KnowARC Reference Manual

Generated by Doxygen 1.4.7

Thu Aug 2 15:40:35 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
- struct **xsd__hexBinary**
- struct **xsd__base64Binary**
- class **BasicType**
- class **String**
- class **Boolean**
- class **Float**
- class **Double**
- class **Decimal**
- class **Duration**
- class **DateTime**
- class **AnyURI**
- class **QName**
- class **NOTATION**
- class **NormalizedString**
- class **Token**
- class **Language**
- class **IDREFS**
- class **ENTITIES**
- class **NMTOKEN**
- class **NMTOKENS**
- class **Name**
- class **NCName**
- class **ID**
- class **IDREF**
- class **ENTITY**
- class **Integer**
- class **NonPositiveInteger**
- class **NegativeInteger**
- class **Long**

- class Time
- class **Period**
- 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 **SimpleCondition**
- class URL
- class URLLocation
- class XMLNode
- class **CheckSum**
- class **CRC32Sum**
- class **MD5Sum**
- class **CheckSumAny**
- class DataBufferPar
- class DataHandle
- class DataPoint
- class DataPointIndex
- class DataPointDirect
- class DataSpeed
- class **DataPointGridFTP**
- class **DMCGridFTP**
- class **GlobusResult**
- class **ListerFile**
- class **Lister**
- class **DataPointLFC**
- class **DMCLFC**
- class **DataPointRLS**
- class **DMCRLS**
- class **DMC**
- class **Daemon**
- class **ServerOptions**
- 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 InformationInterface
- class **InformationContainer**
- class **InformationRequest**
- class **InformationResponse**
- class DMCFactory
- class Loader
- class **ChainContext**
- struct loader_descriptor
- class LoaderFactory
- class MCCFactory
- class ModuleManager
- class PDPFactory
- class RegularExpression

  A regular expression class.

- class PlexerEntry

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

- class Plexer

  The Plexer class, used for routing messages to services.

- class SecHandlerFactory
- class ServiceFactory
- class MCCInterface
- class MCC
- class MCC_Status

  A class for communication of MCC statuses.

- class MessagePayload
- class MessageContextElement
- class MessageContext
- class Message
- class AttributeIterator

  An iterator class for accessing multiple values of an attribute.

- class MessageAttributes

  A class for storage of attribute values.

- class MessageAuth
- class PayloadRawInterface
- struct **PayloadRawBuf**
- class PayloadRaw
- class PayloadSOAP
- class PayloadStreamInterface
- class PayloadStream
- class **PDP**
- class Service
- class SOAPFault

- class SOAPEnvelope
- class SOAPMessage
- class **Attribute**
- class **Request**
- class **ArcPDP**
- class **SecHandler**
- class Security

  Common stuff used by security related slasses.

- class **SimpleListAuthZ**
- class **SimpleListPDP**
- class WSAEndpointReference
- class WSAHeader
- class PayloadWSRF
- class WSRP
- class WSRPFault
- 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
- class **WSRFBaseFault**
- class **WSRFResourceUnknownFault**

- class **WSRFResourceUnavailableFault**
- class MCC_HTTP

  A base class for HTTP client and service MCCs.

- class MCC_HTTP_Service
- class MCC_HTTP_Client
- class PayloadHTTP
- class MCC_SOAP

  A base class for SOAP client and service MCCs.

- class MCC_SOAP_Service
- class **MCC_SOAP_Client**
- class MCC_TCP

  A base class for TCP client and service MCCs.

- class MCC_TCP_Service
- class MCC_TCP_Client
- class PayloadTCPSocket
- class MCC_TLS

  A base class for SOAP client and service MCCs.

- class MCC_TLS_Service
- class MCC_TLS_Client
- class **PayloadTLSMCC**
- class **PayloadTLSSocket**
- class PayloadTLSStream

Typedefs

- typedef enum Arc::XSDTYPETag **XSDTYPE**
- typedef char ∗ **xsd__string**
- typedef bool **xsd__boolean**
- typedef float **xsd__float**
- typedef double **xsd__double**
- typedef double **xsd__decimal**
- typedef Period **xsd__duration**
- typedef Time **xsd__dateTime**
- typedef Time **xsd__time**
- typedef Time **xsd__date**
- typedef Time **xsd__gYearMonth**
- typedef Time **xsd__gYear**
- typedef Time **xsd__gMonthDay**
- typedef Time **xsd__gDay**
- typedef Time **xsd__gMonth**
- typedef char ∗ **xsd__anyURI**
- typedef char ∗ **xsd__QName**
- typedef char ∗ **xsd__NOTATION**
- typedef char ∗ **xsd__normalizedString**
- typedef char ∗ **xsd__token**
- typedef char ∗ **xsd__language**

- typedef char ∗ **xsd__IDREFS**
- typedef char ∗ **xsd__ENTITIES**
- typedef char ∗ **xsd__NMTOKEN**
- typedef char ∗ **xsd__NMTOKENS**
- typedef char ∗ **xsd__Name**
- typedef char ∗ **xsd__NCName**
- typedef char ∗ **xsd__ID**
- typedef char ∗ **xsd__IDREF**
- typedef char ∗ **xsd__ENTITY**
- typedef long long **xsd__integer**
- typedef long long **xsd__nonPositiveInteger**
- typedef long long **xsd__negativeInteger**
- typedef long long **xsd__long**
- typedef int **xsd__int**
- typedef short **xsd__short**
- typedef signed char **xsd__byte**
- typedef unsigned long long **xsd__nonNegativeInteger**
- typedef unsigned long long **xsd__unsignedLong**
- typedef unsigned int **xsd__unsignedInt**
- typedef unsigned short **xsd__unsignedShort**
- typedef unsigned char **xsd__unsignedByte**
- typedef unsigned long long **xsd__positiveInteger**
- typedef std::map< std::string, std::string > **NS**
- typedef bool(∗) **rls_lrc_callback_t** (globus_rls_handle_t ∗h, const URL &url, void ∗arg)
- 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**
- typedef std::set< Attribute ∗ > **Attributes**

Enumerations

- enum **XSDTYPETag** {

  XSD_UNKNOWN = 1, **XSD_INT**, **XSD_FLOAT**, **XSD_STRING**,

  XSD_LONG, **XSD_SHORT**, **XSD_BYTE**, **XSD_UNSIGNEDLONG**,

  XSD_BOOLEAN, **XSD_UNSIGNEDINT**, **XSD_UNSIGNEDSHORT**, **XSD_-UNSIGNEDBYTE**,

  XSD_DOUBLE, **XSD_DECIMAL**, **XSD_DURATION**, **XSD_DATETIME**,

  XSD_TIME, **XSD_DATE**, **XSD_GYEARMONTH**, **XSD_GYEAR**,

  XSD_GMONTHDAY, **XSD_GDAY**, **XSD_GMONTH**, **XSD_HEXBINARY**,

  XSD_BASE64BINARY, **XSD_ANYURI**, **XSD_QNAME**, **XSD_NOTATION**,

  XSD_INTEGER, **XSD_ARRAY**, **USER_TYPE**, **XSD_NMTOKEN**,

  XSD_ANY, **XSD_NONNEGATIVEINTEGER**, **XSD_POSITIVEINTEGER**, **XSD_-NONPOSITIVEINTEGER**,

  XSD_NEGATIVEINTEGER, **XSD_NORMALIZEDSTRING**, **XSD_TOKEN**, **XSD_-LANGUAGE**,

XSD_NAME, **XSD_NCNAME**, **XSD_ID**, **XSD_IDREF**,
XSD_IDREFS, **XSD_ENTITY**, **XSD_ENTITIES**, **XSD_NMTOKENS**,
ATTACHMENT }

