimeVal Arc::HISTORIC

A time very far in the past.

Chapter 5

KnowARC Class Documentation

5.1 Arc::AttributeIterator Class Reference

An iterator class for accessing multiple values of an attribute.

`#include <MessageAttributes.h>`

Public Member Functions

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

Protected Member Functions

- AttributeIterator (AttrConstIter begin, AttrConstIter end)

Protected Attributes

- AttrConstIter current_
- AttrConstIter end_

Friends

- class MessageAttributes

5.1.1 Detailed Description

An iterator class for accessing multiple values of an attribute.

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

Typical usage is:

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

5.1.2 Constructor & Destructor Documentation

5.1.2.1 Arc::AttributeIterator::AttributeIterator ()

Default constructor.

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

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

Protected constructor used by the MessageAttributes class.

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

Parameters:

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

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

5.1.3 Member Function Documentation

5.1.3.1 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.

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

The key of attribute.

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

5.1.3.3 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.

5.1.3.4 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.

5.1.3.5 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.

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

The arrow operator.

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

5.1.4 Friends And Related Function Documentation

5.1.4.1 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.

5.1.5 Member Data Documentation

5.1.5.1 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.

5.1.5.2 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

5.2 Arc::BaseConfig Class Reference

`#include <ClientInterface.h>`

Public Member Functions

- void AddPluginsPath (const std::string &path)
- void AddPrivateKey (const std::string &path)
- void AddCertificate (const std::string &path)
- void AddCAFile (const std::string &path)
- virtual XMLNode MakeConfig (XMLNode cfg) const

Public Attributes

- std::string **key**
- std::string **cert**
- std::string **proxy**
- std::string **cafile**

Protected Attributes

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

5.2.1 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.

5.2.2 Member Function Documentation

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

Add CA file

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

Add certificate

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

Adds non-standard location of plugins

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

Add private key

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

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

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

- ClientInterface.h

5.3 Arc::ChainContext Class Reference

Interface to chain specific functionality.

`#include <Loader.h>`

Public Member Functions

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

Friends

- class **Loader**

5.3.1 Detailed Description

Interface to chain specific functionality.

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

5.3.2 Member Function Documentation

5.3.2.1 Arc::ChainContext::operator MCCFactory * () [inline]

Returns associated MCCFactory object

5.3.2.2 Arc::ChainContext::operator PDPFactory * () [inline]

Returns associated PDPFactory object

5.3.2.3 Arc::ChainContext::operator SecHandlerFactory * () [inline]

Returns associated SecHandlerFactory object

5.3.2.4 Arc::ChainContext::operator ServiceFactory * () [inline]

Returns associated ServiceFactory object

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

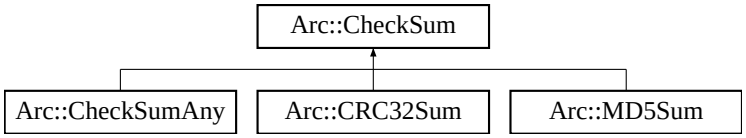
- Loader.h

5.4 Arc::CheckSum Class Reference

Defines interface for variuos checksum manipulations.

`#include <CheckSum.h>`

Inheritance diagram for Arc::CheckSum::



Public Member Functions

- virtual void **start** (void)=0
- virtual void **add** (void *buf, unsigned long long int len)=0
- virtual void **end** (void)=0
- virtual void **result** (unsigned char *&res, unsigned int &len) const =0
- virtual int **print** (char *buf, int len) const
- virtual void **scan** (const char *buf)=0
- virtual **operator bool** (void) const
- virtual bool **operator!** (void) const

5.4.1 Detailed Description

Defines interface for variuos checksum manipulations.

This class is used during data transfers through DataBufferPar class

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

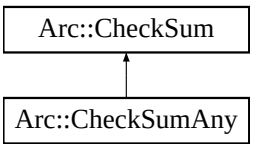
- CheckSum.h

5.5 Arc::CheckSumAny Class Reference

Wraper for CheckSum class.

`#include <CheckSum.h>`

Inheritance diagram for Arc::CheckSumAny::



Public Types

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

  none, **unknown**, **undefined**, **cksum**,

  md5 }

Public Member Functions

- **CheckSumAny** (CheckSum *c=NULL)
- **CheckSumAny** (type type)
- **CheckSumAny** (const char *type)
- virtual void **start** (void)
- virtual void **add** (void *buf, unsigned long long int len)
- virtual void **end** (void)
- virtual void **result** (unsigned char *&res, unsigned int &len) const
- virtual int **print** (char *buf, int len) const
- virtual void **scan** (const char *buf)
- virtual **operator bool** (void) const
- virtual bool **operator!** (void) const
- bool **active** (void)
- type **Type** (void)
- void **operator=** (const char *type)
- bool **operator==** (const char *s)
- bool **operator==** (const CheckSumAny &ck)

Static Public Member Functions

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

5.5.1 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

5.6 Arc::ClientSOAP Class Reference

`#include <ClientInterface.h>`

Public Member Functions

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

Protected Attributes

- MCC * **soap_entry**

5.6.1 Detailed Description

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

5.6.2 Constructor & Destructor Documentation

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

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

5.6.3 Member Function Documentation

5.6.3.1 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:

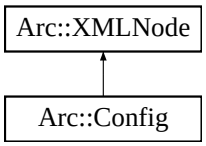
- ClientInterface.h

5.7 Arc::Config Class Reference

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

`#include <ArcConfig.h>`

Inheritance diagram for Arc::Config::



Public Member Functions

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

5.7.1 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.

5.7.2 Constructor & Destructor Documentation

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

Dummy constructor - produces invalid structure

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

Creates empty XML tree

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

Loads configuration document from file 'filename'

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

Parse configuration document from memory

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

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

5.7.3 Member Function Documentation

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

Parse configuration document from file 'filename'

5.7.3.2 void Arc::Config::print (void)

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

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

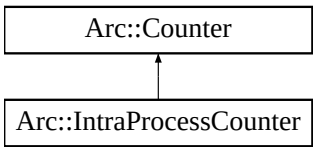
- ArcConfig.h

5.8 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

5.8.1 Detailed Description

A class defining a common interface for counters.

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

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

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

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

Typical usage is:

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

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

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

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

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

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

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

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

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

5.8.2 Member Typedef Documentation

5.8.2.1 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.

5.8.3 Constructor & Destructor Documentation

5.8.3.1 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.

5.8.3.2 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.

5.8.4 Member Function Documentation

5.8.4.1 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.

5.8.4.2 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.

5.8.4.3 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.

5.8.4.4 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.

5.8.4.5 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.

5.8.4.6 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.

5.8.4.7 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.

5.8.4.8 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.

5.8.4.9 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.

5.8.4.10 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.

5.8.4.11 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.

5.8.4.12 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.

5.8.4.13 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.

5.8.4.14 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.

5.8.5 Friends And Related Function Documentation

5.8.5.1 friend class CounterTicket [friend]

The CounterTicket class needs to be a friend.

5.8.5.2 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

5.9 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

5.9.1 Detailed Description

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

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

Typical usage is:

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

5.9.2 Constructor & Destructor Documentation

5.9.2.1 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.

5.9.3 Member Function Documentation

5.9.3.1 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.

5.9.3.2 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.

5.9.3.3 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.

5.9.4 Friends And Related Function Documentation

5.9.4.1 friend class Counter [friend]

The Counter class needs to be a friend.

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

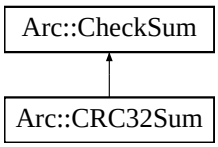
- Counter.h

5.10 Arc::CRC32Sum Class Reference

Implementation of CRC32 checksum.

`#include <CheckSum.h>`

Inheritance diagram for Arc::CRC32Sum::



Public Member Functions

- virtual void **start** (void)
- virtual void **add** (void *buf, unsigned long long int len)
- virtual void **end** (void)
- virtual void **result** (unsigned char *&res, unsigned int &len) const
- virtual int **print** (char *buf, int len) const
- virtual void **scan** (const char *buf)
- virtual **operator bool** (void) const
- virtual bool **operator!** (void) const
- uint32_t **crc** (void) const

5.10.1 Detailed Description

Implementation of CRC32 checksum.

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

- CheckSum.h

5.11 Arc::DataBufferPar Class Reference

Represents set of buffers.

`#include <DataBufferPar.h>`

Public Member Functions

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

Public Attributes

- DataSpeed speed

Classes

- struct **buf_desc**

5.11.1 Detailed Description

Represents set of buffers.

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

5.11.2 Constructor & Destructor Documentation

5.11.2.1 Arc::DataBufferPar::DataBufferPar (unsigned int *size* = 65536, int *blocks* = 3)

Contructor

Parameters:

size size of every buffer in bytes.

blocks number of buffers.

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

Contructor

Parameters:

size size of every buffer in bytes.

blocks number of buffers.

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

5.11.2.3 Arc::DataBufferPar::~DataBufferPar ()

Destructor.

5.11.3 Member Function Documentation

5.11.3.1 unsigned int Arc::DataBufferPar::buffer_size ()

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

5.11.3.2 const CheckSum* Arc::DataBufferPar::checksum_object ()

Returns CheckSum object specified in constructor.

5.11.3.3 bool Arc::DataBufferPar::checksum_valid ()

Returns true if checksum was successfully computed.

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

Returns offset following last piece of data transfered.

5.11.3.5 bool Arc::DataBufferPar::eof_read ()

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

5.11.3.6 void Arc::DataBufferPar::eof_read (bool *v*)

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

5.11.3.7 bool Arc::DataBufferPar::eof_write ()

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

5.11.3.8 void Arc::DataBufferPar::eof_write (bool *v*)

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

5.11.3.9 bool Arc::DataBufferPar::error ()

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

5.11.3.10 bool Arc::DataBufferPar::error_read ()

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

5.11.3.11 void Arc::DataBufferPar::error_read (bool *v*)

Informs object if error accured on 'read' side.

Parameters:

v true if error.

5.11.3.12 bool Arc::DataBufferPar::error_transfer ()

Returns true if eror occured inside object.

5.11.3.13 bool Arc::DataBufferPar::error_write ()

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

5.11.3.14 void Arc::DataBufferPar::error_write (bool *v*)

Informs object if error accured on 'write' side.

Parameters:

v true if error.

5.11.3.15 bool Arc::DataBufferPar::for_read ()

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

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

Request buffer for READING INTO it.

Parameters:

handle returns buffer's number.

length returns size of buffer

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

Returns:

true on success

5.11.3.17 bool Arc::DataBufferPar::for_write ()

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

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

Request buffer for WRITING FROM it.

Parameters:

handle returns buffer's number.

length returns size of buffer

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

5.11.3.19 bool Arc::DataBufferPar::is_notwritten (char * *buf*)

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

Parameters:

buf - address of buffer

5.11.3.20 bool Arc::DataBufferPar::is_notwritten (int *handle*)

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

Parameters:

handle buffer's number.

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

Informs object that data was read into buffer.

Parameters:

buf - address of buffer

length amount of data.

offset offset in stream, file, etc.

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

Informs object that data was read into buffer.

Parameters:

handle buffer's number.

length amount of data.

offset offset in stream, file, etc.

5.11.3.23 bool Arc::DataBufferPar::is_written (char * *buf*)

Informs object that data was written from buffer.

Parameters:

buf - address of buffer

5.11.3.24 bool Arc::DataBufferPar::is_written (int *handle*)

Informs object that data was written from buffer.

Parameters:

handle buffer's number.

5.11.3.25 Arc::DataBufferPar::operator bool (void) [inline]

Check if DataBufferPar object is initialized.

5.11.3.26]

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

Direct access to buffer by number.

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

Reinitialize buffers with different parameters.

Parameters:

size size of every buffer in bytes.

blocks number of buffers.

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

5.11.3.28 bool Arc::DataBufferPar::wait ()

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

5.11.3.29 bool Arc::DataBufferPar::wait_eof ()

Wait till end of transfer happens on any side.

5.11.3.30 bool Arc::DataBufferPar::wait_eof_read ()

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

5.11.3.31 bool Arc::DataBufferPar::wait_eof_write ()

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

5.11.3.32 bool Arc::DataBufferPar::wait_read ()

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

5.11.3.33 bool Arc::DataBufferPar::wait_used ()

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

5.11.3.34 bool Arc::DataBufferPar::wait_write ()

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

5.11.4 Member Data Documentation

5.11.4.1 DataSpeed Arc::DataBufferPar::speed

This object controls transfer speed.

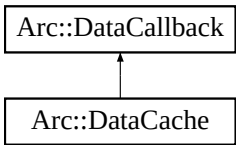
The documentation for this class was generated from the following file:

- DataBufferPar.h

5.12 Arc::DataCache Class Reference

`#include <DataCache.h>`

Inheritance diagram for Arc::DataCache::



Public Types

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

Public Member Functions

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

5.12.1 Detailed Description

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

5.12.2 Constructor & Destructor Documentation

5.12.2.1 Arc::DataCache::DataCache ()

Default constructor (non-functional cache).

5.12.2.2 Arc::DataCache::DataCache (const std::string & *cache_path*, const std::string & *cache_data_path*, const std::string & *cache_link_path*, const std::string & *id*, uid_t *cache_uid*, gid_t *cache_gid*)

Constructor

Parameters:

cache_path path to directory with cache info files

cache_data_path path to directory with cache data files

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

id identifier used to claim files in cache

cache_uid owner of cahce (0 for public cache)

cache_gid owner group of cache (0 for public cache)

5.12.2.3 Arc::DataCache::DataCache (const DataCache & *cache*)

Copy constructor.

5.12.2.4 virtual Arc::DataCache::~DataCache () [virtual]

and destructor

5.12.3 Member Function Documentation

5.12.3.1 virtual bool Arc::DataCache::cb (unsigned long long int *size*) [virtual]

Callback implementation to clean at least 1 byte.

Reimplemented from Arc::DataCallback.

5.12.3.2 bool Arc::DataCache::CheckCreated () [inline]

Check if there is an information about creation time.

5.12.3.3 bool Arc::DataCache::CheckValid () [inline]

Check if there is an information about invalidation time.

5.12.3.4 bool Arc::DataCache::clean (unsigned long long int *size* = 1)

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

Parameters:

size amount to be removed (bytes)

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

Do same as link() but always create copy.

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

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

5.12.3.7 Time Arc::DataCache::GetCreated () [inline]

Get creation time.

5.12.3.8 Time Arc::DataCache::GetValid () [inline]

Get invalidation time.

5.12.3.9 bool Arc::DataCache::link (const std::string & *link_path*, uid_t *uid*, gid_t *gid*)

Parameters:

uid set owner of soft-link to uid

gid set group of soft-link to gid

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

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

Parameters:

link_path path for soft-link or new file.

5.12.3.11 Arc::DataCache::operator bool (void) [inline]

Returns true if object is useable.

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

Set creation time.

Parameters:

val creation time

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

Set invalidation time.

Parameters:

val validity time

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

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

Parameters:

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

available contains true on exit if file is already in cache

5.12.3.15 bool Arc::DataCache::stop (int *file_state* = file_no_error)

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

Parameters:

failure true if download failed

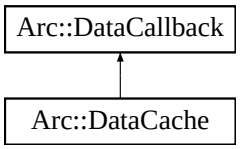
The documentation for this class was generated from the following file:

- DataCache.h

5.13 Arc::DataCallback Class Reference

`#include <DataCallback.h>`

Inheritance diagram for Arc::DataCallback::



Public Member Functions

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

5.13.1 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

5.14 Arc::DataHandle Class Reference

This class is a wrapper around the DataPoint class.

`#include <DataHandle.h>`

Public Member Functions

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

5.14.1 Detailed Description

This class is a wrapper around the DataPoint class.

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

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

- DataHandle.h

5.15 Arc::DataMover Class Reference

`#include <DataMover.h>`

Public Types

- typedef void(∗) **callback** (DataMover ∗, DataMover::result, const std::string &, void ∗)
- success = 0
- read_acquire_error = 1
- write_acquire_error = 2
- read_resolve_error = 3
- write_resolve_error = 4
- preregister_error = 5
- read_start_error = 6
- write_start_error = 7
- read_error = 8
- write_error = 9
- transfer_error = 10
- read_stop_error = 11
- write_stop_error = 12
- postregister_error = 13
- cache_error = 14
- system_error = 15
- credentials_expired_error = 16
- **delete_error** = 17
- **location_unregister_error** = 18
- **unregister_error** = 19
- undefined_error = -1
- enum result {

  success = 0, read_acquire_error = 1, write_acquire_error = 2, read_resolve_error = 3,

  write_resolve_error = 4, preregister_error = 5, read_start_error = 6, write_start_error = 7,

  read_error = 8, write_error = 9, transfer_error = 10, read_stop_error = 11,

  write_stop_error = 12, postregister_error = 13, cache_error = 14, system_error = 15,

  credentials_expired_error = 16, **delete_error** = 17, **location_unregister_error** = 18, **unregister_error** = 19,

  undefined_error = -1 }

Public Member Functions

- DataMover (void)
- ∼DataMover (void)
- result Transfer (DataPoint &source, DataPoint &destination, DataCache &cache, const URLMap &map, std::string &failure_description, callback cb=NULL, void ∗arg=NULL, const char ∗prefix=NULL)
- result Transfer (DataPoint &source, DataPoint &destination, DataCache &cache, const URLMap &map, unsigned long long int min_speed, time_t min_speed_time, unsigned long long int min_average_speed, time_t max_inactivity_time, std::string &failure_description, callback cb=NULL, void ∗arg=NULL, const char ∗prefix=NULL)

- result **Delete** (DataPoint &url, bool errcont=false)
- bool verbose (void)
- void verbose (bool)
- void verbose (const std::string &prefix)
- bool retry (void)
- void retry (bool)
- void secure (bool)
- void passive (bool)
- void force_to_meta (bool)
- bool checks (void)
- void checks (bool v)
- void set_default_min_speed (unsigned long long int min_speed, time_t min_speed_time)
- void set_default_min_average_speed (unsigned long long int min_average_speed)
- void set_default_max_inactivity_time (time_t max_inactivity_time)
- void **set_progress_indicator** (DataSpeed::show_progress_t func=NULL)

Static Public Member Functions

- static const char * **get_result_string** (result r)

5.15.1 Detailed Description

A purpose of this class is to provide interface moves data between 2 locations specified by URLs. It's main action is represented by methods DataMover::Transfer. Instance represents only attributes used during transfer.

5.15.2 Member Enumeration Documentation

5.15.2.1 enum Arc::DataMover::result

Error code/failure reason.

Enumerator:

success Operation completed successfully.

read_acquire_error Source is bad URL or can't be used due to some reason.

write_acquire_error Destination is bad URL or can't be used due to some reason.

read_resolve_error Resolving of meta-URL for source failed.

write_resolve_error Resolving of meta-URL for destination failed.

preregister_error First stage of registration of meta-URL failed.

read_start_error Can't read from source.

write_start_error Can't write to destination.

read_error Failed while reading from source.

write_error Failed while writing to destination.

transfer_error Failed while transfering data (mostly timeout).

read_stop_error Failed while finishing reading from source.

write_stop_error Failed while finishing writing to destination.

postregister_error Last stage of registration of meta-URL failed.

cache_error Error in caching procedure.

system_error Some system function returned unexpected error.

credentials_expired_error Error due to provided credentials are expired.

undefined_error Unknown/undefined error.

5.15.3 Constructor & Destructor Documentation

5.15.3.1 Arc::DataMover::DataMover (void)

Constructor.

5.15.3.2 Arc::DataMover::~DataMover (void)

Destructor.

5.15.4 Member Function Documentation

5.15.4.1 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).

5.15.4.2 bool Arc::DataMover::checks (void)

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

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

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

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

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

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

Set if transfer will be retried in case of failure.

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

Check if transfer will be retried in case of failure.

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

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

5.15.4.8 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.

5.15.4.9 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.

5.15.4.10 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.

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

Initiates transfer from 'source' to 'destination'.

Parameters:

min_speed minimal allowed current speed.

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

min_average_speed minimal allowed average speed.

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

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

Initiates transfer from 'source' to 'destination'.

Parameters:

source source URL.

destination destination URL.

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

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

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

arg passed to 'cb'.

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

5.15.4.13 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.

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

Activate printing information about transfer status.