- enum TimeFormat {
  MDSTime, **ASCTime**, **UserTime**, **ISOTime**,
  UTCTime }
- 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 }
- 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 Time &)
- std::string TimeStamp (const TimeFormat &=Time::GetFormat())
- std::string TimeStamp (Time, const TimeFormat &=Time::GetFormat())
- std::ostream & operator<< (std::ostream &, const Period &)
- 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> std::string tostring (T t, const int width=0, const int precision=0)
- bool CreateThreadFunction (void(∗func)(void ∗), void ∗arg)
- bool MatchXMLName (xmlNodePtr node1, xmlNodePtr node2)
- bool MatchXMLName (xmlNodePtr node, const char ∗name)
- bool MatchXMLName (const XMLNode &node1, const XMLNode &node2)
- bool MatchXMLName (const XMLNode &node, const char ∗name)
- std::string **globus_object_to_string** (globus_object_t ∗err)
- std::ostream & **operator**<< (std::ostream &o, globus_object_t ∗err)
- std::ostream & **operator**<< (std::ostream &o, const GlobusResult &res)
- bool **rls_find_lrcs** (const URL &url, rls_lrc_callback_t callback, void ∗arg)
- bool **rls_find_lrcs** (const URL &url, std::list< URL > lrcs)
- bool **rls_find_lrcs** (std::list< URL > rlis, std::list< URL > lrcs, rls_lrc_callback_t callback, void ∗arg)
- bool **rls_find_lrcs** (std::list< URL > rlis, std::list< URL > lrcs, bool down, bool up, rls_lrc_-callback_t callback, void ∗arg)

- 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)
- BIO_METHOD ∗ **BIO_s_MCC** (void)
- BIO ∗ **BIO_new_MCC** (MCCInterface ∗mcc)
- void **BIO_set_MCC** (BIO ∗b, MCCInterface ∗mcc)

Variables

- Logger **stringLogger**
- const Glib::TimeVal ETERNAL
- const Glib::TimeVal HISTORIC
- Arc::Attribute ∗ root
- Attributes **subjects**
- Attributes **actions**
- Attributes **objects**
- Attributes **conditions**
- const char ∗ **WSRFBaseFaultAction**

4.1.1 Detailed Description

Deal with the serilization and deserilization about basic datatype (Build-in datatype in "XML Schema Part 2: Datatypes Second Edition": http://www.w3.org/TR/xmlschema-2/)

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.

4.1.3.4 enum Arc::WSAFault

WS-Addressing possible faults

Enumerator:

WSAFaultUnknown This is not a fault

WSAFaultInvalidAddressingHeader This is not WS-Addressing fault

4.1.4 Function Documentation

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

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

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

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

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

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

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

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

4.1.4.5 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.

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

This method converts a string to any type

4.1.4.7 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.

4.1.4.8 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.

4.1.4.9 bool Arc::MatchXMLName (xmlNodePtr *node1*, xmlNodePtr *node2*)

Returns true if XML elements have same names

4.1.4.10 bool Arc::MatchXMLName (xmlNodePtr *node*, const char * *name*)

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

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

Returns true if underlying XML elements have same names

4.1.4.12 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

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

Conversion to string.

Conversion from StatusKind to string.

Parameters:

kind The StatusKind to convert.

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

Returns pointer to main buffer of Message payload. NULL if no buffer is present or if payload is not of PayloadRawInterface type.

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

Fills SOAP Fault message look it like corresponding WS-Addressing fault

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

Anylizes 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.

4.1.5.3 class Arc::Attribute∗ Arc::root

<subjects, actions, objects, condition> tuple

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 $\rightarrow$ () 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::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.

5.1.3.3 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.

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

The prefix advance operator.

Advances the iterator to the next value. Works intuitively.

Returns:

A const reference to this iterator.

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

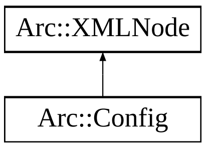
The documentation for this class was generated from the following file:

- MessageAttributes.h

5.2 Arc::Config Class Reference

`#include <ArcConfig.h>`

Inheritance diagram for Arc::Config::



Public Member Functions

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

5.2.1 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.

5.2.2 Constructor & Destructor Documentation

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

Dummy constructor - produces empty structure

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

Loads configuration document from file 'filename'

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

Parse configuration document from memory

5.2.2.4 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.

5.2.3 Member Function Documentation

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

Parse configuration document from file 'filename'

5.2.3.2 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:

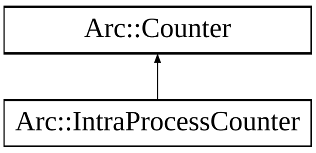
- ArcConfig.h

5.3 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

5.3.1 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(...);
```

5.3.2 Member Typedef Documentation

5.3.2.1 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.

5.3.3 Constructor & Destructor Documentation

5.3.3.1 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.

5.3.3.2 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.

5.3.4 Member Function Documentation

5.3.4.1 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.

5.3.4.2 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.

5.3.4.3 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.

5.3.4.4 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.

5.3.4.5 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.

5.3.4.6 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.

5.3.4.7 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.

5.3.4.8 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.

5.3.4.9 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.

5.3.4.10 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.

5.3.4.11 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.

5.3.4.12 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.

5.3.4.13 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.

5.3.4.14 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.

5.3.5 Friends And Related Function Documentation

5.3.5.1 friend class CounterTicket `[friend]`

The CounterTicket class needs to be a friend.

5.3.5.2 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

5.4 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

5.4.1 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();
```

5.4.2 Constructor & Destructor Documentation

5.4.2.1 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.

5.4.3 Member Function Documentation

5.4.3.1 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.

5.4.3.2 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.

5.4.3.3 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.