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

Check if printing information about transfer status is activated.

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

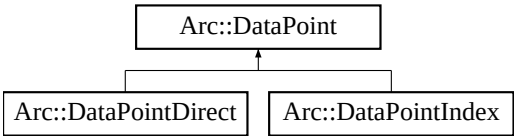
- DataMover.h

5.16 Arc::DataPoint Class Reference

This class is an abstraction of URL.

`#include <DataPoint.h>`

Inheritance diagram for Arc::DataPoint::



Public Types

- **common_failure** = 0
- **credentials_expired_failure** = 1
- enum failure_reason_t { **common_failure** = 0, **credentials_expired_failure** = 1 }

Public Member Functions

- DataPoint (const URL &url)
- virtual bool start_reading (DataBufferPar &buffer)=0
- virtual bool start_writing (DataBufferPar &buffer, DataCallback *space_cb=NULL)=0
- virtual bool stop_reading ()=0
- virtual bool stop_writing ()=0
- virtual bool analyze (analyze_t &arg)=0
- virtual bool check ()=0
- virtual bool local () const =0
- virtual bool remove ()=0
- virtual bool out_of_order ()=0
- virtual void out_of_order (bool v)=0
- virtual void additional_checks (bool v)=0
- virtual bool additional_checks ()=0
- virtual void secure (bool v)=0
- virtual bool secure ()=0
- virtual void passive (bool v)=0
- virtual failure_reason_t failure_reason ()=0
- virtual void range (unsigned long long int start=0, unsigned long long int end=0)=0
- virtual std::string **failure_text** ()=0
- virtual bool meta_resolve (bool source)=0
- virtual bool meta_preregister (bool replication, bool force=false)=0
- virtual bool meta_postregister (bool replication)=0
- virtual bool **meta_register** (bool replication)
- virtual bool meta_preunregister (bool replication)=0
- virtual bool meta_unregister (bool all)=0
- virtual bool get_info (FileInfo &fi)=0
- virtual bool CheckSize () const
- virtual void SetSize (const unsigned long long int val)

- virtual unsigned long long int GetSize () const
- virtual bool CheckCheckSum () const
- virtual void SetCheckSum (const std::string &val)
- virtual const std::string & GetCheckSum () const
- virtual bool CheckCreated () const
- virtual void SetCreated (const Time &val)
- virtual const Time & GetCreated () const
- virtual bool CheckValid () const
- virtual void SetValid (const Time &val)
- virtual const Time & GetValid () const
- virtual bool meta () const =0
- virtual bool accepts_meta ()=0
- virtual bool provides_meta ()=0
- virtual void meta (const DataPoint &p)
- virtual bool meta_compare (const DataPoint &p) const
- virtual bool meta_stored ()=0
- virtual **operator bool** () const
- virtual bool **operator!** () const
- virtual const URL & current_location () const =0
- virtual const std::string & current_meta_location () const =0
- virtual bool next_location ()=0
- virtual bool have_location () const =0
- virtual bool have_locations () const =0
- virtual bool add_location (const std::string &meta, const URL &loc)=0
- virtual bool remove_location ()=0
- virtual bool remove_locations (const DataPoint &p)=0
- virtual bool list_files (std::list< FileInfo > &files, bool resolve=true)=0
- virtual int GetTries () const
- virtual void SetTries (const int n)
- virtual const URL & base_url () const
- virtual std::string str () const

Protected Attributes

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

Static Protected Attributes

- static Logger **logger**

Classes

- class analyze_t

5.16.1 Detailed Description

This class is an abstraction of URL.

It can handle URLs of type file://, ftp://, gsiftp://, http://, https://, httpg:// (HTTP over GSI), se:// (NG web service over HTTPG) and meta-URLs (URLs of Infexing Services) rc://, rls://. DataPoint provides means to resolve meta-URL into multiple URLs and to loop through them.

5.16.2 Member Enumeration Documentation

5.16.2.1 enum Arc::DataPoint::failure_reason_t

Reason of transfer failure.

5.16.3 Constructor & Destructor Documentation

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

Constructor requires URL or meta-URL to be provided.

5.16.4 Member Function Documentation

5.16.4.1 virtual bool Arc::DataPoint::accepts_meta () [pure virtual]

If endpoint can have any use from meta information.

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

5.16.4.2 virtual bool Arc::DataPoint::add_location (const std::string & *meta*, const URL & *loc*) [pure virtual]

Add URL to list.

Parameters:

meta meta-name (name of location/service).

loc URL.

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

5.16.4.3 virtual bool Arc::DataPoint::additional_checks () [pure virtual]

Check if additional checks before 'reading' and 'writing' will be performed.

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

5.16.4.4 virtual void Arc::DataPoint::additional_checks (bool *v*) [pure virtual]

Allow/disallow 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).

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

5.16.4.5 virtual bool Arc::DataPoint::analyze (analyze_t & *arg*) [pure virtual]

Analyze url and provide hints.

Parameters:

arg returns suggested values.

5.16.4.6 virtual const URL& Arc::DataPoint::base_url () const [virtual]

Returns URL which was passed to constructor.

5.16.4.7 virtual bool Arc::DataPoint::check () [pure virtual]

Query remote server or local file system to check if object is accessible. If possible this function will also try to fill meta information about object in associated DataPoint.

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

5.16.4.8 virtual bool Arc::DataPoint::CheckCheckSum () const [inline, virtual]

Check if meta-information 'checksum' is available.

5.16.4.9 virtual bool Arc::DataPoint::CheckCreated () const [inline, virtual]

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

5.16.4.10 virtual bool Arc::DataPoint::CheckSize () const [inline, virtual]

Check if meta-information 'size' is available.

5.16.4.11 virtual bool Arc::DataPoint::CheckValid () const [inline, virtual]

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

5.16.4.12 virtual const URL& Arc::DataPoint::current_location () const [pure virtual]

Returns current (resolved) URL.

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

5.16.4.13 virtual const std::string& Arc::DataPoint::current_meta_location () const [pure virtual]

Returns meta information used to create curent URL. For RC that is location's name. For RLS that is equal to pfn.

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

5.16.4.14 virtual failure_reason_t Arc::DataPoint::failure_reason () [pure virtual]

Returns reason of transfer failure.

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

5.16.4.15 virtual bool Arc::DataPoint::get_info (FileInfo & *fi*) [pure virtual]

Retrieve properties of object pointed by meta-URL of DataPoint object. It works only for meta-URL.

Parameters:

fi contains retrieved information.

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

5.16.4.16 virtual const std::string& Arc::DataPoint::GetCheckSum () const [inline, virtual]

Get value of meta-information 'checksum'.

5.16.4.17 virtual const Time& Arc::DataPoint::GetCreated () const [inline, virtual]

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

5.16.4.18 virtual unsigned long long int Arc::DataPoint::GetSize () const [inline, virtual]

Get value of meta-information 'size'.

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

Returns number of retries left.

5.16.4.20 virtual const Time& Arc::DataPoint::GetValid () const [inline, virtual]

Get value of meta-information 'validity time'.

5.16.4.21 virtual bool Arc::DataPoint::have_location () const [pure virtual]

Returns false if out of retries.

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

5.16.4.22 virtual bool Arc::DataPoint::have_locations () const [pure virtual]

Returns true if number of resolved URLs is not 0.

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

5.16.4.23 virtual bool Arc::DataPoint::list_files (std::list< FileInfo > & *files*, bool *resolve* = true) [pure virtual]

List files in directory or service.

Parameters:

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

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

Implemented in Arc::DataPointDirect.

5.16.4.24 virtual bool Arc::DataPoint::local () const [pure virtual]

Check if file is local (URL is something like file://).

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

5.16.4.25 virtual void Arc::DataPoint::meta (const DataPoint & *p*) [inline, virtual]

Acquire meta-information from another object. Defined values are not overwritten.

Parameters:

p object from which information is taken.

5.16.4.26 virtual bool Arc::DataPoint::meta () const [pure virtual]

Check if URL is meta-URL.

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

5.16.4.27 virtual bool Arc::DataPoint::meta_compare (const DataPoint & *p*) const [inline, virtual]

Compare meta-information form another object. Undefined values are not used for comparison. Default result is 'true'.

Parameters:

p object to which compare.

5.16.4.28 virtual bool Arc::DataPoint::meta_postregister (bool *replication*) [pure virtual]

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

Parameters:

replication if true then file is being replicated between 2 locations registered in Indexing Service under same name.

Implemented in Arc::DataPointDirect.

5.16.4.29 virtual bool Arc::DataPoint::meta_preregister (bool *replication*, bool *force* = false) [pure virtual]

This function registers physical location of file into Indexing Service. It should be called ∗before∗ actual transfer to that location happens.

Parameters:

replication if true then file is being replicated between 2 locations registered in Indexing Service under same name.

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

Implemented in Arc::DataPointDirect.

5.16.4.30 virtual bool Arc::DataPoint::meta_preunregister (bool *replication*) [pure virtual]

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

Implemented in Arc::DataPointDirect.

5.16.4.31 virtual bool Arc::DataPoint::meta_resolve (bool *source*) [pure virtual]

Resolve meta-URL into list of ordinary URLs and obtain meta-information about file. Can be called for object representing ordinary URL or already resolved object.

Parameters:

source true if DataPoint object represents source of information

Implemented in Arc::DataPointDirect.

5.16.4.32 virtual bool Arc::DataPoint::meta_stored () [pure virtual]

Check if file is registered in Indexing Service. Proper value is obtainable only after meta-resolve.

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

5.16.4.33 virtual bool Arc::DataPoint::meta_unregister (bool *all*) [pure virtual]

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.

5.16.4.34 virtual bool Arc::DataPoint::next_location () [pure virtual]

Switch to next location in list of URLs. At last location switch to first if number of allowed retries does not exceeded. Returns false if no retries left.

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

5.16.4.35 virtual void Arc::DataPoint::out_of_order (bool *v*) [pure virtual]

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

Parameters:

v true if allowed.

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

5.16.4.36 virtual bool Arc::DataPoint::out_of_order () [pure virtual]

Returns true if URL can accept scatterd data (like arbitrary access to local file) for 'writing' operation.

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

5.16.4.37 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.

5.16.4.38 virtual bool Arc::DataPoint::provides_meta () [pure virtual]

If endpoint can provide at least some meta information directly.

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

5.16.4.39 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.

5.16.4.40 virtual bool Arc::DataPoint::remove () [pure virtual]

Remove/delete object at URL.

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

5.16.4.41 virtual bool Arc::DataPoint::remove_location () [pure virtual]

Remove current URL from list.

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

5.16.4.42 virtual bool Arc::DataPoint::remove_locations (const DataPoint & *p*) [pure virtual]

Remove locations present in another DataPoint object.

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

5.16.4.43 virtual bool Arc::DataPoint::secure () [pure virtual]

Check if heavy security during data transfer is allowed.

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

5.16.4.44 virtual void Arc::DataPoint::secure (bool *v*) [pure virtual]

Allow/disallow heavy security during data transfer.

Parameters:

v true if allowed (default is true only for gsiftp://).

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

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

Set value of meta-information 'checksum'.

5.16.4.46 virtual void Arc::DataPoint::SetCreated (const Time & *val*) [inline, virtual]

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

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

Set value of meta-information 'size'.

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

Set number of retries.

Reimplemented in Arc::DataPointIndex.

5.16.4.49 virtual void Arc::DataPoint::SetValid (const Time & *val*) [inline, virtual]

Set value of meta-information 'validity time'.

5.16.4.50 virtual bool Arc::DataPoint::start_reading (DataBufferPar & *buffer*) [pure virtual]

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

Parameters:

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

Returns:

true on success.

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

5.16.4.51 virtual bool Arc::DataPoint::start_writing (DataBufferPar & *buffer*, DataCallback * *space_cb* = NULL) [pure virtual]

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

Parameters:

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

space_cb callback which is called if there is not enough to space storing data. Currently implemented only for file:/// URL.

Returns:

true on success.

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

5.16.4.52 virtual bool Arc::DataPoint::stop_reading () [pure virtual]

Stop reading. It MUST be called after corressponding start_reading method. Either after whole data is transfered or to cancel transfer. Use 'buffer' object to find out when data is transfered.

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

5.16.4.53 virtual bool Arc::DataPoint::stop_writing () [pure virtual]

Same as stop_reading but for corresponding start_writing.

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

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

Returns a string representation of the DataPoint.

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

- DataPoint.h

5.17 Arc::DataPoint::analyze_t Class Reference

`#include <DataPoint.h>`

Public Attributes

- long int **bufsize**
- int **bufnum**
- bool **cache**
- bool **local**
- bool **readonly**

5.17.1 Detailed Description

Structure used in analyze() call.

Parameters:

bufsize returns suggested size of buffers to store data.

bufnum returns suggested number of buffers.

cache returns true if url is allowed to be cached.

local return true if URL is accessed locally (file://)

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

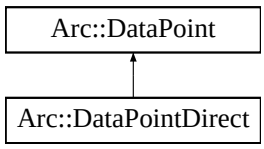
- DataPoint.h

5.18 Arc::DataPointDirect Class Reference

This is kind of generalized file handle.

`#include <DataPointDirect.h>`

Inheritance diagram for Arc::DataPointDirect::



Public Member Functions

- DataPointDirect (const URL &url)
- virtual ~DataPointDirect ()
- virtual bool meta () const
- virtual bool start_reading (DataBufferPar &buffer)
- virtual bool start_writing (DataBufferPar &buffer, DataCallback *space_cb=NULL)
- virtual bool stop_reading ()
- virtual bool stop_writing ()
- virtual bool **analyze** (analyze_t &arg)
- virtual bool check ()
- virtual bool remove ()
- virtual bool list_files (std::list< FileInfo > &files, bool resolve=true)
- virtual bool out_of_order ()
- virtual void out_of_order (bool v)
- virtual void additional_checks (bool v)
- virtual bool additional_checks ()
- virtual void secure (bool v)
- virtual bool secure ()
- virtual void passive (bool v)
- virtual failure_reason_t failure_reason ()
- virtual std::string **failure_text** ()
- virtual void range (unsigned long long int start=0, unsigned long long int end=0)
- virtual bool meta_resolve (bool)
- virtual bool meta_preregister (bool, bool force=false)
- virtual bool meta_postregister (bool)
- virtual bool meta_preunregister (bool)
- virtual bool meta_unregister (bool all)
- virtual bool get_info (FileInfo &fi)
- virtual bool accepts_meta ()
- virtual bool provides_meta ()
- virtual bool meta_stored ()
- virtual bool local () const
- virtual const URL & current_location () const
- virtual const std::string & current_meta_location () const
- virtual bool next_location ()

- virtual bool have_location () const
- virtual bool have_locations () const
- virtual bool add_location (const std::string &, const URL &)
- virtual bool remove_location ()
- virtual bool remove_locations (const DataPoint &p)

Protected Member Functions

- virtual bool **init_handle** ()
- virtual bool **deinit_handle** ()

Protected Attributes

- DataBufferPar * **buffer**
- bool **cacheable**
- bool **linkable**
- bool **is_secure**
- bool **force_secure**
- bool **force_passive**
- bool **reading**
- bool **writing**
- bool **no_checks**
- bool **allow_out_of_order**
- int **transfer_streams**
- unsigned long long int **range_start**
- unsigned long long int **range_end**
- failure_reason_t **failure_code**
- std::string **failure_description**

5.18.1 Detailed Description

This is kind of generalized file handle.

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

5.18.2 Constructor & Destructor Documentation

5.18.2.1 Arc::DataPointDirect::DataPointDirect (const URL & *url*)

Constructor

Parameters:

url URL.

5.18.2.2 virtual Arc::DataPointDirect::~DataPointDirect () [virtual]

Destructor. No comments.

5.18.3 Member Function Documentation

5.18.3.1 virtual bool Arc::DataPointDirect::accepts_meta () [inline, virtual]

If endpoint can have any use from meta information.

Implements Arc::DataPoint.

5.18.3.2 virtual bool Arc::DataPointDirect::add_location (const std::string &, const URL &) [inline, virtual]

Add URL to list.

Parameters:

meta meta-name (name of location/service).

loc URL.

Implements Arc::DataPoint.

5.18.3.3 virtual bool Arc::DataPointDirect::additional_checks () [virtual]

Check if additional checks before 'reading' and 'writing' will be performed.

Implements Arc::DataPoint.

5.18.3.4 virtual void Arc::DataPointDirect::additional_checks (bool *v*) [virtual]

Allow/disallow 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).

Implements Arc::DataPoint.

5.18.3.5 virtual bool Arc::DataPointDirect::check () [virtual]

Query remote server or local file system to check if object is accessible. If possible this function will also try to fill meta information about object in associated DataPoint.

Implements Arc::DataPoint.

5.18.3.6 virtual const URL& Arc::DataPointDirect::current_location () const [virtual]

Returns current (resolved) URL.

Implements Arc::DataPoint.

5.18.3.7 virtual const std::string& Arc::DataPointDirect::current_meta_location () const [virtual]

Returns meta information used to create curent URL. For RC that is location's name. For RLS that is equal to pfn.

Implements Arc::DataPoint.

5.18.3.8 virtual failure_reason_t Arc::DataPointDirect::failure_reason () [virtual]

Returns reason of transfer failure.

Implements Arc::DataPoint.

5.18.3.9 virtual bool Arc::DataPointDirect::get_info (FileInfo & *fi*) [inline, virtual]

Retrieve properties of object pointed by meta-URL of DataPoint object. It works only for meta-URL.

Parameters:

fi contains retrieved information.

Implements Arc::DataPoint.

5.18.3.10 virtual bool Arc::DataPointDirect::have_location () const [inline, virtual]

Returns false if out of retries.

Implements Arc::DataPoint.

5.18.3.11 virtual bool Arc::DataPointDirect::have_locations () const [inline, virtual]

Returns true if number of resolved URLs is not 0.

Implements Arc::DataPoint.

5.18.3.12 virtual bool Arc::DataPointDirect::list_files (std::list< FileInfo > & *files*, bool *resolve* = true) [virtual]

List files in directory or service.

Parameters:

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

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

Implements Arc::DataPoint.

5.18.3.13 virtual bool Arc::DataPointDirect::local () const [inline, virtual]

Check if file is local (URL is something like file://).

Implements Arc::DataPoint.

5.18.3.14 virtual bool Arc::DataPointDirect::meta () const [inline, virtual]

Check if URL is meta-URL.

Implements Arc::DataPoint.

5.18.3.15 virtual bool Arc::DataPointDirect::meta_postregister (bool) [inline, virtual]

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

Parameters:

replication if true then file is being replicated between 2 locations registered in Indexing Service under same name.

Implements Arc::DataPoint.

5.18.3.16 virtual bool Arc::DataPointDirect::meta_preregister (bool, bool *force* = false) [inline, virtual]

This function registers physical location of file into Indexing Service. It should be called ∗before∗ actual transfer to that location happens.

Parameters:

replication if true then file is being replicated between 2 locations registered in Indexing Service under same name.

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

Implements Arc::DataPoint.

5.18.3.17 virtual bool Arc::DataPointDirect::meta_preunregister (bool) [inline, virtual]

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

Implements Arc::DataPoint.

5.18.3.18 virtual bool Arc::DataPointDirect::meta_resolve (bool) [inline, virtual]

Resolve meta-URL into list of ordinary URLs and obtain meta-information about file. Can be called for object representing ordinary URL or already resolved object.

Parameters:

source true if DataPoint object represents source of information

Implements Arc::DataPoint.

5.18.3.19 virtual bool Arc::DataPointDirect::meta_stored () [inline, virtual]

Check if file is registered in Indexing Service. Proper value is obtainable only after meta-resolve.

Implements Arc::DataPoint.

5.18.3.20 virtual bool Arc::DataPointDirect::meta_unregister (bool *all*) [inline, virtual]

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.

5.18.3.21 virtual bool Arc::DataPointDirect::next_location () [inline, virtual]

Switch to next location in list of URLs. At last location switch to first if number of allowed retries does not exceeded. Returns false if no retries left.

Implements Arc::DataPoint.

5.18.3.22 virtual void Arc::DataPointDirect::out_of_order (bool *v*) [virtual]

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

Parameters:

v true if allowed.

Implements Arc::DataPoint.

5.18.3.23 virtual bool Arc::DataPointDirect::out_of_order () [virtual]

Returns true if URL can accept scatterd data (like arbitrary access to local file) for 'writing' operation.

Implements Arc::DataPoint.

5.18.3.24 virtual void Arc::DataPointDirect::passive (bool *v*) [virtual]

Request passive transfers for FTP-like protocols.

Parameters:

true to request.

Implements Arc::DataPoint.

5.18.3.25 virtual bool Arc::DataPointDirect::provides_meta () [inline, virtual]

If endpoint can provide at least some meta information directly.

Implements Arc::DataPoint.

5.18.3.26 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.

5.18.3.27 virtual bool Arc::DataPointDirect::remove () [virtual]

Remove/delete object at URL.

Implements Arc::DataPoint.

5.18.3.28 virtual bool Arc::DataPointDirect::remove_location () [inline, virtual]

Remove current URL from list.

Implements Arc::DataPoint.

5.18.3.29 virtual bool Arc::DataPointDirect::remove_locations (const DataPoint & *p*) [inline, virtual]

Remove locations present in another DataPoint object.

Implements Arc::DataPoint.

5.18.3.30 virtual bool Arc::DataPointDirect::secure () [virtual]

Check if heavy security during data transfer is allowed.

Implements Arc::DataPoint.

5.18.3.31 virtual void Arc::DataPointDirect::secure (bool *v*) [virtual]

Allow/disallow heavy security during data transfer.

Parameters:

v true if allowed (default is true only for gsiftp://).

Implements Arc::DataPoint.

5.18.3.32 virtual bool Arc::DataPointDirect::start_reading (DataBufferPar & *buffer*) [virtual]

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

Parameters:

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

Returns:

true on success.

Implements Arc::DataPoint.

5.18.3.33 virtual bool Arc::DataPointDirect::start_writing (DataBufferPar & *buffer*, DataCallback * *space_cb* = NULL) [virtual]

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

Parameters:

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

space_cb callback which is called if there is not enough to space storing data. Currently implemented only for file:/// URL.

Returns:

true on success.

Implements Arc::DataPoint.

5.18.3.34 virtual bool Arc::DataPointDirect::stop_reading () [virtual]

Stop reading. It MUST be called after corressponding start_reading method. Either after whole data is transfered or to cancel transfer. Use 'buffer' object to find out when data is transfered.

Implements Arc::DataPoint.

5.18.3.35 virtual bool Arc::DataPointDirect::stop_writing () [virtual]

Same as stop_reading but for corresponding start_writing.

Implements Arc::DataPoint.

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

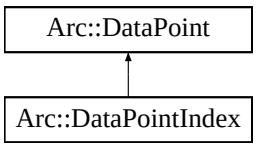
- DataPointDirect.h

5.19 Arc::DataPointIndex Class Reference

Complements DataPoint with attributes common for meta-URLs.

`#include <DataPointIndex.h>`

Inheritance diagram for Arc::DataPointIndex::



Public Member Functions

- **DataPointIndex** (const URL &url)
- virtual bool get_info (FileInfo &fi)
- virtual const URL & current_location () const
- virtual const std::string & current_meta_location () const
- virtual bool next_location ()
- virtual bool have_location () const
- virtual bool have_locations () const
- virtual bool remove_location ()
- virtual bool remove_locations (const DataPoint &p)
- virtual bool add_location (const std::string &meta, const URL &loc)
- virtual bool meta () const
- virtual bool accepts_meta ()
- virtual bool provides_meta ()
- virtual bool meta_stored ()
- virtual void SetTries (const int n)
- virtual bool start_reading (DataBufferPar &buffer)
- virtual bool start_writing (DataBufferPar &buffer, DataCallback *space_cb=NULL)
- virtual bool stop_reading ()
- virtual bool stop_writing ()
- virtual bool **analyze** (analyze_t &arg)
- virtual bool check ()
- virtual bool local () const
- virtual bool remove ()
- virtual void out_of_order (bool v)
- virtual bool out_of_order ()
- virtual void additional_checks (bool v)
- virtual bool additional_checks ()
- virtual void secure (bool v)
- virtual bool secure ()
- virtual void passive (bool v)
- virtual failure_reason_t failure_reason ()
- virtual std::string **failure_text** ()
- virtual void range (unsigned long long int start=0, unsigned long long int end=0)

Protected Attributes

- std::list< URLLocation > locations
- std::list< URLLocation >::iterator **location**
- DataHandle **h**
- bool **is_metaexisting**
- bool **is_resolved**

5.19.1 Detailed Description

Complements DataPoint with attributes common for meta-URLs.

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

5.19.2 Member Function Documentation

5.19.2.1 virtual bool Arc::DataPointIndex::accepts_meta () [inline, virtual]

If endpoint can have any use from meta information.

Implements Arc::DataPoint.

5.19.2.2 virtual bool Arc::DataPointIndex::add_location (const std::string & *meta*, const URL & *loc*) [virtual]

Add URL to list.

Parameters:

meta meta-name (name of location/service).

loc URL.

Implements Arc::DataPoint.

5.19.2.3 virtual bool Arc::DataPointIndex::additional_checks () [virtual]

Check if additional checks before 'reading' and 'writing' will be performed.

Implements Arc::DataPoint.

5.19.2.4 virtual void Arc::DataPointIndex::additional_checks (bool *v*) [virtual]

Allow/disallow 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).

Implements Arc::DataPoint.

5.19.2.5 virtual bool Arc::DataPointIndex::check () [virtual]

Query remote server or local file system to check if object is accessible. If possible this function will also try to fill meta information about object in associated DataPoint.

Implements Arc::DataPoint.

5.19.2.6 virtual const URL& Arc::DataPointIndex::current_location () const [virtual]

Returns current (resolved) URL.

Implements Arc::DataPoint.

5.19.2.7 virtual const std::string& Arc::DataPointIndex::current_meta_location () const [virtual]

Returns meta information used to create curent URL. For RC that is location's name. For RLS that is equal to pfn.

Implements Arc::DataPoint.

5.19.2.8 virtual failure_reason_t Arc::DataPointIndex::failure_reason () [virtual]

Returns reason of transfer failure.

Implements Arc::DataPoint.

5.19.2.9 virtual bool Arc::DataPointIndex::get_info (FileInfo & *fi*) [virtual]

Retrieve properties of object pointed by meta-URL of DataPoint object. It works only for meta-URL.

Parameters:

fi contains retrieved information.

Implements Arc::DataPoint.

5.19.2.10 virtual bool Arc::DataPointIndex::have_location () const [virtual]

Returns false if out of retries.

Implements Arc::DataPoint.

5.19.2.11 virtual bool Arc::DataPointIndex::have_locations () const [virtual]

Returns true if number of resolved URLs is not 0.

Implements Arc::DataPoint.

5.19.2.12 virtual bool Arc::DataPointIndex::local () const [virtual]

Check if file is local (URL is something like file://).

Implements Arc::DataPoint.

5.19.2.13 virtual bool Arc::DataPointIndex::meta () const [inline, virtual]

Check if URL is meta-URL.

Implements Arc::DataPoint.

5.19.2.14 virtual bool Arc::DataPointIndex::meta_stored () [inline, virtual]

Check if file is registered in Indexing Service. Proper value is obtainable only after meta-resolve.

Implements Arc::DataPoint.

5.19.2.15 virtual bool Arc::DataPointIndex::next_location () [virtual]

Switch to next location in list of URLs. At last location switch to first if number of allowed retries does not exceeded. Returns false if no retries left.

Implements Arc::DataPoint.

5.19.2.16 virtual bool Arc::DataPointIndex::out_of_order () [virtual]

Returns true if URL can accept scatterd data (like arbitrary access to local file) for 'writing' operation.

Implements Arc::DataPoint.

5.19.2.17 virtual void Arc::DataPointIndex::out_of_order (bool *v*) [virtual]

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

Parameters:

v true if allowed.

Implements Arc::DataPoint.

5.19.2.18 virtual void Arc::DataPointIndex::passive (bool *v*) [virtual]

Request passive transfers for FTP-like protocols.

Parameters:

true to request.

Implements Arc::DataPoint.

5.19.2.19 virtual bool Arc::DataPointIndex::provides_meta () [inline, virtual]

If endpoint can provide at least some meta information directly.

Implements Arc::DataPoint.

5.19.2.20 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.

5.19.2.21 virtual bool Arc::DataPointIndex::remove () [virtual]

Remove/delete object at URL.

Implements Arc::DataPoint.

5.19.2.22 virtual bool Arc::DataPointIndex::remove_location () [virtual]

Remove current URL from list.

Implements Arc::DataPoint.

5.19.2.23 virtual bool Arc::DataPointIndex::remove_locations (const DataPoint & *p*) [virtual]

Remove locations present in another DataPoint object.

Implements Arc::DataPoint.

5.19.2.24 virtual bool Arc::DataPointIndex::secure () [virtual]

Check if heavy security during data transfer is allowed.

Implements Arc::DataPoint.

5.19.2.25 virtual void Arc::DataPointIndex::secure (bool *v*) [virtual]

Allow/disallow heavy security during data transfer.

Parameters:

v true if allowed (default is true only for gsiftp://).

Implements Arc::DataPoint.

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

Set number of retries.

Reimplemented from Arc::DataPoint.

5.19.2.27 virtual bool Arc::DataPointIndex::start_reading (DataBufferPar & *buffer*) [virtual]

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

Parameters:

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

Returns:

true on success.

Implements Arc::DataPoint.

5.19.2.28 virtual bool Arc::DataPointIndex::start_writing (DataBufferPar & *buffer*, DataCallback * *space_cb* = NULL) [virtual]

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

Parameters:

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

space_cb callback which is called if there is not enough to space storing data. Currently implemented only for file:/// URL.

Returns:

true on success.

Implements Arc::DataPoint.

5.19.2.29 virtual bool Arc::DataPointIndex::stop_reading () [virtual]

Stop reading. It MUST be called after corressponding start_reading method. Either after whole data is transfered or to cancel transfer. Use 'buffer' object to find out when data is transfered.

Implements Arc::DataPoint.

5.19.2.30 virtual bool Arc::DataPointIndex::stop_writing () [virtual]

Same as stop_reading but for corresponding start_writing.

Implements Arc::DataPoint.

5.19.3 Member Data Documentation

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

List of locations at which file can be probably found.

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

- DataPointIndex.h

5.20 Arc::DataSpeed Class Reference

Keeps track of average and instantaneous transfer speed.

`#include <DataSpeed.h>`

Public Types

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

Public Member Functions

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

5.20.1 Detailed Description

Keeps track of average and instantaneous transfer speed.

Also detects data transfer inactivity and other transfer timeouts.

5.20.2 Constructor & Destructor Documentation

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

Constructor

Parameters:

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

5.20.2.2 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.

5.20.2.3 Arc::DataSpeed::~DataSpeed (void)

Destructor.

5.20.3 Member Function Documentation

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

Turn off speed control.

Parameters:

disable true to turn off.

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

Check if maximal inactivity time error was triggered.

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

Check if minimal average speed error was triggered.

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

Check if minimal speed error was triggered.

5.20.3.5 void Arc::DataSpeed::reset (void)

Reset all counters and triggers.

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

Set averaging time period.

Parameters:

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

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

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

Parameters:

max amount of data in bytes.

5.20.3.8 void Arc::DataSpeed::set_max_inactivity_time (time_t *max_inactivity_time*)

Set inactivity tiemout.

Parameters:

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

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

Set minmal avaerage speed.

Parameters:

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

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

Set minimal allowed speed.

Parameters:

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

min_speed_time

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

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

Parameters:

pointer to function which prints information.

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

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

Parameters:

n amount of data transfered (bytes).

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

Returns amount of data this object knows about.

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

Check if speed information is going to be printed.

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

Print information about current speed and amout of data.

Parameters:

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

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

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

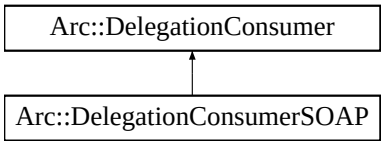
The documentation for this class was generated from the following file:

- DataSpeed.h

5.21 Arc::DelegationConsumer Class Reference

`#include <DelegationInterface.h>`

Inheritance diagram for Arc::DelegationConsumer::



Public Member Functions

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

Protected Member Functions

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

Protected Attributes

- void * **key_**

5.21.1 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.

5.21.2 Constructor & Destructor Documentation

5.21.2.1 Arc::DelegationConsumer::DelegationConsumer (void)

Creates object with new private key

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

Creates object with provided private key

5.21.3 Member Function Documentation

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

Ads private key into certificates chain in 'content'

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

Stores content of this object into a string

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

Private key

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

Return identifier of this object - not implemented

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

Creates private key

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

Make X509 certificate request from internal private key

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

Restores content of object from string

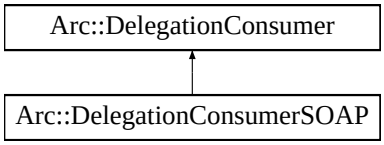
The documentation for this class was generated from the following file:

- DelegationInterface.h

5.22 Arc::DelegationConsumerSOAP Class Reference

`#include <DelegationInterface.h>`

Inheritance diagram for Arc::DelegationConsumerSOAP::



Public Member Functions

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

5.22.1 Detailed Description

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

5.22.2 Constructor & Destructor Documentation

5.22.2.1 Arc::DelegationConsumerSOAP::DelegationConsumerSOAP (void)

Creates object with new private key

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

Creates object with specified private key

5.22.3 Member Function Documentation

5.22.3.1 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.

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

Similar to UpdateCredentials but takes only DelegatedToken XML element

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

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

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

- DelegationInterface.h

5.23 Arc::DelegationContainerSOAP Class Reference

`#include <DelegationInterface.h>`

Public Member Functions

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

Protected Attributes

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

5.23.1 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.

5.23.2 Member Function Documentation

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

See DelegationConsumerSOAP::DelegateCredentialsInit

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

See DelegationConsumerSOAP::DelegatedToken

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

See DelegationConsumerSOAP::UpdateCredentials

5.23.3 Member Data Documentation

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

If true delegation consumer is deleted when connection context is destroyed

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

Lifetime of unused delegation consumer

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

Max. number of delegation consumers

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

Max. times same delegation consumer may accept credentials

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

If true all delegation phases must be performed by same identity

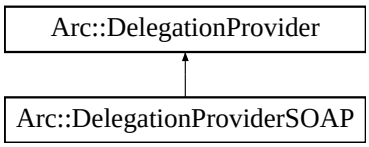
The documentation for this class was generated from the following file:

- DelegationInterface.h

5.24 Arc::DelegationProvider Class Reference

`#include <DelegationInterface.h>`

Inheritance diagram for Arc::DelegationProvider::



Public Member Functions

- DelegationProvider (const std::string &credentials)
- std::string Delegate (const std::string &request)

5.24.1 Detailed Description

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

5.24.2 Constructor & Destructor Documentation

5.24.2.1 Arc::DelegationProvider::DelegationProvider (const std::string & *credentials*)

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

5.24.3 Member Function Documentation

5.24.3.1 std::string Arc::DelegationProvider::Delegate (const std::string & *request*)

Perform delegation. Takes X509 certificate request and creates proxy credentials excluding private key. Result is then fed into DelegationConsumer::Acquire

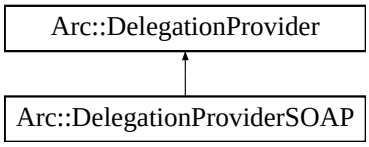
The documentation for this class was generated from the following file:

- DelegationInterface.h

5.25 Arc::DelegationProviderSOAP Class Reference

`#include <DelegationInterface.h>`

Inheritance diagram for Arc::DelegationProviderSOAP::



Public Member Functions

- DelegationProviderSOAP (const std::string &credentials)
- bool DelegateCredentialsInit (MCCInterface &interface, MessageContext *context)
- bool DelegateCredentialsInit (MCCInterface &interface, MessageAttributes *attributes_in, MessageAttributes *attributes_out, MessageContext *context)
- bool UpdateCredentials (MCCInterface &interface, MessageContext *context)
- bool UpdateCredentials (MCCInterface &interface, MessageAttributes *attributes_in, MessageAttributes *attributes_out, MessageContext *context)
- bool DelegatedToken (XMLNode &parent)

Protected Attributes

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

5.25.1 Detailed Description

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

5.25.2 Constructor & Destructor Documentation

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

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

5.25.3 Member Function Documentation

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

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

5.25.3.2 bool Arc::DelegationProviderSOAP::DelegateCredentialsInit (MCCInterface & *interface*, MessageContext * *context*)

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

5.25.3.3 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.

5.25.3.4 bool Arc::DelegationProviderSOAP::UpdateCredentials (MCCInterface & *interface*, MessageAttributes * *attributes_in*, MessageAttributes * *attributes_out*, MessageContext * *context*)

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

5.25.3.5 bool Arc::DelegationProviderSOAP::UpdateCredentials (MCCInterface & *interface*, MessageContext * *context*)

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

5.26 dmc_descriptor Struct Reference

`#include <DMCLoader.h>`

Public Attributes

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

5.26.1 Detailed Description

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

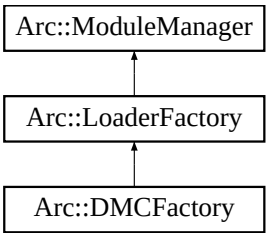
The documentation for this struct was generated from the following file:

- DMCLoader.h

5.27 Arc::DMCFactory Class Reference

`#include <DMCFactory.h>`

Inheritance diagram for Arc::DMCFactory::



Public Member Functions

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

5.27.1 Detailed Description

This class handles shared libraries containing DMCs

5.27.2 Constructor & Destructor Documentation

5.27.2.1 Arc::DMCFactory::DMCFactory (Config * *cfg*)

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

5.27.3 Member Function Documentation

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

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

Reimplemented from Arc::LoaderFactory.

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

- DMCFactory.h

5.28 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

5.28.1 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.

5.28.2 Member Function Documentation

5.28.2.1 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.

5.28.2.2 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.

5.28.2.3 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.

5.28.3 Friends And Related Function Documentation

5.28.3.1 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

5.29 Arc::FileInfo Class Reference

FileInfo stores information about files (metadata).

`#include <FileInfo.h>`

Public Types

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

Public Member Functions

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

5.29.1 Detailed Description

FileInfo stores information about files (metadata).

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

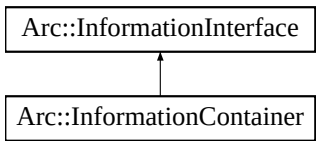
- FileInfo.h

5.30 Arc::InformationContainer Class Reference

Information System document container and processor.

`#include <InformationInterface.h>`

Inheritance diagram for Arc::InformationContainer::



Public Member Functions

- InformationContainer (XMLNode doc, bool copy=false)
- XMLNode Acquire (void)
- void **Release** (void)
- void Assign (XMLNode doc, bool copy=false)

Protected Member Functions

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

Protected Attributes

- XMLNode doc_

5.30.1 Detailed Description

Information System document container and processor.

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

5.30.2 Constructor & Destructor Documentation

5.30.2.1 Arc::InformationContainer::InformationContainer (XMLNode *doc*, bool *copy* = `false`)

Cretes an instance with XML document . If is true this method makes a copy of for intenal use.

5.30.3 Member Function Documentation

5.30.3.1 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()

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

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

5.30.3.3 virtual std::list<XMLNode> Arc::InformationContainer::Get (const std::list< std::string > & *path*) `[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.

Reimplemented from Arc::InformationInterface.

5.30.4 Member Data Documentation

5.30.4.1 XMLNode Arc::InformationContainer::doc_ `[protected]`

Either link or container of XML document

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

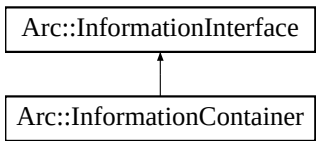
- InformationInterface.h

5.31 Arc::InformationInterface Class Reference

Information System message processor.

`#include <InformationInterface.h>`

Inheritance diagram for Arc::InformationInterface::



Public Member Functions

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

Protected Member Functions

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

Protected Attributes

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

5.31.1 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.

5.31.2 Constructor & Destructor Documentation

5.31.2.1 Arc::InformationInterface::InformationInterface (bool *safe* = `true`)

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

5.31.3 Member Function Documentation

5.31.3.1 virtual std::list<XMLNode> Arc::InformationInterface::Get (const std::list< std::string > & *path*) `[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.