5.4.4 Friends And Related Function Documentation

5.4.4.1 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

5.5 Arc::DataBufferPar Class Reference

```
#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**

5.5.1 Detailed Description

This class represents set of buffers used during data transfer.

5.5.2 Constructor & Destructor Documentation

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

Contructor

Parameters:

size size of every buffer in bytes.

blocks number of buffers.

5.5.2.2 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.

5.5.2.3 Arc::DataBufferPar::∼DataBufferPar ()

Destructor.

5.5.3 Member Function Documentation

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

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

5.5.3.2 const CheckSum∗ Arc::DataBufferPar::checksum_object ()

Returns CheckSum object specified in constructor.

5.5.3.3 bool Arc::DataBufferPar::checksum_valid ()

Returns true if checksum was successfully computed.

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

Returns offset following last piece of data transfered.

5.5.3.5 bool Arc::DataBufferPar::eof_read ()

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

5.5.3.6 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.

5.5.3.7 bool Arc::DataBufferPar::eof_write ()

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

5.5.3.8 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.

5.5.3.9 bool Arc::DataBufferPar::error ()

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

5.5.3.10 bool Arc::DataBufferPar::error_read ()

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

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

Informs object if error accured on 'read' side.

Parameters:

v true if error.

5.5.3.12 bool Arc::DataBufferPar::error_transfer ()

Returns true if eror occured inside object.

5.5.3.13 bool Arc::DataBufferPar::error_write ()

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

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

Informs object if error accured on 'write' side.

Parameters:

v true if error.

5.5.3.15 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.

5.5.3.16 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

5.5.3.17 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.

5.5.3.18 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.

5.5.3.19 bool Arc::DataBufferPar::is_notwritten (char ∗ *buf*)

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

Parameters:

buf - address of buffer

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

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

Parameters:

handle buffer's number.

5.5.3.21 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.

5.5.3.22 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.

5.5.3.23 bool Arc::DataBufferPar::is_written (char ∗ *buf*)

Informs object that data was written from buffer.

Parameters:

buf - address of buffer

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

Informs object that data was written from buffer.

Parameters:

handle buffer's number.

5.5.3.25 Arc::DataBufferPar::operator bool (void) `[inline]`

Check if DataBufferPar object is initialized.

5.5.3.26]

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

Direct access to buffer by number.

5.5.3.27 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.

5.5.3.28 bool Arc::DataBufferPar::wait ()

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

5.5.3.29 bool Arc::DataBufferPar::wait_eof ()

Wait till end of transfer happens on any side.

5.5.3.30 bool Arc::DataBufferPar::wait_eof_read ()

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

5.5.3.31 bool Arc::DataBufferPar::wait_eof_write ()

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

5.5.3.32 bool Arc::DataBufferPar::wait_read ()

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

5.5.3.33 bool Arc::DataBufferPar::wait_used ()

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

5.5.3.34 bool Arc::DataBufferPar::wait_write ()

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

5.5.4 Member Data Documentation

5.5.4.1 DataSpeed Arc::DataBufferPar::speed

This object controls transfer speed.

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

- DataBufferPar.h

5.6 Arc::DataHandle Class Reference

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

Public Member Functions

- **DataHandle** (const URL &url)
- DataPoint $*$ **operator** $\rightarrow$ ()
- bool **operator!** ()
- **operator bool** ()

5.6.1 Detailed Description

The DataHandle class is a wrapper around the DataPoint class that simplifies the construction and use and destruction of DataPoint objects

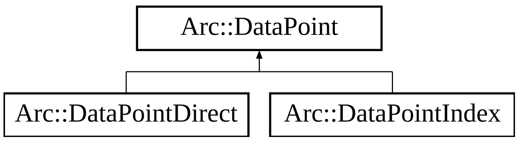
The documentation for this class was generated from the following file:

- DataHandle.h

5.7 Arc::DataPoint Class Reference

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

Inheritance diagram for Arc::DataPoint::



Public Member Functions

- DataPoint (const URL &url)
- virtual bool meta_resolve (bool source __attribute__((unused)))
- virtual bool meta_preregister (bool replication __attribute__((unused)), bool force __attribute__-((unused))=false)
- virtual bool meta_postregister (bool replication __attribute__((unused)))
- virtual bool **meta_register** (bool replication)
- virtual bool meta_preunregister (bool replication __attribute__((unused)))
- virtual bool meta_unregister (bool all __attribute__((unused)))
- virtual bool list_files (std::list< FileInfo > &files __attribute__((unused)), bool resolve __attribute_-_((unused))=true)
- virtual bool get_info (FileInfo &fi __attribute__((unused)))
- virtual bool meta_size_available () const
- virtual void meta_size (unsigned long long int val)
- virtual void meta_size_force (unsigned long long int val)
- virtual unsigned long long int meta_size () const
- virtual bool meta_checksum_available () const
- virtual void meta_checksum (const std::string &val)
- virtual void meta_checksum_force (const std::string &val)
- virtual const std::string & meta_checksum () const
- virtual bool meta_created_available () const
- virtual void meta_created (Time val)
- virtual void meta_created_force (Time val)
- virtual Time meta_created () const
- virtual bool meta_validtill_available () const
- virtual void meta_validtill (Time val)
- virtual void meta_validtill_force (Time val)
- virtual Time meta_validtill () const
- virtual bool meta () const
- virtual bool accepts_meta ()
- virtual bool provides_meta ()
- virtual void meta (const DataPoint &p)
- virtual bool meta_compare (const DataPoint &p) const
- virtual bool meta_stored ()
- virtual bool local () const
- virtual **operator bool** () const
- virtual bool **operator!** () 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 remove_location ()
- virtual bool remove_locations (const DataPoint &p __attribute__((unused)))
- virtual int tries ()
- virtual void tries (int n)
- virtual const URL & base_url () const
- virtual bool add_location (const std::string &meta __attribute__((unused)), const URL &loc __-attribute__((unused)))

Protected Attributes

- URL **url**
- unsigned long long int **meta_size_**
- std::string **meta_checksum_**
- Time **meta_created_**
- Time **meta_validtill_**
- int **tries_left**

Static Protected Attributes

- static Logger **logger**
- static std::string **empty_string_**
- static URL **empty_url_**

Classes

- class FileInfo

5.7.1 Detailed Description

DataPoint 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.

5.7.2 Constructor & Destructor Documentation

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

Constructor requires URL or meta-URL to be provided.

5.7.3 Member Function Documentation

5.7.3.1 virtual bool Arc::DataPoint::accepts_meta () `[inline, virtual]`

If endpoint can have any use from meta information.

Reimplemented in Arc::DataPointIndex.

5.7.3.2 virtual bool Arc::DataPoint::add_location (const std::string &meta __*attribute*__((unused)), const URL &loc __*attribute*__((unused))) `[inline, virtual]`

Add URL to list.

Parameters:

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

loc URL.

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

Returns URL which was passed to constructor.

5.7.3.4 virtual const URL& Arc::DataPoint::current_location () const `[inline, virtual]`

Returns current (resolved) URL.

Reimplemented in Arc::DataPointIndex.

5.7.3.5 virtual const std::string& Arc::DataPoint::current_meta_location () const `[inline, virtual]`

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

Reimplemented in Arc::DataPointIndex.

5.7.3.6 virtual bool Arc::DataPoint::get_info (FileInfo &fi __*attribute*__((unused))) `[inline, virtual]`

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

Parameters:

fi contains retrieved information.

5.7.3.7 virtual bool Arc::DataPoint::have_location () const `[inline, virtual]`

Returns false if out of retries.

Reimplemented in Arc::DataPointIndex.

5.7.3.8 virtual bool Arc::DataPoint::have_locations () const `[inline, virtual]`

Returns true if number of resolved URLs is not 0.

Reimplemented in Arc::DataPointIndex.

5.7.3.9 virtual bool Arc::DataPoint::list_files (std::list< FileInfo > &files __*attribute*__((unused)), bool resolve __*attribute*__((unused)) = `true`) `[inline, virtual]`

Obtain information about objects and their properties available under meta-URL of DataPoint object. It works only for meta-URL.

Parameters:

files list of obtained objects.

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

5.7.3.10 virtual bool Arc::DataPoint::local () const `[inline, virtual]`

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

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

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

Parameters:

p object from which information is taken.

5.7.3.12 virtual bool Arc::DataPoint::meta () const `[inline, virtual]`

Check if URL is meta-URL.

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

5.7.3.13 virtual const std::string& Arc::DataPoint::meta_checksum () const `[inline, virtual]`

Get value of meta-information 'checksum'.

5.7.3.14 virtual void Arc::DataPoint::meta_checksum (const std::string & *val*) `[inline, virtual]`

Set value of meta-information 'checksum' if not already set.

5.7.3.15 virtual bool Arc::DataPoint::meta_checksum_available () const `[inline, virtual]`

Check if meta-information 'checksum' is available.

5.7.3.16 virtual void Arc::DataPoint::meta_checksum_force (const std::string & *val*) `[inline, virtual]`

Set value of meta-information 'checksum'.

5.7.3.17 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.

5.7.3.18 virtual Time Arc::DataPoint::meta_created () const `[inline, virtual]`

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

5.7.3.19 virtual void Arc::DataPoint::meta_created (Time *val*) `[inline, virtual]`

Set value of meta-information 'creation/modification time' if not already set.

5.7.3.20 virtual bool Arc::DataPoint::meta_created_available () const `[inline, virtual]`

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

5.7.3.21 virtual void Arc::DataPoint::meta_created_force (Time *val*) `[inline, virtual]`

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

5.7.3.22 virtual bool Arc::DataPoint::meta_postregister (bool replication __*attribute*__((unused))) `[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.

5.7.3.23 virtual bool Arc::DataPoint::meta_preregister (bool replication __*attribute*__((unused)), bool force __*attribute*__((unused)) = `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.

5.7.3.24 virtual bool Arc::DataPoint::meta_preunregister (bool replication ***__attribute__***((unused))) `[inline, virtual]`

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

5.7.3.25 virtual bool Arc::DataPoint::meta_resolve (bool source ***__attribute__***((unused))) `[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

5.7.3.26 virtual unsigned long long int Arc::DataPoint::meta_size () const `[inline, virtual]`

Get value of meta-information 'size'.

5.7.3.27 virtual void Arc::DataPoint::meta_size (unsigned long long int *val*) `[inline, virtual]`

Set value of meta-information 'size' if not already set.

5.7.3.28 virtual bool Arc::DataPoint::meta_size_available () const `[inline, virtual]`

Check if meta-information 'size' is available.

5.7.3.29 virtual void Arc::DataPoint::meta_size_force (unsigned long long int *val*) `[inline, virtual]`

Set value of meta-information 'size'.

5.7.3.30 virtual bool Arc::DataPoint::meta_stored () `[inline, virtual]`

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

Reimplemented in Arc::DataPointIndex.

5.7.3.31 virtual bool Arc::DataPoint::meta_unregister (bool all *__attribute__*((unused))) `[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.

5.7.3.32 virtual Time Arc::DataPoint::meta_validtill () const `[inline, virtual]`

Get value of meta-information 'validity time'.

5.7.3.33 virtual void Arc::DataPoint::meta_validtill (Time *val*) `[inline, virtual]`

Set value of meta-information 'validity time' if not already set.

5.7.3.34 virtual bool Arc::DataPoint::meta_validtill_available () const `[inline, virtual]`

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

5.7.3.35 virtual void Arc::DataPoint::meta_validtill_force (Time *val*) `[inline, virtual]`

Set value of meta-information 'validity time'.

5.7.3.36 virtual bool Arc::DataPoint::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.

Reimplemented in Arc::DataPointIndex.

5.7.3.37 virtual bool Arc::DataPoint::provides_meta () `[inline, virtual]`

If endpoint can provide at least some meta information directly.

Reimplemented in Arc::DataPointIndex.

5.7.3.38 virtual bool Arc::DataPoint::remove_location () `[inline, virtual]`

Remove current URL from list.

Reimplemented in Arc::DataPointIndex.

5.7.3.39 virtual bool Arc::DataPoint::remove_locations (const DataPoint &p *__attribute__*((unused))) `[inline, virtual]`

Remove locations present in another DataPoint object.

5.7.3.40 virtual void Arc::DataPoint::tries (int *n*) `[virtual]`

Set number of retries.

Reimplemented in Arc::DataPointIndex.

5.7.3.41 virtual int Arc::DataPoint::tries () `[virtual]`

Returns number of retries left.

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

- DataPoint.h

5.8 Arc::DataPoint::FileInfo Class Reference