Reimplemented in Arc::InformationContainer.

5.31.4 Member Data Documentation

5.31.4.1 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

5.32 Arc::InformationRequest Class Reference

Request for information in InfoSystem.

`#include <InformationInterface.h>`

Public Member Functions

- InformationRequest (void)
- InformationRequest (const std::list< std::string > &path)
- InformationRequest (const std::list< std::list< std::string > > &paths)
- InformationRequest (XMLNode query)
- **operator bool** (void)
- bool **operator!** (void)
- SOAPEnvelope * SOAP (void)

5.32.1 Detailed Description

Request for information in InfoSystem.

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

5.32.2 Constructor & Destructor Documentation

5.32.2.1 Arc::InformationRequest::InformationRequest (void)

Dummy constructor

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

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

5.32.2.3 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.

5.32.2.4 Arc::InformationRequest::InformationRequest (XMLNode *query*)

Request for attributes specified by XPath query.

5.32.3 Member Function Documentation

5.32.3.1 SOAPEnvelope* Arc::InformationRequest::SOAP (void)

Returns generated SOAP message

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

- InformationInterface.h

5.33 Arc::InformationResponse Class Reference

Informational response from InfoSystem.

`#include <InformationInterface.h>`

Public Member Functions

- InformationResponse (SOAPEnvelope &soap)
- **operator bool** (void)
- bool **operator!** (void)
- std::list< XMLNode > Result (void)

5.33.1 Detailed Description

Informational response from InfoSystem.

This is a convenience wrapper analyzing WS-ResourceProperties response from InfoSystem interface of service.

5.33.2 Constructor & Destructor Documentation

5.33.2.1 Arc::InformationResponse::InformationResponse (SOAPEnvelope & *soap*)

Constructor parses WS-ResourceProperties ressponse. Provided SOAPEnvelope object must be valid as long as this object is in use.

5.33.3 Member Function Documentation

5.33.3.1 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:

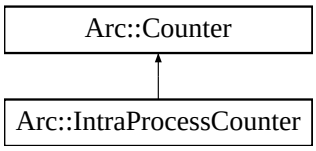
- InformationInterface.h

5.34 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)

5.34.1 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.

5.34.2 Constructor & Destructor Documentation

5.34.2.1 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.

5.34.2.2 virtual Arc::IntraProcessCounter::~IntraProcessCounter () [virtual]

Destructor.

This is the destructor of the IntraProcessCounter class. Does not need to do anything.

5.34.3 Member Function Documentation

5.34.3.1 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.

5.34.3.2 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.

5.34.3.3 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.

5.34.3.4 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.

5.34.3.5 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.

5.34.3.6 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.

5.34.3.7 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.

5.34.3.8 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.

5.34.3.9 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.

5.34.3.10 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

5.35 Arc::Loader Class Reference

Creator of Message Component Chains (MCC).

`#include <Loader.h>`

Public Types

- typedef std::map< std::string, MCC * > **mcc_container_t**
- typedef std::map< std::string, Service * > **service_container_t**
- typedef std::map< std::string, ArcSec::SecHandler * > **sechandler_container_t**
- typedef std::map< std::string, DMC * > **dmc_container_t**
- typedef std::map< std::string, Plexer * > **plexer_container_t**

Public Member Functions

- Loader (Config *cfg)
- ~Loader ()
- MCC * operator[ ] (const std::string &id)

Static Public Attributes

- static Logger **logger**

Friends

- class **ChainContext**

5.35.1 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.

5.35.2 Constructor & Destructor Documentation

5.35.2.1 Arc::Loader::Loader (Config * *cfg*)

Constructor that takes whole XML configuration and creates component chains

5.35.2.2 Arc::Loader::~Loader ()

Destructor destroys all components created by constructor

5.35.3 Member Function Documentation

5.35.3.1]

MCC* Arc::Loader::operator[ ] (const std::string & *id*)

Access entry MCCs in chains. Those are compnents exposed for external access using 'entry' attribute

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

- Loader.h

5.36 Arc::loader_descriptor Struct Reference

Identifier of plugin.

`#include <LoaderFactory.h>`

Public Attributes

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

5.36.1 Detailed Description

Identifier of plugin.

This structure describes set of elements stored in shared library. It contains name of plugin, version number and pointer to function which creates an instance of object.

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

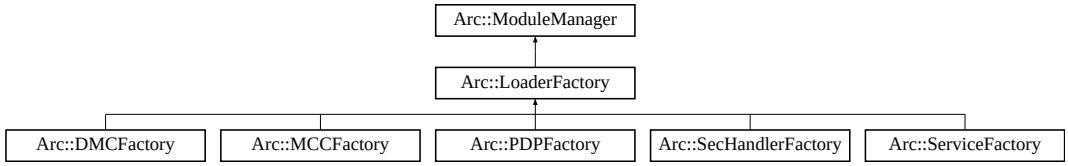
- LoaderFactory.h

5.37 Arc::LoaderFactory Class Reference

Plugin handler.

`#include <LoaderFactory.h>`

Inheritance diagram for Arc::LoaderFactory::



Public Member Functions

- void load_all_instances (const std::string &libname)

Protected Member Functions

- LoaderFactory (Config *cfg, const std::string &id)
- void * get_instance (const std::string &name, Arc::Config *cfg, Arc::ChainContext *ctx)
- void * **get_instance** (const std::string &name, int version, Arc::Config *cfg, Arc::ChainContext *ctx)
- void * **get_instance** (const std::string &name, int min_version, int max_version, Arc::Config *cfg, Arc::ChainContext *ctx)

5.37.1 Detailed Description

Plugin handler.

This class handles shared libraries containing loadable classes

5.37.2 Constructor & Destructor Documentation

5.37.2.1 Arc::LoaderFactory::LoaderFactory (Config * *cfg*, const std::string & *id*) [protected]

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

5.37.3 Member Function Documentation

5.37.3.1 void* Arc::LoaderFactory::get_instance (const std::string & *name*, Arc::Config * *cfg*, Arc::ChainContext * *ctx*) [protected]

These methods load shared library named lib'name', locates symbol named 'id_' representing descriptor of elements and calls it's constructor function. Supplied configuration tree and context are passed to constructor. Returns created instance. This classes must not be used directly. Inheriting classes must implement it with proper type casting.

Reimplemented in Arc::DMCFactory, Arc::MCCFactory, Arc::PDPFactory, Arc::SecHandlerFactory, and Arc::ServiceFactory.

5.37.3.2 void Arc::LoaderFactory::load_all_instances (const std::string & *libname*)

Loads shared library named 'libname' and identifies all elements it provides. Subsequent calls to get_-instance() methods will be able to locate needed elements even if they are not stored in library named after element name.

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

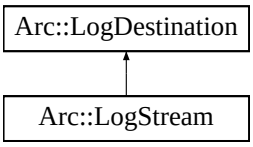
- LoaderFactory.h

5.38 Arc::LogDestination Class Reference

A base class for log destinations.

`#include <Logger.h>`

Inheritance diagram for Arc::LogDestination::



Public Member Functions

- virtual void log (const LogMessage &message)=0

Protected Member Functions

- LogDestination ()
- LogDestination (const std::string &locale)

Protected Attributes

- std::string **locale**

5.38.1 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.

5.38.2 Constructor & Destructor Documentation

5.38.2.1 Arc::LogDestination::LogDestination () `[protected]`

Default constructor.

This destination will use the default locale.

5.38.2.2 Arc::LogDestination::LogDestination (const std::string & *locale*) `[protected]`

Constructor with specific locale.

This destination will use the specified locale.

5.38.3 Member Function Documentation

5.38.3.1 virtual void Arc::LogDestination::log (const LogMessage & *message*) [pure virtual]

Logs a LogMessage to this LogDestination.

Implemented in Arc::LogStream.

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

- Logger.h

5.39 Arc::Logger Class Reference

A logger class.

`#include <Logger.h>`

Public Member Functions

- Logger (Logger &parent, const std::string &subdomain)
- Logger (Logger &parent, const std::string &subdomain, LogLevel threshold)
- void addDestination (LogDestination &destination)
- void removeDestinations (void)
- void setThreshold (LogLevel threshold)
- LogLevel getThreshold () const
- void msg (LogMessage message)
- void msg (LogLevel level, const std::string &str,...)

Static Public Member Functions

- static Logger & getRootLogger ()

5.39.1 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.

5.39.2 Constructor & Destructor Documentation

5.39.2.1 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.

5.39.2.2 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.

5.39.3 Member Function Documentation

5.39.3.1 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.

5.39.3.2 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.

5.39.3.3 LogLevel Arc::Logger::getThreshold () const

Returns the threshold.

Returns the threshold.

Returns:

The threshold of this Logger.

5.39.3.4 void Arc::Logger::msg (LogLevel *level*, const std::string & *str*, ...)

Loggs a message text.

Loggs 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.

5.39.3.5 void Arc::Logger::msg (LogMessage *message*)

Sends a LogMessage.

Sends a LogMessage.

Parameters:

The LogMessage to send.

5.39.3.6 void Arc::Logger::removeDestinations (void)

Removes all LogDestinations.

5.39.3.7 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

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

- Logger.h

5.40 Arc::LogMessage Class Reference

A class for log messages.

`#include <Logger.h>`

Public Member Functions

- LogMessage (LogLevel level, const std::string &message, va_list &v)
- LogMessage (LogLevel level, const std::string &message, const std::string &identifier, va_list &v)
- LogLevel getLevel () const

Protected Member Functions

- void setIdentifier (std::string identifier)

Friends

- class Logger
- std::ostream & operator<< (std::ostream &os, const LogMessage &message)

5.40.1 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.

5.40.2 Constructor & Destructor Documentation

5.40.2.1 Arc::LogMessage::LogMessage (LogLevel *level*, const std::string & *message*, va_list & *v*)

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.

5.40.2.2 Arc::LogMessage::LogMessage (LogLevel *level*, const std::string & *message*, const std::string & *identifier*, va_list & *v*)

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.

5.40.3 Member Function Documentation

5.40.3.1 LogLevel Arc::LogMessage::getLevel () const

Returns the level of the LogMessage.

Returns the level of the LogMessage.

Returns:

The level of the LogMessage.

5.40.3.2 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.

5.40.4 Friends And Related Function Documentation

5.40.4.1 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.

5.40.4.2 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:

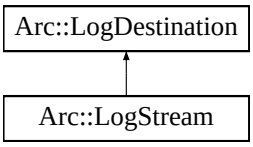
- Logger.h

5.41 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)

5.41.1 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.

5.41.2 Constructor & Destructor Documentation

5.41.2.1 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.

5.41.2.2 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.

5.41.3 Member Function Documentation

5.41.3.1 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:

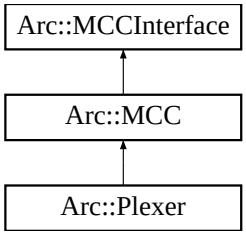
- Logger.h

5.42 Arc::MCC Class Reference

Message Chain Component - base class for every MCC plugin.

`#include <MCC.h>`

Inheritance diagram for Arc::MCC::



Public Member Functions

- MCC (Arc::Config *)
- virtual void Next (Arc::MCCInterface *next, const std::string &label="")
- virtual void AddSecHandler (Arc::Config *cfg, ArcSec::SecHandler *sechandler, const std::string &label="")
- virtual void Unlink (void)
- virtual Arc::MCC_Status process (Arc::Message &, Arc::Message &)

Protected Member Functions

- Arc::MCCInterface * **Next** (const std::string &label="")
- bool ProcessSecHandlers (Arc::Message &message, const std::string &label="")

Protected Attributes

- std::map< std::string, Arc::MCCInterface * > next_
- std::map< std::string, std::list< ArcSec::SecHandler * > > sechandlers_

Static Protected Attributes

- static Arc::Logger logger

5.42.1 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.

5.42.2 Constructor & Destructor Documentation

5.42.2.1 Arc::MCC::MCC (Arc::Config *) [inline]

Example contructor - MCC takes at least it's configuration subtree

5.42.3 Member Function Documentation

5.42.3.1 virtual void Arc::MCC::AddSecHandler (Arc::Config * *cfg*, ArcSec::SecHandler * *sechandler*, const std::string & *label* = "") [virtual]

Add security components/handlers to this MCC. Security handlers are stacked into few queues with each queue identified by it's label. Queue labeled 'incoming' is executed for every 'request' message after message is processes by MCC for service side and before processing on client side. Queue 'outgoing' is run for response message before it is processed by MCC algorithms on service side and after processing on client side. Those labels are just a matter of agreement and some MCCs may implement different queues executed at various message processing steps.

5.42.3.2 virtual void Arc::MCC::Next (Arc::MCCInterface * *next*, const std::string & *label* = "") [virtual]

Add reference to next MCC in chain. This method is called by Loader for every potentially labeled link to next component which implements MCCInterface. If next is NULL corresponding link is removed.

Reimplemented in Arc::Plexer.

5.42.3.3 virtual Arc::MCC_Status Arc::MCC::process (Arc::Message &, Arc::Message &) [inline, virtual]

Dummy Message processing method. Just a placeholder.

Implements Arc::MCCInterface.

Reimplemented in Arc::Plexer.

5.42.3.4 bool Arc::MCC::ProcessSecHandlers (Arc::Message & *message*, const std::string & *label* = "") [protected]

Executes security handlers of specified queue. Returns true if message is authorized for further processing or if there are no security handlers which implement authorization functionality. This is a convenience method and has to be called by implemention of MCC.

5.42.3.5 virtual void Arc::MCC::Unlink (void) [virtual]

Removing all links. Useful for destroying chains.

5.42.4 Member Data Documentation

5.42.4.1 Arc::Logger Arc::MCC::logger [static, protected]

A logger for MCCs.

A logger intended to be the parent of loggers in the different MCCs.

Reimplemented in Arc::Plexer.

5.42.4.2 std::map<std::string,Arc::MCCInterface*> Arc::MCC::next_ [protected]

Set of labeled "next" components. Each implemented MCC must call process() metthod of corresponding MCCInterface from this set in own process() method.

5.42.4.3 std::map<std::string,std::list<ArcSec::SecHandler*> > Arc::MCC::sechandlers_ [protected]

Set o flabeled authentication and authorization handlers. MCC calls sequence of handlers at specific point depending on associated identifier. In most aces those are "in" and "out" for incoming and outgoing messages correspondingly.

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

- MCC.h

5.43 mcc_descriptor Struct Reference

Identifier of Message Chain Componet (MCC) plugin.

`#include <MCCLoader.h>`

Public Attributes

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

5.43.1 Detailed Description

Identifier of Message Chain Componet (MCC) plugin.

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

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

- MCCLoader.h

5.44 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

5.44.1 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.

5.44.2 Constructor & Destructor Documentation

5.44.2.1 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.")

5.44.3 Member Function Documentation

5.44.3.1 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.

5.44.3.2 StatusKind Arc::MCC_Status::getKind () const

Returns the status kind.

Returns the status kind of this object.

Returns:

The status kind of this object.

5.44.3.3 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.

5.44.3.4 bool Arc::MCC_Status::isOk () const

Is the status kind ok?

This method returns true iff the status kind of this object is STATUS_OK

Returns:

true iff kind==STATUS_OK

5.44.3.5 Arc::MCC_Status::operator bool (void) const [inline]

Is the status kind ok?

This method returns true iff the status kind of this object is STATUS_OK

Returns:

true iff kind==STATUS_OK

5.44.3.6 Arc::MCC_Status::operator std::string () const

Conversion to string.

This operator converts a MCC_Status object to a string.

5.44.3.7 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:

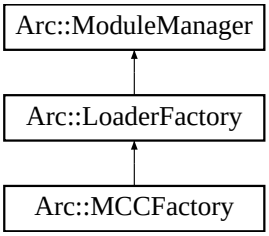
- MCC_Status.h

5.45 Arc::MCCFactory Class Reference

MCC Plugins handler.

`#include <MCCFactory.h>`

Inheritance diagram for Arc::MCCFactory::



Public Member Functions

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

5.45.1 Detailed Description

MCC Plugins handler.

This class handles shared libraries containing MCCs

5.45.2 Constructor & Destructor Documentation

5.45.2.1 Arc::MCCFactory::MCCFactory (Config * *cfg*)

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

5.45.3 Member Function Documentation

5.45.3.1 MCC* Arc::MCCFactory::get_instance (const std::string & *name*, Config * *cfg*, ChainContext * *ctx*)

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

Reimplemented from Arc::LoaderFactory.

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

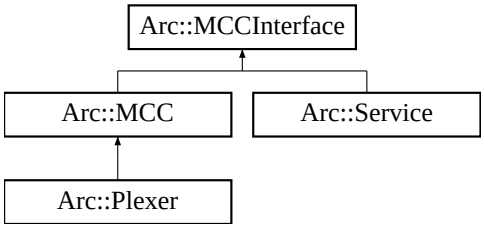
- MCCFactory.h

5.46 Arc::MCCInterface Class Reference

Interface for communication between MCC, Service and Plexer objects.

`#include <MCC.h>`

Inheritance diagram for Arc::MCCInterface::



Public Member Functions

- virtual Arc::MCC_Status process (Arc::Message &request, Arc::Message &response)=0

5.46.1 Detailed Description

Interface for communication between MCC, Service and Plexer objects.

The Interface is made of method process() which is called by previous MCC in chain. For memory management policies please read description of Message class.

5.46.2 Member Function Documentation

5.46.2.1 virtual Arc::MCC_Status Arc::MCCInterface::process (Arc::Message & *request*, Arc::Message & *response*) [pure virtual]

Method for processing of requests and responses. This method is called by preceeding MCC in chain when a request needs to be processed. This method must call similar method of next MCC in chain unless any failure happens. Result returned by call to next MCC should be processed and passed back to previous MCC. In case of failure this method is expected to generate valid error response and return it back to previous MCC without calling the next one.

Parameters:

request The request that needs to be processed.

response A Message object that will contain the response of the request when the method returns.

Returns:

An object representing the status of the call.

Implemented in Arc::Plexer, and Arc::MCC.

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

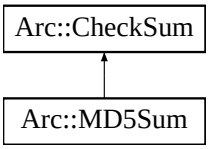
- MCC.h

5.47 Arc::MD5Sum Class Reference

Implementation of MD5 checksum.

`#include <CheckSum.h>`

Inheritance diagram for Arc::MD5Sum::



Public Member Functions

- virtual void **start** (void)
- virtual void **add** (void *buf, unsigned long long int len)
- virtual void **end** (void)
- virtual void **result** (unsigned char *&res, unsigned int &len) const
- virtual int **print** (char *buf, int len) const
- virtual void **scan** (const char *buf)
- virtual **operator bool** (void) const
- virtual bool **operator!** (void) const

5.47.1 Detailed Description

Implementation of MD5 checksum.

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

- CheckSum.h

5.48 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 *new_payload)
- MessageAttributes * Attributes (void)
- void **Attributes** (MessageAttributes *attributes)
- MessageAuth * Auth (void)
- void **Auth** (MessageAuth *auth)
- MessageContext * Context (void)
- void Context (MessageContext *context)

5.48.1 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.

5.48.2 Constructor & Destructor Documentation

5.48.2.1 Arc::Message::Message (void) [inline]

Dummy constructor

5.48.2.2 Arc::Message::Message (Message & *msg*) [inline]

Copy constructor. Ensures shallow copy.

5.48.2.3 Arc::Message::Message (long *msg_ptr_addr*)

Copy constructor. Used by language bindigs

5.48.2.4 Arc::Message::~Message (void) [inline]

Destructor does not affect refered objects

5.48.3 Member Function Documentation

5.48.3.1 MessageAttributes* Arc::Message::Attributes (void) [inline]

Returns a pointer to the current attributes object or NULL if no attributes object has been assigned.

5.48.3.2 MessageAuth* Arc::Message::Auth (void) [inline]

Returns a pointer to the current authentication/authorization object or NULL if no object has been assigned.

5.48.3.3 void Arc::Message::Context (MessageContext * *context*) [inline]

Assigns message context object

5.48.3.4 MessageContext* Arc::Message::Context (void) [inline]

Returns a pointer to the current context object or NULL if no object has been assigned. Last case can happen only if first MCC in a chain is connectionless like one implementing UDP protocol.