```
#include <DataPoint.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="")
- operator bool ()

Public Attributes

- std::string **name**
- std::list< URL > **urls**
- unsigned long long int **size**
- std::string **checksum**
- Time **created**
- Time **valid**
- Type **type**

5.8.1 Detailed Description

FileInfo stores information about file (meta-information). Although all members are public it is mot desirable to modify them directly outside DataPoint class.

5.8.2 Member Function Documentation

5.8.2.1 Arc::DataPoint::FileInfo::operator bool (void) `[inline]`

If object is valid.

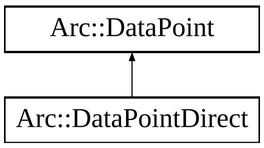
The documentation for this class was generated from the following file:

- DataPoint.h

5.9 Arc::DataPointDirect Class Reference

`#include <DataPointDirect.h>`

Inheritance diagram for Arc::DataPointDirect::



Public Types

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

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)

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**

Classes

- class analyze_t

5.9.1 Detailed Description

DataPointDirect 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.

5.9.2 Member Enumeration Documentation

5.9.2.1 enum Arc::DataPointDirect::failure_reason_t

Reason of transfer failure.

5.9.3 Constructor & Destructor Documentation

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

Constructor

Parameters:

url URL.

5.9.3.2 virtual Arc::DataPointDirect::∼DataPointDirect () `[virtual]`

Destructor. No comments.

5.9.4 Member Function Documentation

5.9.4.1 virtual bool Arc::DataPointDirect::additional_checks () `[virtual]`

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

5.9.4.2 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).

5.9.4.3 virtual bool Arc::DataPointDirect::analyze (analyze_t & *arg*) `[virtual]`

Analyze url and provide hints.

Parameters:

arg returns suggested values.

5.9.4.4 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.

5.9.4.5 virtual failure_reason_t Arc::DataPointDirect::failure_reason () `[virtual]`

Returns reason of transfer failure.

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

List files in directory or service (URL must point to directory/group/service access point).

Parameters:

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

resolve if false no information about attributes will be retrieved.

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

Check if URL is meta-URL.

Reimplemented from Arc::DataPoint.

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

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

Parameters:

v true if allowed.

5.9.4.9 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.

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

Request passive transfers for FTP-like protocols.

Parameters:

true to request.

5.9.4.11 virtual void Arc::DataPointDirect::range (unsigned long long int *start* = `0`, unsigned long long int *end* = `0`) `[virtual]`

Set range of bytes to retrieve. Default values correspond to whole file.

5.9.4.12 virtual bool Arc::DataPointDirect::remove () `[virtual]`

Remove/delete object at URL.

5.9.4.13 virtual bool Arc::DataPointDirect::secure () `[virtual]`

Check if heavy security during data transfer is allowed.

5.9.4.14 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://).

5.9.4.15 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.