5.48.3.5 Message& Arc::Message::operator= (Message & *msg*) [inline]

Assignment. Ensures shallow copy.

5.48.3.6 MessagePayload* Arc::Message::Payload (MessagePayload * *new_payload*) [inline]

Replaces payload with new one. Returns the old one.

5.48.3.7 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

5.49 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_

5.49.1 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.

5.49.2 Constructor & Destructor Documentation

5.49.2.1 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.

5.49.3 Member Function Documentation

5.49.3.1 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.

5.49.3.2 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.

5.49.3.3 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.

5.49.3.4 AttributeIterator Arc::MessageAttributes::getAll (void) const

Access all value and attributes.

5.49.3.5 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.

5.49.3.6 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.

5.49.3.7 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.

5.49.3.8 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.

5.49.4 Member Data Documentation

5.49.4.1 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

5.50 Arc::MessageAuth Class Reference

Contains authencity information, authorization tokens and decisions.

`#include <MessageAuth.h>`

Public Member Functions

- void **set** (const std::string &key, const AuthObject &value)
- AuthObject **get** (const std::string &key, int index=0)
- void **remove** (const std::string &key)

5.50.1 Detailed Description

Contains authencity information, authorization tokens and decisions.

Functionality of this class is not defined yet.

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

- MessageAuth.h

5.51 Arc::MessageContext Class Reference

Handler for context of message context.

`#include <Message.h>`

Public Member Functions

- void Add (const std::string &name, MessageContextElement *element)
- MessageContextElement * **operator[]** (const std::string &id)

5.51.1 Detailed Description

Handler for context 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.

5.51.2 Member Function Documentation

5.51.2.1 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

5.52 Arc::MessageContextElement Class Reference

Top class for elements contained in message context.

`#include <Message.h>`

5.52.1 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:

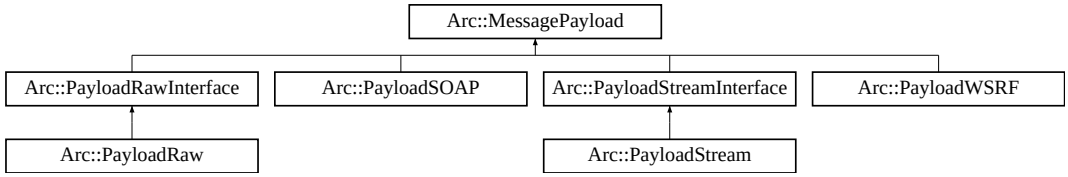
- Message.h

5.53 Arc::MessagePayload Class Reference

Base class for content of message passed through chain.

`#include <Message.h>`

Inheritance diagram for Arc::MessagePayload::



5.53.1 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:

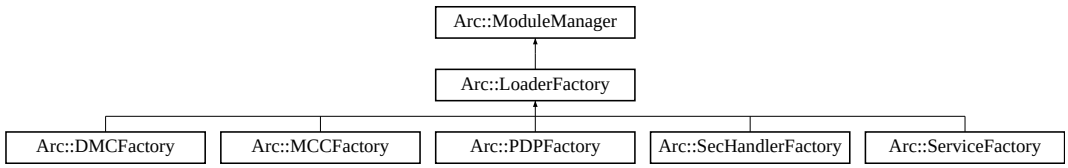
- Message.h

5.54 Arc::ModuleManager Class Reference

Manager of shared libraries.

`#include <ModuleManager.h>`

Inheritance diagram for Arc::ModuleManager::



Public Member Functions

- ModuleManager (Arc::Config *cfg)
- Glib::Module * load (const std::string &name)
- void setCfg (Arc::Config *cfg)

5.54.1 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.

5.54.2 Constructor & Destructor Documentation

5.54.2.1 Arc::ModuleManager::ModuleManager (Arc::Config * *cfg*)

Constructor. It is supposed to process correponding configuration subtree and tune module loading parameters accordingly. Currently it only sets modulr directory to current one.

5.54.3 Member Function Documentation

5.54.3.1 Glib::Module* Arc::ModuleManager::load (const std::string & *name*)

Finds module 'name' in cache or loads corresponding shared library

5.54.3.2 void Arc::ModuleManager::setCfg (Arc::Config * *cfg*)

Input the configuration subtree, and trigger the module loading (do almost the same as the Constructor); It is function desgined for ClassLoader to adopt the singleton pattern

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

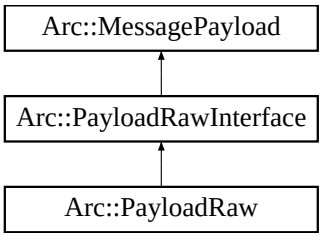
- ModuleManager.h

5.55 Arc::PayloadRaw Class Reference

Raw byte multi-buffer.

`#include <PayloadRaw.h>`

Inheritance diagram for Arc::PayloadRaw::



Public Member Functions

- PayloadRaw (void)
- virtual ~PayloadRaw (void)
- virtual char operator[] (int pos) const
- virtual char * Content (int pos=-1)
- virtual int Size (void) const
- virtual char * Insert (int pos=0, int size=0)
- virtual char * Insert (const char *s, int pos=0, int size=0)
- virtual char * Buffer (unsigned int num=0)
- virtual int BufferSize (unsigned int num=0) const
- virtual int BufferPos (unsigned int num=0) const
- virtual bool Truncate (unsigned int size)

Protected Attributes

- int **offset_**
- int **size_**
- std::vector< PayloadRawBuf > **buf_**

5.55.1 Detailed Description

Raw byte multi-buffer.

This is implementation of PayloadRawInterface. Buffers are memory blocks logically placed one after another.

5.55.2 Constructor & Destructor Documentation

5.55.2.1 Arc::PayloadRaw::PayloadRaw (void) [inline]

Constructor. Created object contains no buffers.

5.55.2.2 virtual Arc::PayloadRaw::~PayloadRaw (void) [virtual]

Destructor. Frees allocated buffers.

5.55.3 Member Function Documentation

5.55.3.1 virtual char* Arc::PayloadRaw::Buffer (unsigned int *num* = 0) [virtual]

Returns pointer to num'th buffer

Implements Arc::PayloadRawInterface.

5.55.3.2 virtual int Arc::PayloadRaw::BufferPos (unsigned int *num* = 0) const [virtual]

Returns position of num'th buffer

Implements Arc::PayloadRawInterface.

5.55.3.3 virtual int Arc::PayloadRaw::BufferSize (unsigned int *num* = 0) const [virtual]

Returns length of num'th buffer

Implements Arc::PayloadRawInterface.

5.55.3.4 virtual char* Arc::PayloadRaw::Content (int *pos* = -1) [virtual]

Get pointer to buffer content at global position 'pos'. By default to beginning of main buffer whatever that means.

Implements Arc::PayloadRawInterface.

5.55.3.5 virtual char* Arc::PayloadRaw::Insert (const char * *s*, int *pos* = 0, int *size* = 0) [virtual]

Create new buffer at global position 'pos' of size 'size'. Created buffer is filled with content of memory at 's'. If 'size' is 0 content at 's' is expected to be null-terminated.

Implements Arc::PayloadRawInterface.

5.55.3.6 virtual char* Arc::PayloadRaw::Insert (int *pos* = 0, int *size* = 0) [virtual]

Create new buffer at global position 'pos' of size 'size'.

Implements Arc::PayloadRawInterface.

5.55.3.7]

virtual char Arc::PayloadRaw::operator[ ] (int *pos*) const [virtual]

Returns content of byte at specified position. Specified position 'pos' is treated as global one and goes through all buffers placed one after another.

Implements Arc::PayloadRawInterface.

5.55.3.8 virtual int Arc::PayloadRaw::Size (void) const [virtual]

Returns logical size of whole structure.

Implements Arc::PayloadRawInterface.

5.55.3.9 virtual bool Arc::PayloadRaw::Truncate (unsigned int *size*) [virtual]

Change size of stored information. If size exceeds end of allocated buffer, buffers are not re-allocated, only logical size is extended. Buffers with location behind new size are deallocated.

Implements Arc::PayloadRawInterface.

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

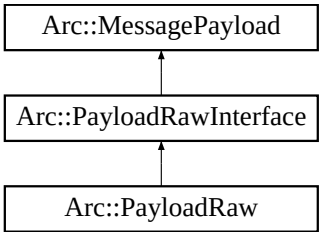
- PayloadRaw.h

5.56 Arc::PayloadRawInterface Class Reference

Random Access Payload for Message objects.

`#include <PayloadRaw.h>`

Inheritance diagram for Arc::PayloadRawInterface::



Public Member Functions

- virtual char operator[ ] (int pos) const =0
- virtual char * Content (int pos=-1)=0
- virtual int Size (void) const =0
- virtual char * Insert (int pos=0, int size=0)=0
- virtual char * Insert (const char *s, int pos=0, int size=0)=0
- virtual char * Buffer (unsigned int num)=0
- virtual int BufferSize (unsigned int num) const =0
- virtual int BufferPos (unsigned int num) const =0
- virtual bool Truncate (unsigned int size)=0

5.56.1 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.

5.56.2 Member Function Documentation

5.56.2.1 virtual char* Arc::PayloadRawInterface::Buffer (unsigned int *num*) [pure virtual]

Returns pointer to num'th buffer

Implemented in Arc::PayloadRaw.

5.56.2.2 virtual int Arc::PayloadRawInterface::BufferPos (unsigned int *num*) const [pure virtual]

Returns position of num'th buffer

Implemented in Arc::PayloadRaw.

5.56.2.3 virtual int Arc::PayloadRawInterface::BufferSize (unsigned int *num*) const [pure virtual]

Returns length of num'th buffer

Implemented in Arc::PayloadRaw.

5.56.2.4 virtual char* Arc::PayloadRawInterface::Content (int *pos* = -1) [pure virtual]

Get pointer to buffer content at global position 'pos'. By default to beginning of main buffer whatever that means.

Implemented in Arc::PayloadRaw.

5.56.2.5 virtual char* Arc::PayloadRawInterface::Insert (const char * *s*, int *pos* = 0, int *size* = 0) [pure virtual]

Create new buffer at global position 'pos' of size 'size'. Created buffer is filled with content of memory at 's'. If 'size' is 0 content at 's' is expected to be null-terminated.

Implemented in Arc::PayloadRaw.

5.56.2.6 virtual char* Arc::PayloadRawInterface::Insert (int *pos* = 0, int *size* = 0) [pure virtual]

Create new buffer at global position 'pos' of size 'size'.

Implemented in Arc::PayloadRaw.

5.56.2.7]

virtual char Arc::PayloadRawInterface::operator[ ] (int *pos*) const [pure virtual]

Returns content of byte at specified position. Specified position 'pos' is treated as global one and goes through all buffers placed one after another.

Implemented in Arc::PayloadRaw.

5.56.2.8 virtual int Arc::PayloadRawInterface::Size (void) const [pure virtual]

Returns logical size of whole structure.

Implemented in Arc::PayloadRaw.

5.56.2.9 virtual bool Arc::PayloadRawInterface::Truncate (unsigned int *size*) [pure virtual]

Change size of stored information. If size exceeds end of allocated buffer, buffers are not re-allocated, only logical size is extended. Buffers with location behind new size are deallocated.

Implemented in Arc::PayloadRaw.

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

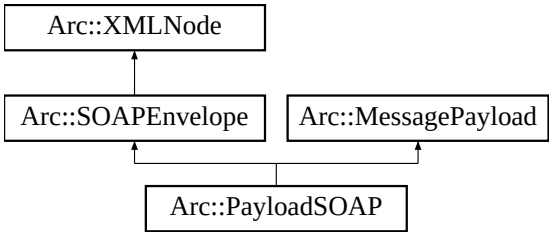
- PayloadRaw.h

5.57 Arc::PayloadSOAP Class Reference

Payload of Message with SOAP content.

`#include <PayloadSOAP.h>`

Inheritance diagram for Arc::PayloadSOAP::



Public Member Functions

- PayloadSOAP (const Arc::NS &ns, bool fault=false)
- PayloadSOAP (const Arc::SOAPEnvelope &soap)
- PayloadSOAP (const Arc::MessagePayload &source)

5.57.1 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.

5.57.2 Constructor & Destructor Documentation

5.57.2.1 Arc::PayloadSOAP::PayloadSOAP (const Arc::NS & *ns*, bool *fault* = `false`)

Constructor - creates new Message payload

5.57.2.2 Arc::PayloadSOAP::PayloadSOAP (const Arc::SOAPEnvelope & *soap*)

Constructor - creates Message payload from SOAP document. Provided SOAP document must exist as long as created object exists.

5.57.2.3 Arc::PayloadSOAP::PayloadSOAP (const Arc::MessagePayload & *source*)

Constructor - creates SOAP message from payload. PayloadRawInterface and derived classes are supported.

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

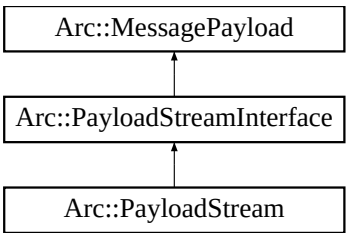
- PayloadSOAP.h

5.58 Arc::PayloadStream Class Reference

POSIX handle as Payload.

`#include <PayloadStream.h>`

Inheritance diagram for Arc::PayloadStream::



Public Member Functions

- PayloadStream (int h=-1)
- virtual ~PayloadStream (void)
- virtual bool Get (char *buf, int &size)
- virtual bool Get (std::string &buf)
- virtual std::string Get (void)
- virtual bool Put (const char *buf, int size)
- virtual bool Put (const std::string &buf)
- virtual bool Put (const char *buf)
- virtual operator bool (void)
- virtual bool operator! (void)
- virtual int Timeout (void) const
- virtual void Timeout (int to)
- virtual int GetHandle (void)

Protected Attributes

- int **timeout_**
- int handle_
- bool seekable_

5.58.1 Detailed Description

POSIX handle as Payload.

Thsi is an implemetation of PayloadStreamInterface for generic POSIX handle.

5.58.2 Constructor & Destructor Documentation

5.58.2.1 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.

5.58.2.2 virtual Arc::PayloadStream::~PayloadStream (void) [inline, virtual]

Destructor.

5.58.3 Member Function Documentation

5.58.3.1 virtual std::string Arc::PayloadStream::Get (void) [inline, virtual]

Read as many as possible (sane amount) of bytes.

Implements Arc::PayloadStreamInterface.

5.58.3.2 virtual bool Arc::PayloadStream::Get (std::string & *buf*) [virtual]

Read as many as possible (sane amount) of bytes into buf.

Implements Arc::PayloadStreamInterface.

5.58.3.3 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.

5.58.3.4 virtual int Arc::PayloadStream::GetHandle (void) [inline, virtual]

Returns POSIX handle of the stream. This method is deprecated and will be removed soon. Currently it is only used by Transport Layer Security MCC.

5.58.3.5 virtual Arc::PayloadStream::operator bool (void) [inline, virtual]

Returns true if stream is valid.

Implements Arc::PayloadStreamInterface.

5.58.3.6 virtual bool Arc::PayloadStream::operator! (void) [inline, virtual]

Returns true if stream is invalid.

Implements Arc::PayloadStreamInterface.

5.58.3.7 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.

5.58.3.8 virtual bool Arc::PayloadStream::Put (const std::string & *buf*) [inline, virtual]

Push information from 'buf' into stream. Returns true on success.

Implements Arc::PayloadStreamInterface.

5.58.3.9 virtual bool Arc::PayloadStream::Put (const char * *buf*, int *size*) [virtual]

Push 'size' bytes from 'buf' into stream. Returns true on success.

Implements Arc::PayloadStreamInterface.

5.58.3.10 virtual void Arc::PayloadStream::Timeout (int *to*) [inline, virtual]

Set current timeout for Get() and Put() operations.

Implements Arc::PayloadStreamInterface.

5.58.3.11 virtual int Arc::PayloadStream::Timeout (void) const [inline, virtual]

Query current timeout for Get() and Put() operations.

Implements Arc::PayloadStreamInterface.

5.58.4 Member Data Documentation

5.58.4.1 int Arc::PayloadStream::handle_ [protected]

Timeout for read/write operations

5.58.4.2 bool Arc::PayloadStream::seekable_ [protected]

Handle for operations

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

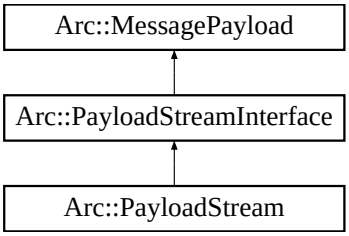
- PayloadStream.h

5.59 Arc::PayloadStreamInterface Class Reference

Stream-like Payload for Message object.

`#include <PayloadStream.h>`

Inheritance diagram for Arc::PayloadStreamInterface::



Public Member Functions

- virtual bool Get (char *buf, int &size)=0
- virtual bool Get (std::string &buf)=0
- virtual std::string Get (void)=0
- virtual bool Put (const char *buf, int size)=0
- virtual bool Put (const std::string &buf)=0
- virtual bool Put (const char *buf)=0
- virtual operator bool (void)=0
- virtual bool operator! (void)=0
- virtual int Timeout (void) const =0
- virtual void Timeout (int to)=0

5.59.1 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.

5.59.2 Member Function Documentation

5.59.2.1 virtual std::string Arc::PayloadStreamInterface::Get (void) [pure virtual]

Read as many as possible (sane amount) of bytes.

Implemented in Arc::PayloadStream.

5.59.2.2 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.

5.59.2.3 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.

5.59.2.4 virtual Arc::PayloadStreamInterface::operator bool (void) [pure virtual]

Returns true if stream is valid.

Implemented in Arc::PayloadStream.

5.59.2.5 virtual bool Arc::PayloadStreamInterface::operator! (void) [pure virtual]

Returns true if stream is invalid.

Implemented in Arc::PayloadStream.

5.59.2.6 virtual bool Arc::PayloadStreamInterface::Put (const char * *buf*) [pure virtual]

Push null terminated information from 'buf' into stream. Returns true on success.

Implemented in Arc::PayloadStream.

5.59.2.7 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.

5.59.2.8 virtual bool Arc::PayloadStreamInterface::Put (const char * *buf*, int *size*) [pure virtual]

Push 'size' bytes from 'buf' into stream. Returns true on success.

Implemented in Arc::PayloadStream.

5.59.2.9 virtual void Arc::PayloadStreamInterface::Timeout (int *to*) [pure virtual]

Set current timeout for Get() and Put() operations.

Implemented in Arc::PayloadStream.

5.59.2.10 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:

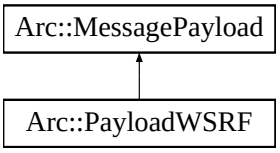
- PayloadStream.h

5.60 Arc::PayloadWSRF Class Reference

This class combines MessagePayload with WSRF.

`#include <PayloadWSRF.h>`

Inheritance diagram for Arc::PayloadWSRF::



Public Member Functions

- PayloadWSRF (const SOAPEnvelope &soap)
- PayloadWSRF (WSRF &wsrp)
- PayloadWSRF (const MessagePayload &source)
- **operator WSRF &** (void)
- **operator bool** (void)

Protected Attributes

- WSRF & **wsrf_**
- bool **owner_**

5.60.1 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.

5.60.2 Constructor & Destructor Documentation

5.60.2.1 Arc::PayloadWSRF::PayloadWSRF (const SOAPEnvelope & *soap*)

Constructor - creates Message payload from SOAP message. Returns invalid WSRF if SOAP does not represent WS-ResourceProperties

5.60.2.2 Arc::PayloadWSRF::PayloadWSRF (WSRF & *wsrp*)

Constructor - creates Message payload with acquired WSRF message. WSRF message will be destroyed by destructor of this object.

5.60.2.3 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

5.61 pdp_descriptor Struct Reference

Identifier of Policy Decision Point (PDP) plugin.

`#include <PDPLoader.h>`

Public Attributes

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

5.61.1 Detailed Description

Identifier of Policy Decision Point (PDP) plugin.

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

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

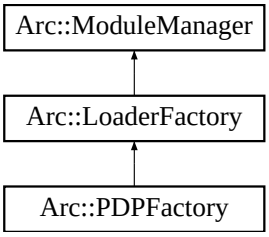
- PDPLoader.h

5.62 Arc::PDPFactory Class Reference

PDP Plugins handler.