5.9.4.16 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.

5.9.4.17 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.

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

Same as stop_reading but for corresponding start_writing.

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

- DataPointDirect.h

5.10 Arc::DataPointDirect::analyze_t Class Reference

```
#include <DataPointDirect.h>
```

Public Attributes

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

5.10.1 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://)

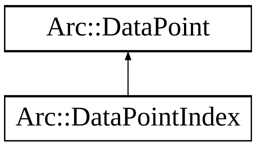
The documentation for this class was generated from the following file:

- DataPointDirect.h

5.11 Arc::DataPointIndex Class Reference

`#include <DataPoint.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 tries (int n)

Protected Member Functions

- void **fix_unregistered** (bool all)

Protected Attributes

- std::list< Location > locations
- std::list< Location >::iterator **location**
- bool **is_metaexisting**
- bool **is_resolved**

Classes

- class Location

5.11.1 Detailed Description

DataPointIndex complements DataPoint with attributes common for meta-URLs It should never be used directly.

5.11.2 Member Function Documentation

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

If endpoint can have any use from meta information.

Reimplemented from Arc::DataPoint.

5.11.2.2 virtual const URL& Arc::DataPointIndex::current_location () const `[inline, virtual]`

Returns current (resolved) URL.

Reimplemented from Arc::DataPoint.

5.11.2.3 virtual const std::string& Arc::DataPointIndex::current_meta_location () const `[inline, virtual]`

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

Reimplemented from Arc::DataPoint.

5.11.2.4 virtual bool Arc::DataPointIndex::have_location () const `[virtual]`

Returns false if out of retries.

Reimplemented from Arc::DataPoint.

5.11.2.5 virtual bool Arc::DataPointIndex::have_locations () const `[virtual]`

Returns true if number of resolved URLs is not 0.

Reimplemented from Arc::DataPoint.

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

Check if URL is meta-URL.

Reimplemented from Arc::DataPoint.

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

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

Reimplemented from Arc::DataPoint.

5.11.2.8 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.

Reimplemented from Arc::DataPoint.

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

If endpoint can provide at least some meta information directly.

Reimplemented from Arc::DataPoint.

5.11.2.10 virtual bool Arc::DataPointIndex::remove_location () `[virtual]`

Remove current URL from list.

Reimplemented from Arc::DataPoint.

5.11.2.11 virtual void Arc::DataPointIndex::tries (int *n*) `[virtual]`

Set number of retries.

Reimplemented from Arc::DataPoint.

5.11.3 Member Data Documentation

5.11.3.1 std::list<Location> 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:

- DataPoint.h

5.12 Arc::DataPointIndex::Location Class Reference

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

Public Member Functions

- **Location** (const URL &url)
- **Location** (const std::string &meta, const URL &url, bool existing=true)

Public Attributes

- std::string **meta**
- URL **url**
- bool **existing**
- void ∗ **arg**

5.12.1 Detailed Description

DataPointIndex::Location represents physical service at which files are located aka "base URL" inculding it's name (as given in Indexing Service). Currently it is used only internally by classes derived from DataPointIndex class and for printing debug information.

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

- DataPoint.h

5.13 Arc::DataSpeed Class Reference

`#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)

5.13.1 Detailed Description

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

5.13.2 Constructor & Destructor Documentation

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

Constructor

Parameters:

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

5.13.2.2 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.

5.13.2.3 Arc::DataSpeed::∼DataSpeed (void)

Destructor.

5.13.3 Member Function Documentation

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

Turn off speed control.

Parameters:

disable true to turn off.

5.13.3.2 bool Arc::DataSpeed::max_inactivity_time_failure () `[inline]`

Check if maximal inactivity time error was triggered.

5.13.3.3 bool Arc::DataSpeed::min_average_speed_failure () `[inline]`

Check if minimal average speed error was triggered.

5.13.3.4 bool Arc::DataSpeed::min_speed_failure () `[inline]`

Check if minimal speed error was triggered.

5.13.3.5 void Arc::DataSpeed::reset (void)

Reset all counters and triggers.

5.13.3.6 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).

5.13.3.7 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.

5.13.3.8 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.

5.13.3.9 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.

5.13.3.10 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

5.13.3.11 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.

5.13.3.12 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).

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

Returns amount of data this object knows about.

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

Check if speed information is going to be printed.

5.13.3.15 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.

5.13.3.16 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

5.14 dmc_descriptor Struct Reference

```
#include <DMCLoader.h>
```

Public Attributes

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

5.14.1 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.

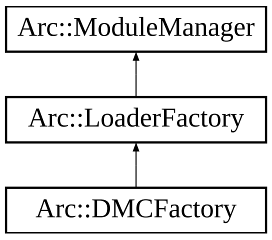
The documentation for this struct was generated from the following file:

- DMCLoader.h

5.15 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)

5.15.1 Detailed Description

This class handles shared libraries containing DMCs

5.15.2 Constructor & Destructor Documentation

5.15.2.1 Arc::DMCFactory::DMCFactory (Config ∗ *cfg*)

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

5.15.3 Member Function Documentation

5.15.3.1 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

5.16 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

5.16.1 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.

5.16.2 Member Function Documentation

5.16.2.1 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.

5.16.2.2 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.

5.16.2.3 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.

5.16.3 Friends And Related Function Documentation

5.16.3.1 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

5.17 Arc::InformationInterface Class Reference

```
#include <InformationInterface.h>
```

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_**

5.17.1 Detailed Description

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.

5.17.2 Constructor & Destructor Documentation

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

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

5.17.3 Member Function Documentation

5.17.3.1 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.

5.17.4 Member Data Documentation

5.17.4.1 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:

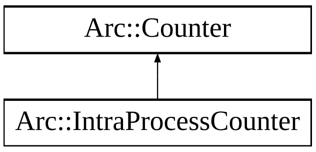
- InformationInterface.h

5.18 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)

5.18.1 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.

5.18.2 Constructor & Destructor Documentation

5.18.2.1 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.

5.18.2.2 virtual Arc::IntraProcessCounter::∼IntraProcessCounter () `[virtual]`

Destructor.

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

5.18.3 Member Function Documentation

5.18.3.1 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.

5.18.3.2 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.

5.18.3.3 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.

5.18.3.4 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.

5.18.3.5 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.

5.18.3.6 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.

5.18.3.7 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.

5.18.3.8 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.

5.18.3.9 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.

5.18.3.10 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

5.19 Arc::Loader Class Reference

`#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, 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**

5.19.1 Detailed Description

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 are destroyed if this object is destroyed. Chains are created in 2 steps. First all components are loaded and corresponding objects are created. Constructor are supplied with corresponding configuration subtrees. During next step components are linked together by calling their Next() methods. Each creates labeled link to next component in a chain. 2 step method has an advantage over 1 step because it allows loops in chains and makes loading procedure 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.

5.19.2 Constructor & Destructor Documentation

5.19.2.1 Arc::Loader::Loader (Config ∗ *cfg*)

Constructor that takes whole XML configuration and creates component chains

5.19.2.2 Arc::Loader::∼Loader ()

Destructor destroys all components created by constructor

5.19.3 Member Function Documentation

5.19.3.1]

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

5.20 Arc::loader_descriptor Struct Reference

```
#include <LoaderFactory.h>
```

Public Attributes

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

5.20.1 Detailed Description

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.