`#include <PDPFactory.h>`

Inheritance diagram for Arc::PDPFactory::



Public Member Functions

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

5.62.1 Detailed Description

PDP Plugins handler.

This class handles shared libraries containing PDPs

5.62.2 Constructor & Destructor Documentation

5.62.2.1 Arc::PDPFactory::PDPFactory (Config * *cfg*)

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

5.62.3 Member Function Documentation

5.62.3.1 ArcSec::PDP* Arc::PDPFactory::get_instance (const std::string & *name*, Config * *cfg*, ChainContext * *ctx*)

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

Reimplemented from Arc::LoaderFactory.

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

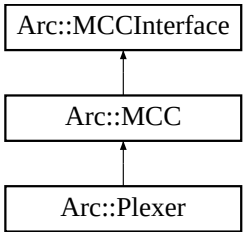
- PDPFactory.h

5.63 Arc::Plexer Class Reference

The Plexer class, used for routing messages to services.

`#include <Plexer.h>`

Inheritance diagram for Arc::Plexer::



Public Member Functions

- Plexer (Config *cfg)
- virtual ~Plexer ()
- virtual void Next (MCCInterface *next, const std::string &label)
- virtual MCC_Status process (Message &request, Message &response)

Static Public Attributes

- static Arc::Logger logger

5.63.1 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.

5.63.2 Constructor & Destructor Documentation

5.63.2.1 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.

5.63.2.2 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.

5.63.3 Member Function Documentation

5.63.3.1 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.

5.63.3.2 virtual MCC_Status Arc::Plexer::process (Message & *request*, Message & *response*) [virtual]

Rout request messages to appropriate services.

Routs the request message to the appropriate service. Currently routing is based on the value of the "Request-URI" attribute, but that may be replaced by some other attribute once the attributes discussion is finished.

Reimplemented from Arc::MCC.

5.63.4 Member Data Documentation

5.63.4.1 Arc::Logger Arc::Plexer::logger [static]

A logger for MCCs.

A logger intended to be the parent of loggers in the different MCCs.

Reimplemented from Arc::MCC.

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

- Plexer.h

5.64 Arc::PlexerEntry Class Reference

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

`#include <Plexer.h>`

Friends

- class **Plexer**

5.64.1 Detailed Description

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

A helper class that stores a label (regex) and a pointer to a service.

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

- Plexer.h

5.65 Arc::RegularExpression Class Reference

A regular expression class.

`#include <ArcRegex.h>`

Public Member Functions

- RegularExpression (std::string pattern)
- RegularExpression (const RegularExpression &regex)
- ~RegularExpression ()
- const RegularExpression & operator= (const RegularExpression &regex)
- bool isOk ()
- bool hasPattern (std::string str)
- bool match (const std::string &str) const
- bool match (const std::string &str, std::list< std::string > &unmatched, std::list< std::string > &matched) const
- std::string getPattern ()

5.65.1 Detailed Description

A regular expression class.

This class is a wrapper around the functions provided in regex.h.

5.65.2 Constructor & Destructor Documentation

5.65.2.1 Arc::RegularExpression::RegularExpression (std::string *pattern*)

Creates a reges from a pattern string.

5.65.2.2 Arc::RegularExpression::RegularExpression (const RegularExpression & *regex*)

Copy constructor.

5.65.2.3 Arc::RegularExpression::~RegularExpression ()

Destructor.

5.65.3 Member Function Documentation

5.65.3.1 std::string Arc::RegularExpression::getPattern ()

Returns patter.

5.65.3.2 bool Arc::RegularExpression::hasPattern (std::string *str*)

Returns true if this regex has the pattern provided.

5.65.3.3 bool Arc::RegularExpression::isOk ()

Returns true if the pattern of this regex is ok.

5.65.3.4 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 'macthed'.

5.65.3.5 bool Arc::RegularExpression::match (const std::string & *str*) const

Returns true if this regex matches whole string provided.

5.65.3.6 const RegularExpression& Arc::RegularExpression::operator= (const RegularExpression & *regex*)

Assignment operator.

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

- ArcRegex.h

5.66 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)
- int Result (void)
- bool Running (void)
- int ReadStdout (int timeout, char *buf, int size)
- int ReadStderr (int timeout, char *buf, int size)
- int WriteStdin (int timeout, const char *buf, int size)
- void AssignStdout (std::string &str)
- void AssignStderr (std::string &str)
- void AssignStdin (std::string &str)
- void KeepStdout (bool keep=true)
- void KeepStderr (bool keep=true)
- void KeepStdin (bool keep=true)
- void CloseStdout (void)
- void CloseStderr (void)
- void CloseStdin (void)
- void **AssignInitializer** (void(*initializer_func)(void *), void *initializer_arg)
- void Kill (int timeout)

Protected Member Functions

- bool **stdout_handler** (Glib::IOCondition cond)
- bool **stderr_handler** (Glib::IOCondition cond)
- bool **stdin_handler** (Glib::IOCondition cond)
- void **child_handler** (Glib::Pid pid, int result)

Protected Attributes

- int **stdout_**
- int **stderr_**
- int **stdin_**
- std::string * **stdout_str_**
- std::string * **stderr_str_**
- std::string * **stdin_str_**
- bool **stdout_keep_**
- bool **stderr_keep_**
- bool **stdin_keep_**
- sigc::connection **stdout_conn_**

- sigc::connection **stderr_conn_**
- sigc::connection **stdin_conn_**
- sigc::connection **child_conn_**
- Glib::Pid **pid_**
- Glib::ArrayHandle< std::string > **argv_**
- void(* **initializer_func_**)(void *)
- void * **initializer_arg_**
- bool **started_**
- bool **running_**
- int **result_**
- Glib::Mutex **lock_**
- Glib::Cond **cond_**

Friends

- class **RunPump**

5.66.1 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.

5.66.2 Constructor & Destructor Documentation

5.66.2.1 Arc::Run::Run (const std::string & *cmdline*)

Constructor preapres object to run cmdline

5.66.2.2 Arc::Run::Run (const std::list< std::string > & *argv*)

Constructor preapres object to run executable and arguments specified in argv

5.66.2.3 Arc::Run::~Run (void)

Destructor kill running executable and releases associated resources

5.66.3 Member Function Documentation

5.66.3.1 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.

5.66.3.2 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.

5.66.3.3 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.

5.66.3.4 void Arc::Run::CloseStderr (void)

Closes pipe associated with stderr handle

5.66.3.5 void Arc::Run::CloseStdin (void)

Closes pipe associated with stdin handle

5.66.3.6 void Arc::Run::CloseStdout (void)

Closes pipe associated with stdout handle

5.66.3.7 void Arc::Run::KeepStderr (bool *keep* = `true`)

Keep stderr same as parent's if keep = true

5.66.3.8 void Arc::Run::KeepStdin (bool *keep* = `true`)

Keep stdin same as parent's if keep = true

5.66.3.9 void Arc::Run::KeepStdout (bool *keep* = `true`)

Keep stdout same as parent's if keep = true

5.66.3.10 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

5.66.3.11 Arc::Run::operator bool (void) `[inline]`

Returns true if object is valid

5.66.3.12 bool Arc::Run::operator! (void) `[inline]`

Returns true if object is invalid

5.66.3.13 int Arc::Run::ReadStderr (int *timeout*, char * *buf*, int *size*)

Read from stderr handle of running executable. This method may be used while stderr is directed to string. But result is unpredictable.

5.66.3.14 int Arc::Run::ReadStdout (int *timeout*, char * *buf*, int *size*)

Read from stdout handle of running executable. This method may be used while stdout is directed to string. But result is unpredictable.

5.66.3.15 int Arc::Run::Result (void) [inline]

Returns exit code of execution.

5.66.3.16 bool Arc::Run::Running (void) [inline]

Return true if execution is going on.

5.66.3.17 bool Arc::Run::Start (void)

Starts running executable. This method may be called only once.

5.66.3.18 bool Arc::Run::Wait (int *timeout*)

Wait till execution finished or till timeout seconds expires. Returns true if execution is complete.

5.66.3.19 int Arc::Run::WriteStdin (int *timeout*, const char * *buf*, int *size*)

Write to stdin handle of running executable. This method may be used while stdin is directed to string. But result is unpredictable.

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

- Run.h

5.67 sechandler_descriptor Struct Reference

Identifier of SecHandler plugin.

`#include <SecHandlerLoader.h>`

Public Attributes

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

5.67.1 Detailed Description

Identifier of SecHandler plugin.

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

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

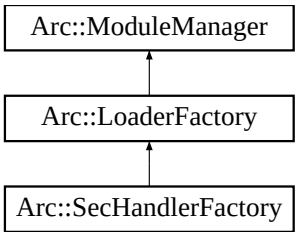
- SecHandlerLoader.h

5.68 Arc::SecHandlerFactory Class Reference

SecHandler Plugins handler.

`#include <SecHandlerFactory.h>`

Inheritance diagram for Arc::SecHandlerFactory::



Public Member Functions

- SecHandlerFactory (Config ∗cfg)
- ArcSec::SecHandler ∗ get_instance (const std::string &name, Config ∗cfg, ChainContext ∗ctx)
- ArcSec::SecHandler ∗ **get_instance** (const std::string &name, int version, Config ∗cfg, ChainContext ∗ctx)
- ArcSec::SecHandler ∗ **get_instance** (const std::string &name, int min_version, int max_version, Config ∗cfg, ChainContext ∗ctx)

5.68.1 Detailed Description

SecHandler Plugins handler.

This class handles shared libraries containing SecHandlers

5.68.2 Constructor & Destructor Documentation

5.68.2.1 Arc::SecHandlerFactory::SecHandlerFactory (Config ∗ *cfg*)

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

5.68.3 Member Function Documentation

5.68.3.1 ArcSec::SecHandler∗ Arc::SecHandlerFactory::get_instance (const std::string & *name*, Config ∗ *cfg*, ChainContext ∗ *ctx*)

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

Reimplemented from Arc::LoaderFactory.

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

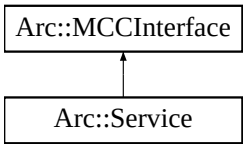
- SecHandlerFactory.h

5.69 Arc::Service Class Reference

Service - last component in a Message Chain.

`#include <Service.h>`

Inheritance diagram for Arc::Service::



Public Member Functions

- Service (Arc::Config *)
- virtual void AddSecHandler (Arc::Config *cfg, ArcSec::SecHandler *sechandler, const std::string &label="")

Protected Member Functions

- bool ProcessSecHandlers (Arc::Message &message, const std::string &label="")

Protected Attributes

- std::map< std::string, std::list< ArcSec::SecHandler * > > sechandlers_

Static Protected Attributes

- static Logger **logger**

5.69.1 Detailed Description

Service - last component in a Message Chain.

This is virtual class which defines interface (in a future also common functionality) for every Service plugin. Interface is made of method process() which is called by Plexer or MCC class. There is one Service object created for every service description processed by Loader class objects. Classes derived from Service class must implement process() method of MCCInterface. It is up to developer how internal state of service is stored and communicated to other services and external utilites. Service is free to expect any type of payload passed to it and generate any payload as well. Useful types depend on MCCs in chain which leads to that service. For example if service is expected to by linked to SOAP MCC it must accept and generate messages with PayloadSOAP payload. Method process() of class derived from Service class may be called concurently in multiple threads. Developers must take that into account and write thread-safe implementation. Simple example of service is provided in /src/tests/echo/echo.cpp of source tree. The way to write client couterpart of corresponding service is undefined yet. For example see /src/tests/echo/test.cpp .

5.69.2 Constructor & Destructor Documentation

5.69.2.1 Arc::Service::Service (Arc::Config *) [inline]

Example contructor - Server takes at least it's configuration subtree

5.69.3 Member Function Documentation

5.69.3.1 virtual void Arc::Service::AddSecHandler (Arc::Config * *cfg*, ArcSec::SecHandler * *sechandler*, const std::string & *label* = "") [virtual]

Add security components/handlers to this MCC. For more information please see description of MCC::AddSecHandler

5.69.3.2 bool Arc::Service::ProcessSecHandlers (Arc::Message & *message*, const std::string & *label* = "") [protected]

Executes security handlers of specified queue. For more information please see description of MCC::ProcessSecHandlers

5.69.4 Member Data Documentation

5.69.4.1 std::map<std::string,std::list<ArcSec::SecHandler*> > Arc::Service::sechandlers_ [protected]

Set of labeled authentication and authorization handlers. MCC calls sequence of handlers at specific point depending on associated identifier. in most aces those are "in" and "out" for incoming and outgoing messages correspondingly.

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

- Service.h

5.70 service_descriptor Struct Reference

Identifier of Service plugin.

`#include <ServiceLoader.h>`

Public Attributes

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

5.70.1 Detailed Description

Identifier of Service plugin.

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

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

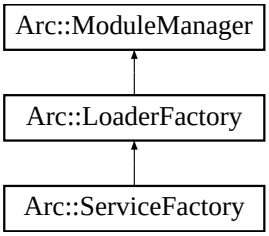
- ServiceLoader.h

5.71 Arc::ServiceFactory Class Reference

Service Plugins handler.

`#include <ServiceFactory.h>`

Inheritance diagram for Arc::ServiceFactory::



Public Member Functions

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

5.71.1 Detailed Description

Service Plugins handler.

This class handles shared libraries containing Services

5.71.2 Constructor & Destructor Documentation

5.71.2.1 Arc::ServiceFactory::ServiceFactory (Config * *cfg*)

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

5.71.3 Member Function Documentation

5.71.3.1 Service* Arc::ServiceFactory::get_instance (const std::string & *name*, Config * *cfg*, ChainContext * *ctx*)

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

Reimplemented from Arc::LoaderFactory.

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

- ServiceFactory.h

5.72 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)

5.72.1 Detailed Description

Simple triggered condition.

Provides condition and semaphor objects in one element.

5.72.2 Member Function Documentation

5.72.2.1 void Arc::SimpleCondition::broadcast (void) [inline]

Signal about condition to all waiting threads

5.72.2.2 void Arc::SimpleCondition::lock (void) [inline]

Acquire semaphor

5.72.2.3 void Arc::SimpleCondition::reset (void) [inline]

Reset object to initial state

5.72.2.4 void Arc::SimpleCondition::signal (void) [inline]

Signal about condition

5.72.2.5 void Arc::SimpleCondition::signal_nonblock (void) [inline]

Signal about condition without using semaphor

5.72.2.6 void Arc::SimpleCondition::unlock (void) [inline]

Release semaphor

5.72.2.7 bool Arc::SimpleCondition::wait (int *t*) [inline]

Wait for condition no longer than t milliseconds

5.72.2.8 void Arc::SimpleCondition::wait (void) [inline]

Wait for condition

5.72.2.9 void Arc::SimpleCondition::wait_nonblock (void) [inline]

Wait for condition without using semaphor

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

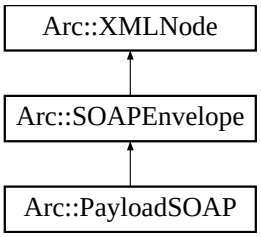
- Thread.h

5.73 Arc::SOAPEnvelope Class Reference

Extends XMLNode class to support structures of SOAP message.

`#include <SOAPEnvelope.h>`

Inheritance diagram for Arc::SOAPEnvelope::



Public Types

- **Version_1_1**
- **Version_1_2**
- enum **SOAPVersion** { **Version_1_1**, **Version_1_2** }

Public Member Functions

- SOAPEnvelope (const std::string &xml)
- SOAPEnvelope (const char *xml, int len=-1)
- SOAPEnvelope (const NS &ns, bool fault=false)
- SOAPEnvelope (XMLNode root)
- SOAPEnvelope (const SOAPEnvelope &soap)
- SOAPEnvelope * New (void)
- void Namespaces (const NS &namespaces)
- void GetXML (std::string &xml) const
- XMLNode Header (void)
- bool IsFault (void)
- SOAPFault * Fault (void)
- SOAPEnvelope & operator= (const SOAPEnvelope &soap)
- SOAPVersion **Version** (void)

5.73.1 Detailed Description

Extends XMLNode class to support structures of SOAP message.

All XMLNode methods are exposed by inheriting from XMLNode and node itself is translated into Envelope part of SOAP.

5.73.2 Constructor & Destructor Documentation

5.73.2.1 Arc::SOAPEnvelope::SOAPEnvelope (const std::string & *xml*)

Create new SOAP message from textual representation of XML document. Created XML structure is owned by this instance. This constructor also sets default namespaces to default prefixes as specified below.

5.73.2.2 Arc::SOAPEnvelope::SOAPEnvelope (const char * *xml*, int *len* = -1)

Same as previous

5.73.2.3 Arc::SOAPEnvelope::SOAPEnvelope (const NS & *ns*, bool *fault* = false)

Create new SOAP message with specified namespaces. Created XML structure is owned by this instance. If argument fault is set to true created message is fault.

5.73.2.4 Arc::SOAPEnvelope::SOAPEnvelope (XMLNode *root*)

Acquire XML document as SOAP message. Created XML structure is NOT owned by this instance.

5.73.2.5 Arc::SOAPEnvelope::SOAPEnvelope (const SOAPEnvelope & *soap*)

Create a copy of another SOAPEnvelope object.

5.73.3 Member Function Documentation

5.73.3.1 SOAPFault* Arc::SOAPEnvelope::Fault (void) [inline]

Get Fault part of message. Returns NULL if message is not Fault.

5.73.3.2 void Arc::SOAPEnvelope::GetXML (std::string & *xml*) const

Fills argument with this instance XML subtree textual representation

Reimplemented from Arc::XMLNode.

5.73.3.3 XMLNode Arc::SOAPEnvelope::Header (void) [inline]

Get SOAP header as XML node

5.73.3.4 bool Arc::SOAPEnvelope::IsFault (void) [inline]

Returns true if message is Fault

5.73.3.5 void Arc::SOAPEnvelope::Namespaces (const NS & *namespaces*)

Modify assigned namespaces. Default namespaces and prefixes are soap-enc http://schemas.xmlsoap.org/soap/encoding/ soap-env http://schemas.xmlsoap.org/soap/envelope/ xsi http://www.w3.org/2001/XMLSchema-instance xsd http://www.w3.org/2001/XMLSchema

Reimplemented from Arc::XMLNode.

5.73.3.6 SOAPEnvelope* Arc::SOAPEnvelope::New (void)

Creates complete copy of SOAP. Do not use New() method of XMLNode - use this one.

5.73.3.7 SOAPEnvelope& Arc::SOAPEnvelope::operator= (const SOAPEnvelope & *soap*)

Makes this object a copy of another SOAPEnvelope object.

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

- SOAPEnvelope.h

5.74 Arc::SOAPFault Class Reference

Interface to SOAP Fault message.

`#include <SOAPEnvelope.h>`

Public Types

- **undefined**
- **unknown**
- **VersionMismatch**
- **MustUnderstand**
- **Sender**
- **Receiver**
- **DataEncodingUnknown**
- enum SOAPFaultCode {

  undefined, **unknown**, **VersionMismatch**, **MustUnderstand**,

  Sender, **Receiver**, **DataEncodingUnknown** }

Public Member Functions

- SOAPFault (XMLNode &body)
- operator bool (void)
- SOAPFaultCode Code (void)
- void Code (SOAPFaultCode code)
- std::string Subcode (int level)
- void Subcode (int level, const char *subcode)
- std::string Reason (int num=0)
- void Reason (int num, const char *reason)
- void Reason (const char *reason)
- std::string Node (void)
- void Node (const char *node)
- std::string Role (void)
- void Role (const char *role)
- XMLNode Detail (bool create=false)

Friends

- class **SOAPEnvelope**

5.74.1 Detailed Description

Interface to SOAP Fault message.

SOAPFault class provides a convenience interface for accessing elements of SOAP faults. It also tries to expose single interface for both version 1.0 and 1.2 faults. This class is not intended to 'own' any information stored. It's purpose is to manipulate information which is kept under control of XMLNode or SOAPEnvelope classes. If instance does not refer to valid SOAP Fault structure all manipulation methods will have no effect.