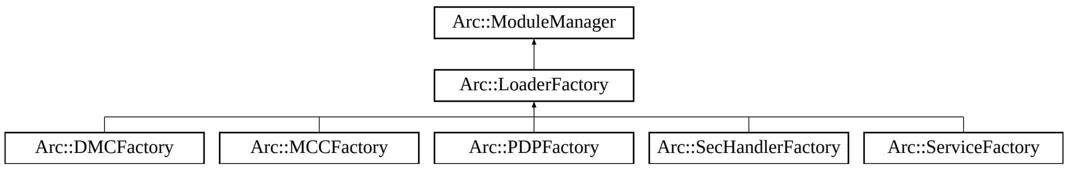
The documentation for this struct was generated from the following file:

- LoaderFactory.h

5.21 Arc::LoaderFactory Class Reference

`#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)

5.21.1 Detailed Description

This class handles shared libraries containing loadable classes

5.21.2 Constructor & Destructor Documentation

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

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

5.21.3 Member Function Documentation

5.21.3.1 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 'id' representing descriptor of elements and calls it's constructor function. Supplied configuration tree is passed to constructor. Returns created instance. This classes must be rewritten in real implementation with proper type casting.

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

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

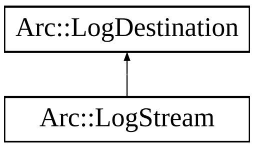
- LoaderFactory.h

5.22 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 ()

5.22.1 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.

5.22.2 Constructor & Destructor Documentation

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

Default constructor.

The only constructor needed by subclasses, since the LogDestination class has no attributes.

5.22.3 Member Function Documentation

5.22.3.1 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

5.23 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 setThreshold (LogLevel threshold)
- LogLevel getThreshold () const
- void msg (LogMessage message)
- void msg (LogLevel level, const std::string &str,...)

Static Public Attributes

- static Logger rootLogger

5.23.1 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.

5.23.2 Constructor & Destructor Documentation

5.23.2.1 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.

5.23.2.2 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.

5.23.3 Member Function Documentation

5.23.3.1 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.

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

Returns the threshold.

Returns the threshold.

Returns:

> The threshold of this Logger.

5.23.3.3 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.

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

Sends a LogMessage.

Sends a LogMessage.

Parameters:

> ***The*** LogMessage to send.

5.23.3.5 void Arc::Logger::setThreshold (LogLevel *threshold*)

Sets the threshold.

This method sets the threshold of the Logger. Any message sent to this Logger that has a level below this threshold will be discarded.

Parameters:

The threshold

5.23.4 Member Data Documentation

5.23.4.1 Logger Arc::Logger::rootLogger `[static]`

The root Logger.

This is the root Logger. It is an ancestor of any other Logger and allways exists.

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

- Logger.h

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

Protected Member Functions

- void setIdentifier (std::string identifier)

Friends

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

5.24.1 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.

5.24.2 Constructor & Destructor Documentation

5.24.2.1 Arc::LogMessage::LogMessage (LogLevel *level*, const std::string & *message*, va_list *v* = `NULL`)

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.

5.24.2.2 Arc::LogMessage::LogMessage (LogLevel *level*, const std::string & *message*, const std::string & *identifier*, va_list *v* = `NULL`)

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.

5.24.3 Member Function Documentation

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

Returns the level of the LogMessage.

Returns the level of the LogMessage.

Returns:

The level of the LogMessage.

5.24.3.2 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.

5.24.4 Friends And Related Function Documentation

5.24.4.1 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.

5.24.4.2 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:

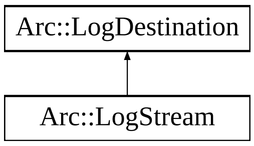
- Logger.h

5.25 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)
- virtual void log (const LogMessage &message)

5.25.1 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.

5.25.2 Constructor & Destructor Documentation

5.25.2.1 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.

5.25.3 Member Function Documentation

5.25.3.1 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.

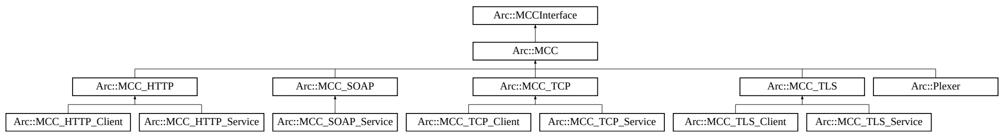
The documentation for this class was generated from the following file:

- Logger.h

5.26 Arc::MCC Class Reference

`#include <MCC.h>`

Inheritance diagram for Arc::MCC::



Public Member Functions

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

Protected Member Functions

- Arc::MCCInterface ∗ **Next** (const std::string &label="")

Protected Attributes

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

Static Protected Attributes

- static Arc::Logger logger

5.26.1 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.

5.26.2 Constructor & Destructor Documentation

5.26.2.1 Arc::MCC::MCC (Arc::Config ∗cfg __*attribute*__((unused))) `[inline]`

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

5.26.3 Member Function Documentation

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

SecHandler

5.26.3.2 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 set NULL corresponding link is removed.

Reimplemented in Arc::Plexer, and Arc::MCC_TLS_Client.

5.26.3.3 virtual Arc::MCC_Status Arc::MCC::process (Arc::Message &request __*attribute*__((unused)), Arc::Message &response __*attribute*__((unused))) [inline, virtual]

Dummy Message processing method. Just a placeholder.

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

Removing all links. Useful for destroying chains.

5.26.4 Member Data Documentation

5.26.4.1 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, Arc::MCC_HTTP, Arc::MCC_SOAP, Arc::MCC_TCP, and Arc::MCC_-TLS.

5.26.4.2 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.

5.26.4.3 std::map<std::string,std::list<Arc::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

5.27 mcc_descriptor Struct Reference

`#include <MCCLoader.h>`

Public Attributes

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

5.27.1 Detailed Description

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:

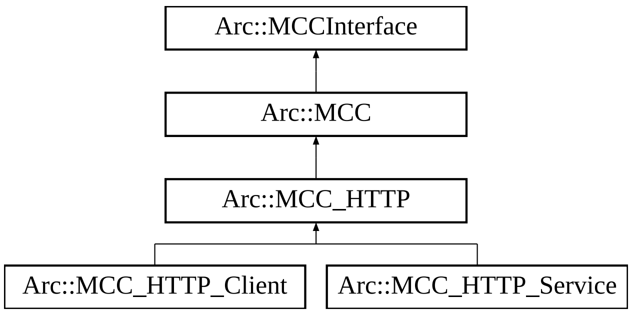
- MCCLoader.h

5.28 Arc::MCC_HTTP Class Reference

A base class for HTTP client and service MCCs.

`#include <MCCHTTP.h>`

Inheritance diagram for Arc::MCC_HTTP::



Public Member Functions

- **MCC_HTTP** (Arc::Config ∗cfg)

Static Protected Attributes

- static Arc::Logger logger

5.28.1 Detailed Description

A base class for HTTP client and service MCCs.

This is a base class for HTTP client and service MCCs. It provides some common functionality for them, i.e. so far only a logger.

5.28.2 Member Data Documentation

5.28.2.1 Arc::Logger Arc::MCC_HTTP::logger `[static, protected]`

A logger for MCCs.

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

Reimplemented from Arc::MCC.

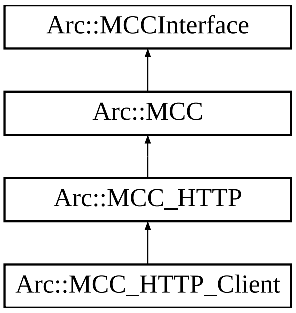
The documentation for this class was generated from the following file:

- MCCHTTP.h

5.29 Arc::MCC_HTTP_Client Class Reference

```
#include <MCCHTTP.h>
```

Inheritance diagram for Arc::MCC_HTTP_Client::



Public Member Functions

- **MCC_HTTP_Client** (Arc::Config ∗cfg)
- virtual MCC_Status process (Message &, Message &)

Protected Attributes

- std::string **method_**
- std::string **endpoint_**

5.29.1 Detailed Description

This class is a client part of HTTP MCC. It accepts PayloadRawInterface payload and uses it as body to generate HTTP request. Request is passed to next MCC as PayloadRawInterface type of payload. Returned PayloadStreamInterface payload is parsed into HTTP respinse and it's body is passed back to calling MCC.

5.29.2 Member Function Documentation

5.29.2.1 virtual MCC_Status Arc::MCC_HTTP_Client::process (Message &, Message &) `[virtual]`

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

Parameters:

request The request that needs to be processed.

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

Returns:

An object representing the status of the call.

Implements Arc::MCCInterface.

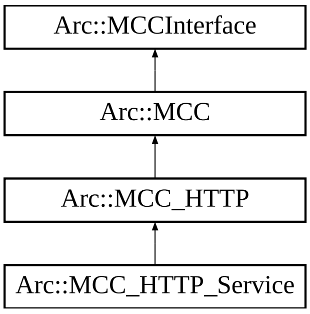
The documentation for this class was generated from the following file:

- MCCHTTP.h

5.30 Arc::MCC_HTTP_Service Class Reference

```
#include <MCCHTTP.h>
```

Inheritance diagram for Arc::MCC_HTTP_Service::



Public Member Functions

- **MCC_HTTP_Service** (Arc::Config ∗cfg)
- virtual MCC_Status process (Message &, Message &)

5.30.1 Detailed Description

This class implements MCC to processes HTTP request. On input payload with PayloadStreamInterface is expected. HTTP message is read from stream ans it's body is converted into PayloadRaw and passed next MCC. Returned payload of PayloadRawInterface type is treated as body part of returning PayloadHTTP. Generated HTTP response is sent though stream passed in input payload.

5.30.2 Member Function Documentation

5.30.2.1 virtual MCC_Status Arc::MCC_HTTP_Service::process (Message &, Message &) `[virtual]`

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

Parameters:

request The request that needs to be processed.

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

Returns:

An object representing the status of the call.

Implements Arc::MCCInterface.

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

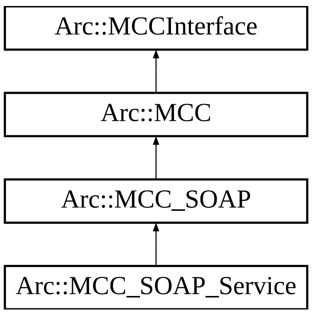
- MCCHTTP.h

5.31 Arc::MCC_SOAP Class Reference

A base class for SOAP client and service MCCs.

`#include <MCCSOAP.h>`

Inheritance diagram for Arc::MCC_SOAP::



Public Member Functions

- **MCC_SOAP** (Arc::Config ∗cfg)

Static Protected Attributes

- static Arc::Logger logger

5.31.1 Detailed Description

A base class for SOAP client and service MCCs.

This is a base class for SOAP client and service MCCs. It provides some common functionality for them, i.e. so far only a logger.

5.31.2 Member Data Documentation

5.31.2.1 Arc::Logger Arc::MCC_SOAP::logger `[static, protected]`

A logger for MCCs.

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

Reimplemented from Arc::MCC.

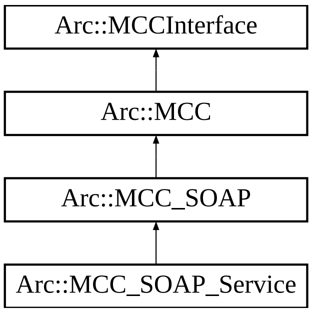
The documentation for this class was generated from the following file:

- MCCSOAP.h

5.32 Arc::MCC_SOAP_Service Class Reference

```
#include <MCCSOAP.h>
```

Inheritance diagram for Arc::MCC_SOAP_Service::



Public Member Functions

- **MCC_SOAP_Service** (Arc::Config ∗cfg)
- virtual MCC_Status process (Message &, Message &)

5.32.1 Detailed Description

This MCC parses SOAP message from input payload. On input payload with PayloadRawInterface is expected. It's converted into PayloadSOAP and passed next MCC. Returned PayloadSOAP is converted into PayloadRaw and returned to calling MCC.

5.32.2 Member Function Documentation

5.32.2.1 virtual MCC_Status Arc::MCC_SOAP_Service::process (Message &, Message &) [virtual]

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

Parameters:

request The request that needs to be processed.

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

Returns:

An object representing the status of the call.

Implements Arc::MCCInterface.

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

- MCCSOAP.h

5.33 Arc::MCC_Status Class Reference

A class for communication of MCC statuses.

`#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

5.33.1 Detailed Description

A class for communication of MCC statuses.

This class is used to communicate status between MCCs. It contains a status kind, a string specifying the origin (MCC) of the status object and an explanation.

5.33.2 Constructor & Destructor Documentation

5.33.2.1 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.")

5.33.3 Member Function Documentation

5.33.3.1 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.

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

Returns the status kind.

Returns the status kind of this object.

Returns:

The status kind of this object.

5.33.3.3 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.

5.33.3.4 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

5.33.3.5 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

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

Conversion to string.

This operator converts a MCC_Status object to a string.

5.33.3.7 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:

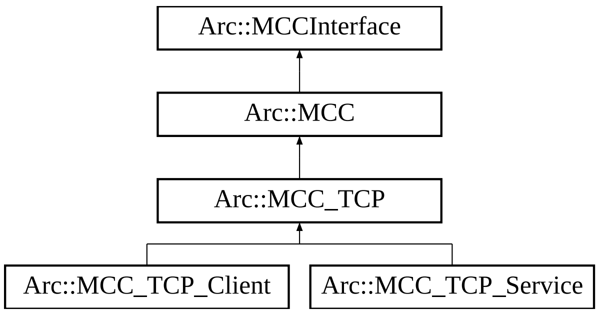
- MCC_Status.h

5.34 Arc::MCC_TCP Class Reference

A base class for TCP client and service MCCs.

`#include <MCCTCP.h>`

Inheritance diagram for Arc::MCC_TCP::



Public Member Functions

- **MCC_TCP** (Arc::Config ∗cfg)

Static Protected Attributes

- static Arc::Logger logger

Friends

- class **PayloadTCPSocket**

5.34.1 Detailed Description

A base class for TCP client and service MCCs.