5.74.2 Member Enumeration Documentation

5.74.2.1 enum Arc::SOAPFault::SOAPFaultCode

Fault codes of SOAP specs

5.74.3 Constructor & Destructor Documentation

5.74.3.1 Arc::SOAPFault::SOAPFault (XMLNode & *body*)

Parse Fault elements of SOAP Body or any other XML tree with Fault element

5.74.4 Member Function Documentation

5.74.4.1 void Arc::SOAPFault::Code (SOAPFaultCode *code*)

Set Fault Code element

5.74.4.2 SOAPFaultCode Arc::SOAPFault::Code (void)

Returns Fault Code element

5.74.4.3 XMLNode Arc::SOAPFault::Detail (bool *create* = `false`)

Access Fault Detail element. If create is set to true this element is creted if not present.

5.74.4.4 void Arc::SOAPFault::Node (const char * *node*)

Set content of Fault Node element to 'node'

5.74.4.5 std::string Arc::SOAPFault::Node (void)

Returns content of Fault Node element

5.74.4.6 Arc::SOAPFault::operator bool (void) `[inline]`

Returns true if instance refers to SOAP Fault

5.74.4.7 void Arc::SOAPFault::Reason (const char * *reason*) `[inline]`

Set Fault Reason element at top level

5.74.4.8 void Arc::SOAPFault::Reason (int *num*, const char * *reason*)

Set Fault Reason content at various levels to 'reason'

5.74.4.9 std::string Arc::SOAPFault::Reason (int *num* = 0)

Returns content of Fault Reason element at various levels

5.74.4.10 void Arc::SOAPFault::Role (const char * *role*)

Set content of Fault Role element to 'role'

5.74.4.11 std::string Arc::SOAPFault::Role (void)

Returns content of Fault Role element

5.74.4.12 void Arc::SOAPFault::Subcode (int *level*, const char * *subcode*)

Set Fault Subcode element at various levels (0 is for Code) to 'subcode'

5.74.4.13 std::string Arc::SOAPFault::Subcode (int *level*)

Returns Fault Subcode element at various levels (0 is for Code)

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

- SOAPEnvelope.h

5.75 Arc::SOAPMessage Class Reference

Message restricted to SOAP payload.

`#include <SOAPMessage.h>`

Public Member Functions

- SOAPMessage (void)
- SOAPMessage (long msg_ptr_addr)
- SOAPMessage (Arc::Message &msg)
- ~SOAPMessage (void)
- Arc::SOAPEnvelope * Payload (void)
- void Payload (Arc::SOAPEnvelope *new_payload)
- Arc::MessageAttributes * Attributes (void)
- void **Attributes** (Arc::MessageAttributes *attributes)
- Arc::MessageAuth * **Auth** (void)
- void **Auth** (Arc::MessageAuth *auth)
- Arc::MessageContext * **Context** (void)
- void **Context** (Arc::MessageContext *context)

5.75.1 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.

5.75.2 Constructor & Destructor Documentation

5.75.2.1 Arc::SOAPMessage::SOAPMessage (void) [inline]

Dummy constructor

5.75.2.2 Arc::SOAPMessage::SOAPMessage (long *msg_ptr_addr*)

Copy constructor. Used by language bindigs

5.75.2.3 Arc::SOAPMessage::SOAPMessage (Arc::Message & *msg*)

Copy constructor. Ensures shallow copy.

5.75.2.4 Arc::SOAPMessage::~SOAPMessage (void)

Destructor does not affect refered objects

5.75.3 Member Function Documentation

5.75.3.1 Arc::MessageAttributes* Arc::SOAPMessage::Attributes (void) [inline]

Returns a pointer to the current attributes object or NULL if no attributes object has been assigned.

5.75.3.2 void Arc::SOAPMessage::Payload (Arc::SOAPEnvelope * *new_payload*)

Replace payload with a COPY of new one

5.75.3.3 Arc::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

5.76 Arc::Time Class Reference

A class for storing and manipulating times.

`#include <DateTime.h>`

Public Member Functions

- Time ()
- Time (const time_t &)
- Time (const std::string &)
- Time & operator= (const time_t &)
- Time & operator= (const Time &)
- void SetTime (const time_t &)
- time_t GetTime () const
- operator std::string () const
- std::string str (const TimeFormat &=time_format) const
- bool operator< (const Time &) const
- bool operator> (const Time &) const
- bool operator<= (const Time &) const
- bool operator>= (const Time &) const
- bool operator== (const Time &) const
- bool operator!= (const Time &) const
- Time operator+ (const Period &) const
- Time operator- (const Period &) const

Static Public Member Functions

- static void SetFormat (const TimeFormat &)
- static TimeFormat GetFormat ()

5.76.1 Detailed Description

A class for storing and manipulating times.

5.76.2 Constructor & Destructor Documentation

5.76.2.1 Arc::Time::Time ()

Default constructor. The time is put equal the current time.

5.76.2.2 Arc::Time::Time (const time_t &)

Constructor that takes a time_t variable and stores it.

5.76.2.3 Arc::Time::Time (const std::string &)

Constructor that tries to convert a string into a time_t.

5.76.3 Member Function Documentation

5.76.3.1 static TimeFormat Arc::Time::GetFormat () [static]

Gets the default format for time strings.

5.76.3.2 time_t Arc::Time::GetTime () const

gets the time

5.76.3.3 Arc::Time::operator std::string () const

Returns a string representation of the time, using the default format.

5.76.3.4 bool Arc::Time::operator!= (const Time &) const

Comparing two Time objects.

5.76.3.5 Time Arc::Time::operator+ (const Period &) const

Adding Time object with Period object.

5.76.3.6 Time Arc::Time::operator- (const Period &) const

Subtracting Period object from Time object.

5.76.3.7 bool Arc::Time::operator< (const Time &) const

Comparing two Time objects.

5.76.3.8 bool Arc::Time::operator<= (const Time &) const

Comparing two Time objects.

5.76.3.9 Time& Arc::Time::operator= (const Time &)

Assignment operator from a Time.

5.76.3.10 Time& Arc::Time::operator= (const time_t &)

Assignment operator from a time_t.

5.76.3.11 bool Arc::Time::operator== (const Time &) const

Comparing two Time objects.

5.76.3.12 bool Arc::Time::operator> (const Time &) const

Comparing two Time objects.

5.76.3.13 bool Arc::Time::operator>= (const Time &) const

Comparing two Time objects.

5.76.3.14 static void Arc::Time::SetFormat (const TimeFormat &) [static]

Sets the default format for time strings.

5.76.3.15 void Arc::Time::SetTime (const time_t &)

sets the time

5.76.3.16 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:

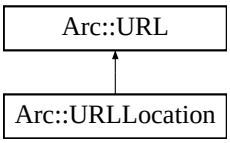
- DateTime.h

5.77 Arc::URL Class Reference

Class to hold general URL's.

`#include <URL.h>`

Inheritance diagram for Arc::URL::



Public Member Functions

- URL ()
- URL (const std::string &url)
- virtual ~URL ()
- const std::string & Protocol () const
- void ChangeProtocol (const std::string &newprot)
- const std::string & Username () const
- const std::string & Passwd () const
- const std::string & Host () const
- void ChangeHost (const std::string &newhost)
- int Port () const
- void ChangePort (int newport)
- const std::string & Path () const
- void ChangePath (const std::string &newpath)
- std::string BaseDN () const
- const std::map< std::string, std::string > & HTTPOptions () const
- const std::string & HTTPOption (const std::string &option, const std::string &undefined="") const
- const std::map< std::string, std::string > & Options () const
- const std::string & Option (const std::string &option, const std::string &undefined="") const
- void AddOption (const std::string &option, const std::string &value, bool overwrite=true)
- const std::list< URLLocation > & Locations () const
- const std::map< std::string, std::string > & CommonLocOptions () const
- const std::string & CommonLocOption (const std::string &option, const std::string &undefined="") const
- virtual std::string str () const
- virtual std::string fullstr () const
- virtual std::string ConnectionURL () const
- bool operator< (const URL &url) const
- bool operator== (const URL &url) const
- operator bool () const
- bool **operator!** () const

Static Protected Member Functions

- static std::string BaseDN2Path (const std::string &)
- static std::string Path2BaseDN (const std::string &)

Protected Attributes

- std::string protocol
- std::string username
- std::string passwd
- std::string host
- int port
- std::string path
- std::map< std::string, std::string > httpoptions
- std::map< std::string, std::string > urloptions
- std::list< URLLocation > locations
- std::map< std::string, std::string > commonlocoptions

Friends

- std::ostream & operator<< (std::ostream &out, const URL &u)

5.77.1 Detailed Description

Class to hold general URL's.

The URL is split into protocol, hostname, port and path.

5.77.2 Constructor & Destructor Documentation

5.77.2.1 Arc::URL::URL ()

Empty constructor. Necessary when the class is part of another class and the like.

5.77.2.2 Arc::URL::URL (const std::string & *url*)

Constructs a new URL from a string representation.

5.77.2.3 virtual Arc::URL::~URL () [virtual]

URL Destructor

5.77.3 Member Function Documentation

5.77.3.1 void Arc::URL::AddOption (const std::string & *option*, const std::string & *value*, bool *overwrite* = true)

Adds a URL option.

5.77.3.2 std::string Arc::URL::BaseDN () const

In case of ldap-protocol, return the basedn of the URL.

5.77.3.3 static std::string Arc::URL::BaseDN2Path (const std::string &) [static, protected]

a private method that converts an ldap basedn to a path.

5.77.3.4 void Arc::URL::ChangeHost (const std::string & *newhost*)

Changes the hostname of the URL.

5.77.3.5 void Arc::URL::ChangePath (const std::string & *newpath*)

Changes the path of the URL.

5.77.3.6 void Arc::URL::ChangePort (int *newport*)

Changes the port of the URL.

5.77.3.7 void Arc::URL::ChangeProtocol (const std::string & *newprot*)

Changes the protocol of the URL.

5.77.3.8 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.

5.77.3.9 const std::map<std::string, std::string>& Arc::URL::CommonLocOptions () const

Returns the common location options if any.

5.77.3.10 virtual std::string Arc::URL::ConnectionURL () const [virtual]

Returns a string representation with protocol, host and port only

5.77.3.11 virtual std::string Arc::URL::fullstr () const [virtual]

Returns a string representation including options and locations

Reimplemented in Arc::URLLocation.

5.77.3.12 const std::string& Arc::URL::Host () const

Returns the hostname of the URL.

5.77.3.13 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.

5.77.3.14 const std::map<std::string, std::string>& Arc::URL::HTTPOptions () const

Returns HTTP options if any.

5.77.3.15 const std::list<URLLocation>& Arc::URL::Locations () const

Returns the locations if any.

5.77.3.16 Arc::URL::operator bool () const

Check if instance holds valid URL

5.77.3.17 bool Arc::URL::operator< (const URL & *url*) const

Compares one URL to another

5.77.3.18 bool Arc::URL::operator== (const URL & *url*) const

Is one URL equal to another?

5.77.3.19 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.

5.77.3.20 const std::map<std::string, std::string>& Arc::URL::Options () const

Returns URL options if any.

5.77.3.21 const std::string& Arc::URL::Passwd () const

Returns the password of the URL.

5.77.3.22 const std::string& Arc::URL::Path () const

Returns the path of the URL.

5.77.3.23 static std::string Arc::URL::Path2BaseDN (const std::string &) [static, protected]

a private method that converts an ldap path to a basedn.

5.77.3.24 int Arc::URL::Port () const

Returns the port of the URL.

5.77.3.25 const std::string& Arc::URL::Protocol () const

Returns the protocol of the URL.

5.77.3.26 virtual std::string Arc::URL::str () const [virtual]

Returns a string representation of the URL.
Reimplemented in Arc::URLLocation.

5.77.3.27 const std::string& Arc::URL::Username () const

Returns the username of the URL.

5.77.4 Friends And Related Function Documentation

5.77.4.1 std::ostream& operator<< (std::ostream & *out*, const URL & *u*) [friend]

Overloaded operator << to print a URL.

5.77.5 Member Data Documentation

5.77.5.1 std::map<std::string, std::string> Arc::URL::commonlocoptions [protected]

common location options for index server URLs.

5.77.5.2 std::string Arc::URL::host [protected]

hostname of the url.

5.77.5.3 std::map<std::string, std::string> Arc::URL::httpoptions [protected]

HTTP options of the url.

5.77.5.4 std::list<URLLocation> Arc::URL::locations [protected]

locations for index server URLs.

5.77.5.5 std::string Arc::URL::passwd [protected]

password of the url.

5.77.5.6 std::string Arc::URL::path [protected]

the url path.

5.77.5.7 int Arc::URL::port [protected]

portnumber of the url.

5.77.5.8 std::string Arc::URL::protocol [protected]

the url protocol.

5.77.5.9 std::map<std::string, std::string> Arc::URL::urloptions [protected]

options of the url.

5.77.5.10 std::string Arc::URL::username [protected]

username of the url.

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

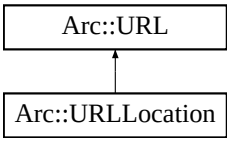
- URL.h

5.78 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

5.78.1 Detailed Description

Class to hold a resolved URL location.

It is specific to file indexing service registrations.

5.78.2 Constructor & Destructor Documentation

5.78.2.1 Arc::URLLocation::URLLocation (const std::string & *url*)

Creates a URLLocation from a string representaion.

5.78.2.2 Arc::URLLocation::URLLocation (const std::string & *url*, const std::string & *name*)

Creates a URLLocation from a string representaion and a name.

5.78.2.3 Arc::URLLocation::URLLocation (const URL & *url*)

Creates a URLLocation from a URL.

5.78.2.4 Arc::URLLocation::URLLocation (const URL & *url*, const std::string & *name*)

Creates a URLLocation from a URL and a name.

5.78.2.5 Arc::URLLocation::URLLocation (const std::map< std::string, std::string > & *options*, const std::string & *name*)

Creates a URLLocation from options and a name.

5.78.2.6 virtual Arc::URLLocation::~URLLocation () [virtual]

URLLocation destructor.

5.78.3 Member Function Documentation

5.78.3.1 virtual std::string Arc::URLLocation::fullstr () const [virtual]

Returns a string representation including options and locations

Reimplemented from Arc::URL.

5.78.3.2 const std::string& Arc::URLLocation::Name () const

Returns the URLLocation name.

5.78.3.3 virtual std::string Arc::URLLocation::str () const [virtual]

Returns a string representation of the URLLocation.

Reimplemented from Arc::URL.

5.78.4 Member Data Documentation

5.78.4.1 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

5.79 Arc::UsernameToken Class Reference

Interface for manipulation of WS-Security Username Token Profile.

`#include <UsernameToken.h>`

Public Member Functions

- UsernameToken (SOAPEnvelope &soap)
- UsernameToken (SOAPEnvelope &soap, std::string &uid, bool pwdtype, bool milliseconds)
- UsernameToken (SOAPEnvelope &soap, std::string &username, bool mac, int iteration, std::string &id)

Protected Attributes

- XMLNode **header_**

5.79.1 Detailed Description

Interface for manipulation of WS-Security Username Token Profile.

5.79.2 Constructor & Destructor Documentation

5.79.2.1 Arc::UsernameToken::UsernameToken (SOAPEnvelope & *soap*)

Link to existing SOAP header to parse username token information

5.79.2.2 Arc::UsernameToken::UsernameToken (SOAPEnvelope & *soap*, std::string & *uid*, bool *pwdtype*, bool *milliseconds*)

Set username token information into the SOAP header

Parameters:

soap the SOAP message

username <wsse:Username>...</wsse:Username>

password <wsse:Password Type="...">...</wsse:Password>

uid <wsse:UsernameToken wsu:ID="...">

pwdtype <wsse:Password Type="...">...</wsse:Password>

milliseconds precision of created time — <wsu:Created>...</wsu/Created>

5.79.2.3 Arc::UsernameToken::UsernameToken (SOAPEnvelope & *soap*, std::string & *username*, bool *mac*, int *iteration*, std::string & *id*)

Set username token information into the SOAP header

Parameters:

soap the SOAP message

username <wsse:Username>...</wsse:Username>

salt <wsse11:Salt>...</wsse11:Salt>

iteration <wsse11:Iteration>...</wsse11:Iteration>

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

- UsernameToken.h

5.80 Arc::WSAEndpointReference Class Reference

Interface for manipulation of WS-Adressing Endpoint Reference.

`#include <WSA.h>`

Public Member Functions

- WSAEndpointReference (XMLNode epr)
- WSAEndpointReference (const std::string &address)
- WSAEndpointReference (void)
- ~WSAEndpointReference (void)
- std::string Address (void) const
- void Address (const std::string &uri)
- WSAEndpointReference & operator= (const std::string &address)
- XMLNode ReferenceParameters (void)
- XMLNode MetaData (void)
- operator XMLNode (void)

Protected Attributes

- XMLNode **epr_**

5.80.1 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.

5.80.2 Constructor & Destructor Documentation

5.80.2.1 Arc::WSAEndpointReference::WSAEndpointReference (XMLNode *epr*)

Linking to existing EPR in XML tree

5.80.2.2 Arc::WSAEndpointReference::WSAEndpointReference (const std::string & *address*)

Creating independent EPR - not implemented

5.80.2.3 Arc::WSAEndpointReference::WSAEndpointReference (void)

Dummy constructor - creates invalid instance

5.80.2.4 Arc::WSAEndpointReference::~WSAEndpointReference (void)

Destructor. All empty elements of EPR XML are destroyed here too

5.80.3 Member Function Documentation

5.80.3.1 void Arc::WSAEndpointReference::Address (const std::string & *uri*)

Assigns new Address value. If EPR had no Address element it is created.

5.80.3.2 std::string Arc::WSAEndpointReference::Address (void) const

Returns Address (URL) encoded in EPR

5.80.3.3 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.

5.80.3.4 Arc::WSAEndpointReference::operator XMLNode (void)

Returns reference to EPR top XML node

5.80.3.5 WSAEndpointReference& Arc::WSAEndpointReference::operator= (const std::string & *address*)

Same as Address(uri)

5.80.3.6 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

5.81 Arc::WSAHeader Class Reference

Interface for manipulation WS-Addressing information in SOAP header.

`#include <WSA.h>`

Public Member Functions

- WSAHeader (SOAPEnvelope &soap)
- WSAHeader (const std::string &action)
- std::string To (void) const
- void To (const std::string &uri)
- WSAEndpointReference From (void)
- WSAEndpointReference ReplyTo (void)
- WSAEndpointReference FaultTo (void)
- std::string Action (void) const
- void Action (const std::string &uri)
- std::string MessageID (void) const
- void MessageID (const std::string &uri)
- std::string RelatesTo (void) const
- void RelatesTo (const std::string &uri)
- std::string RelationshipType (void) const
- void RelationshipType (const std::string &uri)
- XMLNode ReferenceParameter (int n)
- XMLNode ReferenceParameter (const std::string &name)
- XMLNode NewReferenceParameter (const std::string &name)
- operator XMLNode (void)

Static Public Member Functions

- static bool Check (SOAPEnvelope &soap)

Protected Attributes

- XMLNode **header_**
- bool header_allocated_

5.81.1 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.

5.81.2 Constructor & Destructor Documentation

5.81.2.1 Arc::WSAHeader::WSAHeader (SOAPEnvelope & *soap*)

Linking to a header of existing SOAP message

5.81.2.2 Arc::WSAHeader::WSAHeader (const std::string & *action*)

Creating independent SOAP header - not implemented

5.81.3 Member Function Documentation

5.81.3.1 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.

5.81.3.2 std::string Arc::WSAHeader::Action (void) const

Returns content of Action element of SOAP Header.

5.81.3.3 static bool Arc::WSAHeader::Check (SOAPEnvelope & *soap*) [static]

Tells if specified SOAP message has WSA header

5.81.3.4 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.

5.81.3.5 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.

5.81.3.6 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.

5.81.3.7 std::string Arc::WSAHeader::MessageID (void) const

Returns content of MessageID element of SOAP Header.

5.81.3.8 XMLNode Arc::WSAHeader::NewReferenceParameter (const std::string & *name*)

Creates new ReferenceParameter element with specified name. Returns reference to created element.

5.81.3.9 Arc::WSAHeader::operator XMLNode (void)

Returns reference to SOAP Header - not implemented

5.81.3.10 XMLNode Arc::WSAHeader::ReferenceParameter (const std::string & *name*)

Returns first ReferenceParameter element with specified name

5.81.3.11 XMLNode Arc::WSAHeader::ReferenceParameter (int *n*)

Return n-th ReferenceParameter element

5.81.3.12 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.

5.81.3.13 std::string Arc::WSAHeader::RelatesTo (void) const

Returns content of RelatesTo element of SOAP Header.

5.81.3.14 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.

5.81.3.15 std::string Arc::WSAHeader::RelationshipType (void) const

Returns content of RelationshipType element of SOAP Header.

5.81.3.16 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.

5.81.3.17 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.

5.81.3.18 std::string Arc::WSAHeader::To (void) const

Returns content of To element of SOAP Header.

5.81.4 Member Data Documentation

5.81.4.1 bool Arc::WSAHeader::header_allocated_ [protected]

SOAP header element

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

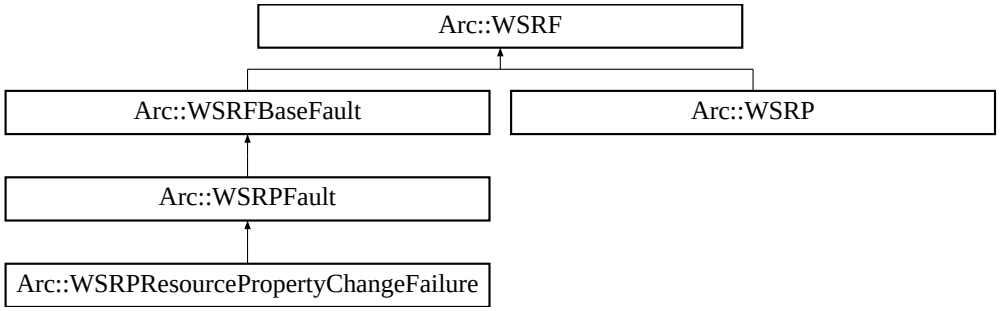
- WSA.h

5.82 Arc::WSRF Class Reference

Base class for every WSRF message.

`#include <WSRF.h>`

Inheritance diagram for Arc::WSRF::



Public Member Functions

- WSRF (SOAPEnvelope &soap, const std::string &action="")
- WSRF (bool fault=false, const std::string &action="")
- virtual SOAPEnvelope & SOAP (void)
- virtual operator bool (void)
- virtual bool **operator!** (void)

Protected Member Functions

- void set_namespaces (void)

Protected Attributes

- SOAPEnvelope & **soap_**
- bool allocated_
- bool valid_

5.82.1 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.

5.82.2 Constructor & Destructor Documentation

5.82.2.1 Arc::WSRF::WSRF (SOAPEnvelope & *soap*, const std::string & *action* = "")

Constructor - creates object out of supplied SOAP tree.

5.82.2.2 Arc::WSRF::WSRF (bool *fault* = `false`, const std::string & *action* = `""`)

Constructor - creates new WSRF object

5.82.3 Member Function Documentation

5.82.3.1 virtual Arc::WSRF::operator bool (void) `[inline, virtual]`

Returns true if instance is valid

5.82.3.2 void Arc::WSRF::set_namespaces (void) `[protected]`

set WS Resource namespaces and default prefixes in SOAP message

Reimplemented in Arc::WSRP, and Arc::WSRFBaseFault.

5.82.3.3 virtual SOAPEnvelope& Arc::WSRF::SOAP (void) `[inline, virtual]`

Direct access to underlying SOAP element

5.82.4 Member Data Documentation

5.82.4.1 bool Arc::WSRF::allocated_ `[protected]`

Associated SOAP message - it's SOAP message after all

5.82.4.2 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:

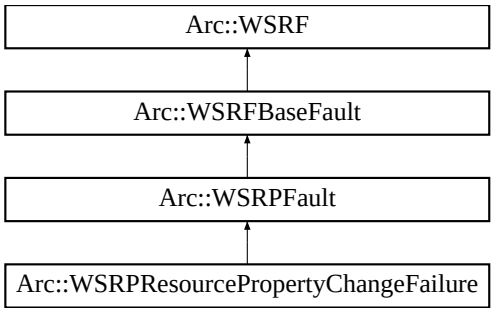
- WSRF.h

5.83 Arc::WSRFBaseFault Class Reference

Base class for WSRF fault messages.

`#include <WSRFBaseFault.h>`

Inheritance diagram for Arc::WSRFBaseFault::



Public Member Functions

- WSRFBaseFault (SOAPEnvelope &soap)
- WSRFBaseFault (const std::string &type)
- std::string **Type** (void)
- Time **Timestamp** (void)
- void **Timestamp** (Time)
- WSAEndpointReference **Originator** (void)
- void **ErrorCode** (const std::string &dialect, const XMLNode &error)
- XMLNode **ErrorCode** (void)
- std::string **ErrorCodeDialect** (void)
- void **Description** (int pos, const std::string &desc, const std::string &lang)
- std::string **Description** (int pos)
- std::string **DescriptionLang** (int pos)
- void **FaultCause** (int pos, const XMLNode &cause)
- XMLNode **FaultCause** (int pos)

Protected Member Functions

- void set_namespaces (void)

5.83.1 Detailed Description

Base class for WSRF fault messages.

Use classes inherited from it for specific faults.

5.83.2 Constructor & Destructor Documentation

5.83.2.1 Arc::WSRFBaseFault::WSRFBaseFault (SOAPEnvelope & *soap*)

Constructor - creates object out of supplied SOAP tree.

5.83.2.2 Arc::WSRFBaseFault::WSRFBaseFault (const std::string & *type*)

Constructor - creates new WSRF fault

5.83.3 Member Function Documentation

5.83.3.1 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:

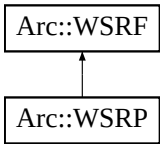
- WSRFBaseFault.h

5.84 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)

5.84.1 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.

5.84.2 Constructor & Destructor Documentation

5.84.2.1 Arc::WSRP::WSRP (bool *fault* = `false`, const std::string & *action* = `""`)

Constructor - prepares object for creation of new WSRP request/response/fault

5.84.2.2 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.

5.84.3 Member Function Documentation

5.84.3.1 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:

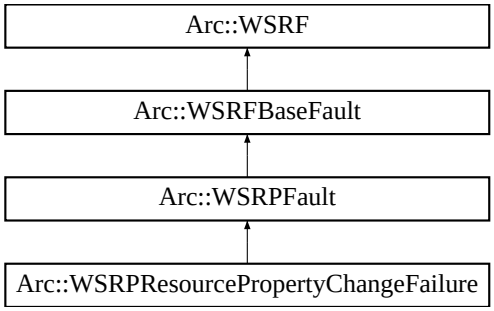
- WSResourceProperties.h

5.85 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)

5.85.1 Detailed Description

Base class for WS-ResourceProperties faults.

5.85.2 Constructor & Destructor Documentation

5.85.2.1 Arc::WSRPFault::WSRPFault (SOAPEnvelope & *soap*)

Constructor - creates object out of supplied SOAP tree.

5.85.2.2 Arc::WSRPFault::WSRPFault (const std::string & *type*)

Constructor - creates new WSRP fault

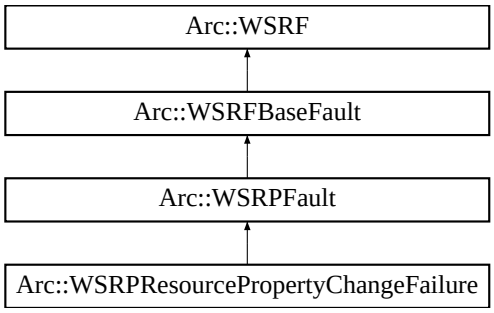
The documentation for this class was generated from the following file:

- WSResourceProperties.h

5.86 Arc::WSRPResourcePropertyChangeFailure Class Reference

`#include <WSResourceProperties.h>`

Inheritance diagram for Arc::WSRPResourcePropertyChangeFailure::



Public Member Functions

- WSRPResourcePropertyChangeFailure (SOAPEnvelope &soap)
- WSRPResourcePropertyChangeFailure (const std::string &type)
- XMLNode **CurrentProperties** (bool create=false)
- XMLNode **RequestedProperties** (bool create=false)

5.86.1 Detailed Description

Base class for WS-ResourceProperties faults which contain ResourcePropertyChangeFailure

5.86.2 Constructor & Destructor Documentation

5.86.2.1 Arc::WSRPResourcePropertyChangeFailure::WSRPResourcePropertyChangeFailure (SOAPEnvelope & *soap*) [inline]

Constructor - creates object out of supplied SOAP tree.

5.86.2.2 Arc::WSRPResourcePropertyChangeFailure::WSRPResourcePropertyChangeFailure (const std::string & *type*) [inline]

Constructor - creates new WSRP fault

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

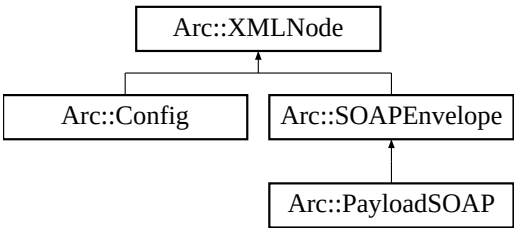
- WSResourceProperties.h

5.87 Arc::XMLNode Class Reference

Wrapper for LibXML library Tree interface.

`#include <XMLNode.h>`

Inheritance diagram for Arc::XMLNode::



Public Member Functions

- XMLNode (void)
- XMLNode (const XMLNode &node)
- XMLNode (const std::string &xml)
- XMLNode (const char *xml, int len=-1)
- XMLNode (const Arc::NS &ns, const char *name)
- ~XMLNode (void)
- void New (XMLNode &new_node) const
- operator bool (void) const
- bool operator! (void) const
- XMLNode Child (int n=0) const
- XMLNode operator[ ] (const char *name) const
- XMLNode operator[ ] (const std::string &name) const
- XMLNode operator[ ] (int n) const
- 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
- void Name (const char *name)
- void Name (const std::string &name)
- void GetXML (std::string &xml) const
- void GetDoc (std::string &xml) const
- operator std::string (void) const
- XMLNode & operator= (const char *content)
- XMLNode & operator= (const std::string &content)
- void Set (const std::string &content)
- XMLNode & operator= (const XMLNode &node)
- XMLNode Attribute (int n=0) const
- XMLNode Attribute (const char *name) const
- XMLNode Attribute (const std::string &name) const
- XMLNode NewAttribute (const char *name)
- XMLNode NewAttribute (const std::string &name)

- int AttributesSize (void) const
- void Namespaces (const NS &namespaces)
- NS Namespaces (void)
- std::string NamespacePrefix (const char *urn)
- XMLNode NewChild (const char *name, int n=-1, bool global_order=false)
- XMLNode NewChild (const std::string &name, int n=-1, bool global_order=false)
- XMLNode NewChild (const char *name, const NS &namespaces, int n=-1, bool global_order=false)
- XMLNode NewChild (const std::string &name, const NS &namespaces, int n=-1, bool global_order=false)
- XMLNode NewChild (const XMLNode &node, int n=-1, bool global_order=false)
- void Replace (const XMLNode &node)
- void Destroy (void)
- std::list< XMLNode > XPathLookup (const std::string &xpathExpr, const Arc::NS &nsList)
- XMLNode GetRoot (void)

Protected Member Functions

- XMLNode (xmlNodePtr node)

Protected Attributes

- xmlNodePtr **node_**
- bool is_owner_
- bool is_temporary_

Friends

- class **XMLNodeContainer**
- bool MatchXMLName (const XMLNode &node1, const XMLNode &node2)
- bool MatchXMLName (const XMLNode &node, const char *name)
- bool MatchXMLName (const XMLNode &node, const std::string &name)
- bool MatchXMLNamespace (const XMLNode &node1, const XMLNode &node2)
- bool MatchXMLNamespace (const XMLNode &node, const char *uri)
- bool MatchXMLNamespace (const XMLNode &node, const std::string &uri)

5.87.1 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.

5.87.2 Constructor & Destructor Documentation

5.87.2.1 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.

5.87.2.2 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.

5.87.2.3 Arc::XMLNode::XMLNode (const XMLNode & *node*) [inline]

Copies existing instance. Underlying XML element is NOT copied. Ownership is NOT inherited.

5.87.2.4 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

5.87.2.5 Arc::XMLNode::XMLNode (const char * *xml*, int *len* = -1)

Same as previous

5.87.2.6 Arc::XMLNode::XMLNode (const Arc::NS & *ns*, const char * *name*)

Creates empty XML document structure with specified namespaces. Created XML contains only root element named 'name'. Created structure is pointed and owned by constructed instance

5.87.2.7 Arc::XMLNode::~XMLNode (void)

Destructor Also destroys underlying XML document if owned by this instance

5.87.3 Member Function Documentation

5.87.3.1 XMLNode Arc::XMLNode::Attribute (const std::string & *name*) const [inline]

Returns XMLNode instance representing first attribute of node with specified by name

5.87.3.2 XMLNode Arc::XMLNode::Attribute (const char * *name*) const

Returns XMLNode instance representing first attribute of node with specified by name

5.87.3.3 XMLNode Arc::XMLNode::Attribute (int *n* = 0) const

Returns list of all attributes of node.

Returns XMLNode instance reresenting n-th attribute of node.

5.87.3.4 int Arc::XMLNode::AttributesSize (void) const

Returns number of attributes of node

5.87.3.5 XMLNode Arc::XMLNode::Child (int *n* = 0) const

Returns XMLNode instance representing n-th child of XML element. If such does not exist invalid XMLNode instance is returned

5.87.3.6 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

5.87.3.7 std::string Arc::XMLNode::FullName (void) const [inline]

Returns prefix:name of XML node

5.87.3.8 XMLNode Arc::XMLNode::Get (const std::string & *name*) const [inline]

Same as operator[ ]

5.87.3.9 void Arc::XMLNode::GetDoc (std::string & *xml*) const

Fills argument with whole XML document textual representation

5.87.3.10 XMLNode Arc::XMLNode::GetRoot (void)

Get the root node from any child node of the tree

5.87.3.11 void Arc::XMLNode::GetXML (std::string & *xml*) const

Fills argument with this instance XML subtree textual representation

Reimplemented in Arc::SOAPEnvelope.

5.87.3.12 void Arc::XMLNode::Name (const std::string & *name*) [inline]

Assigns new name to XML node

5.87.3.13 void Arc::XMLNode::Name (const char * *name*)

Assigns new name to XML node

5.87.3.14 std::string Arc::XMLNode::Name (void) const

Returns name of XML node

5.87.3.15 std::string Arc::XMLNode::NamespacePrefix (const char * *urn*)

Returns prefix of specified namespace. Empty string if no such namespace.

5.87.3.16 NS Arc::XMLNode::Namespaces (void)

Returns namespaces known at this node

5.87.3.17 void Arc::XMLNode::Namespaces (const NS & *namespaces*)

Assigns namespaces of XML document at point specified by this instance. If namespace already exists it gets new prefix. New namespaces are added. It is usefull to apply this method to XML being processed in order to refer to it's elements by known prefix.

Reimplemented in Arc::SOAPEnvelope.

5.87.3.18 void Arc::XMLNode::New (XMLNode & *new_node*) const

Creates a copy of XML (sub)tree. If object does not represent whole document - top level document is created. 'new_node' becomes a pointer owning new XML document.

5.87.3.19 XMLNode Arc::XMLNode::NewAttribute (const std::string & *name*) [inline]

Creates new attribute with specified name.

5.87.3.20 XMLNode Arc::XMLNode::NewAttribute (const char * *name*)

Creates new attribute with specified name.

5.87.3.21 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

5.87.3.22 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)

5.87.3.23 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)

5.87.3.24 XMLNode Arc::XMLNode::NewChild (const std::string & *name*, int *n* = -1, bool *global_order* = false) [inline]

Same as NewChild(const char*,int,bool)

5.87.3.25 XMLNode Arc::XMLNode::NewChild (const char * *name*, int *n* = -1, bool *global_order* = false)

Creates new child XML element at specified position with specified name. Default is to put it at end of list. If global order is true position applies to whole set of children, otherwise only to children of same name

5.87.3.26 Arc::XMLNode::operator bool (void) const [inline]

Returns true if instance points to XML element - valid instance

5.87.3.27 Arc::XMLNode::operator std::string (void) const

Returns textual content of node excluding content of children nodes

5.87.3.28 bool Arc::XMLNode::operator! (void) const [inline]

Returns true if instance does not point to XML element - invalid instance

5.87.3.29 XMLNode& Arc::XMLNode::operator= (const XMLNode & *node*)

Make instance refer to another XML node. Ownership is not inherited.

5.87.3.30 XMLNode& Arc::XMLNode::operator= (const std::string & *content*) [inline]

Sets textual content of node. All existing children nodes are discarded.

5.87.3.31 XMLNode& Arc::XMLNode::operator= (const char * *content*)

Sets textual content of node. All existing children nodes are discarded.

5.87.3.32]

XMLNode Arc::XMLNode::operator[ ] (int *n*) const

Returns XMLNode instance representing n-th node in sequence of siblings of same name. It's main purpose is to be used to retrieve element in array of children of same name like node["name"][5]

5.87.3.33]

XMLNode Arc::XMLNode::operator[ ] (const std::string & *name*) const [inline]

Similar to previous method

5.87.3.34]

XMLNode Arc::XMLNode::operator[ ] (const char * *name*) const

Returns XMLNode instance representing first child element with specified name. Name may be "namespace_prefix:name" or simply "name". In last case namespace is ignored. If such node does not exist invalid XMLNode instance is returned

5.87.3.35 std::string Arc::XMLNode::Prefix (void) const

Returns namespace prefix of XML node

5.87.3.36 void Arc::XMLNode::Replace (const XMLNode & *node*)

Makes a copy of supplied XML node and makes this instance refere to it

5.87.3.37 void Arc::XMLNode::Set (const std::string & *content*) [inline]

Same as operator=. Used for bindings.

5.87.3.38 int Arc::XMLNode::Size (void) const

Returns number of children nodes

5.87.3.39 std::list<XMLNode> Arc::XMLNode::XPathLookup (const std::string & *xpathExpr*, const Arc::NS & *nsList*)

Uses xPath to look up the whole xml structure, Returns a list of XMLNode points. The xpathExpr should be like "//xx:child1/" which indicates the namespace and node that you would like to find; The nsList is the namespace the result should belong to (e.g. xx="uri:test"). Query is run on whole XML document but only the elements belonging to this XML subtree are returned.

5.87.4 Friends And Related Function Documentation

5.87.4.1 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

5.87.4.2 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

5.87.4.3 bool MatchXMLName (const XMLNode & *node1*, const XMLNode & *node2*) [friend]

Returns true if underlying XML elements have same names

5.87.4.4 bool MatchXMLNamespace (const XMLNode & *node*, const std::string & *uri*) [friend]

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

5.87.4.5 bool MatchXMLNamespace (const XMLNode & *node*, const char * *uri*) [friend]

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

5.87.4.6 bool MatchXMLNamespace (const XMLNode & *node1*, const XMLNode & *node2*) [friend]

Returns true if underlying XML elements belong to same namespaces

5.87.5 Member Data Documentation

5.87.5.1 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.

5.87.5.2 bool Arc::XMLNode::is_temporary_ [protected]

This variable is for future

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

- XMLNode.h

5.88 Arc::XMLNodeContainer Class Reference

`#include <XMLNode.h>`

Public Member Functions

- XMLNodeContainer (void)
- XMLNodeContainer (const XMLNodeContainer &)
- XMLNodeContainer & **operator=** (const XMLNodeContainer &)
- void **Add** (const XMLNode &)
- void **Add** (const std::list< XMLNode > &)
- void **AddNew** (const XMLNode &)
- void **AddNew** (const std::list< XMLNode > &)
- int **Size** (void)
- XMLNode **operator[]** (int)

5.88.1 Detailed Description

Container for multiple XMLNode elements

5.88.2 Constructor & Destructor Documentation

5.88.2.1 Arc::XMLNodeContainer::XMLNodeContainer (void)

Default constructor

5.88.2.2 Arc::XMLNodeContainer::XMLNodeContainer (const XMLNodeContainer &)

Copy constructor

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

- XMLNode.h

Index