This is a base class for TCP client and service MCCs. It provides some common functionality for them, i.e. so far only a logger.

5.34.2 Member Data Documentation

5.34.2.1 Arc::Logger Arc::MCC_TCP::logger `[static, protected]`

A logger for MCCs.

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

Reimplemented from Arc::MCC.

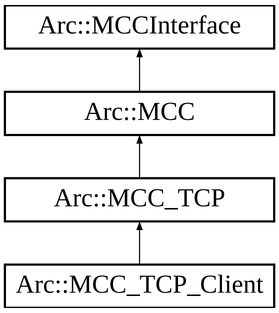
The documentation for this class was generated from the following file:

- MCCTCP.h

5.35 Arc::MCC_TCP_Client Class Reference

`#include <MCCTCP.h>`

Inheritance diagram for Arc::MCC_TCP_Client::



Public Member Functions

- **MCC_TCP_Client** (Arc::Config ∗cfg)
- virtual MCC_Status process (Message &, Message &)

5.35.1 Detailed Description

This class is MCC implementing TCP client. Upon creation it connects to specified TCP post at specified host. process() method ccepts PayloadRawInterface type of payload. Specified payload is sent over TCP socket. It returns PayloadStreamInterface payload for previous MCC to read response.

5.35.2 Member Function Documentation

5.35.2.1 virtual MCC_Status Arc::MCC_TCP_Client::process (Message &, Message &) `[virtual]`

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

Parameters:

request	The request that needs to be processed.
response	A Message object that will contain the response of the request when the method returns.

Returns:

An object representing the status of the call.

Implements Arc::MCCInterface.

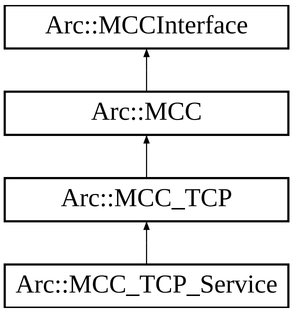
The documentation for this class was generated from the following file:

- MCCTCP.h

5.36 Arc::MCC_TCP_Service Class Reference

```
#include <MCCTCP.h>
```

Inheritance diagram for Arc::MCC_TCP_Service::



Public Member Functions

- MCC_TCP_Service (Arc::Config ∗cfg)
- virtual MCC_Status process (Message &, Message &)

Friends

- class **mcc_tcp_exec_t**

Classes

- class **mcc_tcp_exec_t**

5.36.1 Detailed Description

This class is MCC implementing TCP server. Upon creation this object binds to specified TCP ports and listens for incoming TCP connections on dedicated thread. Each connection is accepted and dedicated thread is created. Then that thread is used to call process() method of next MCC in chain. That method is passed payload implementing PayloadStreamInterface. On response payload with PayloadRawInterface is expected. Alternatively called MCC may use provided PayloadStreamInterface to send it's response back directly.

5.36.2 Constructor & Destructor Documentation

5.36.2.1 Arc::MCC_TCP_Service::MCC_TCP_Service (Arc::Config ∗ *cfg*)

executing function for connection thread

5.36.3 Member Function Documentation

5.36.3.1 virtual MCC_Status Arc::MCC_TCP_Service::process (Message &, Message &) `[virtual]`

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

Parameters:

request The request that needs to be processed.

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

Returns:

An object representing the status of the call.

Implements Arc::MCCInterface.

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

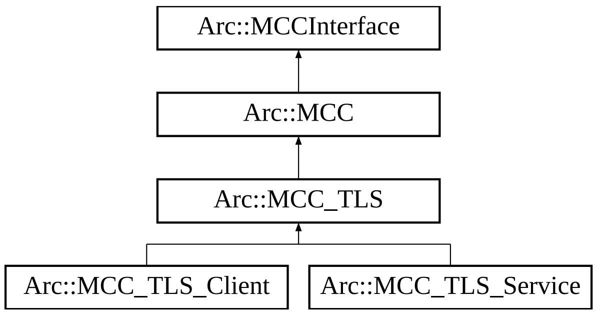
- MCCTCP.h

5.37 Arc::MCC_TLS Class Reference

A base class for SOAP client and service MCCs.

`#include <MCCTLS.h>`

Inheritance diagram for Arc::MCC_TLS::



Public Member Functions

- **MCC_TLS** (Arc::Config ∗cfg)

Protected Member Functions

- bool **tls_random_seed** (std::string filename, long n)
- bool **tls_load_certificate** (SSL_CTX ∗sslctx, const std::string &cert_file, const std::string &key_file, const std::string &password, const std::string &random_file)
- bool **do_ssl_init** (void)

Static Protected Attributes

- static Arc::Logger logger

5.37.1 Detailed Description

A base class for SOAP client and service MCCs.

This is a base class for SOAP client and service MCCs. It provides some common functionality for them, i.e. so far only a logger.

5.37.2 Member Data Documentation

5.37.2.1 Arc::Logger Arc::MCC_TLS::logger `[static, protected]`

A logger for MCCs.

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

Reimplemented from Arc::MCC.

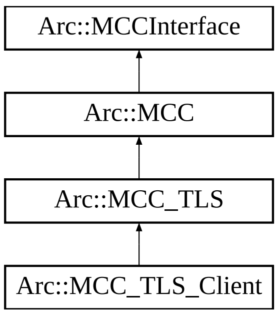
The documentation for this class was generated from the following file:

- MCCTLS.h

5.38 Arc::MCC_TLS_Client Class Reference

```
#include <MCCTLS.h>
```

Inheritance diagram for Arc::MCC_TLS_Client::



Public Member Functions

- **MCC_TLS_Client** (Arc::Config ∗cfg)
- virtual MCC_Status process (Message &, Message &)
- virtual void Next (MCCInterface ∗next, const std::string &label="")

5.38.1 Detailed Description

This class is MCC implementing TLS client. Unfortunately, the MCC_TLS_Client would be put behind MCC_TCP_Client, which looks different with server side(MCC_TLS_Server is put between MCC_-HTTP_Server and MCC_TCP_Server). the MCC_TLS_Client should get the socket fd and attache (I use "attache" :)) ssl to the fd, and the socket fd is created in MCC_TCP_client and also used as s_.Put() in MCC_TCP_client to flush tcp request, the ssl attachement should be done before s_.Put() call. So I just put MCC_TLS_Client behind MCC_TCP_Client. Also there PayloadTLSStream that implement Payload-StreamInterface, which is specified for TLS method, such like "SSL_read()" "SSL_write". As Alexsandr's advice, we could replace TCP with TLS, it will be considered and done later.

5.38.2 Member Function Documentation

5.38.2.1 virtual void Arc::MCC_TLS_Client::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.

5.38.2.2 virtual MCC_Status Arc::MCC_TLS_Client::process (Message &, Message &) [virtual]

Method for processing of requests and responses. This method is called by preceeding MCC in chain when a request needs to be processed. This method must call similar method of next MCC in chain unless any failure happens. Result returned by call to next MCC should be processed and passed back to previous

MCC. In case of failure this method is expected to generate valid error response and return it back to previous MCC without calling the next one.

Parameters:

request The request that needs to be processed.

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

Returns:

An object representing the status of the call.

Implements Arc::MCCInterface.

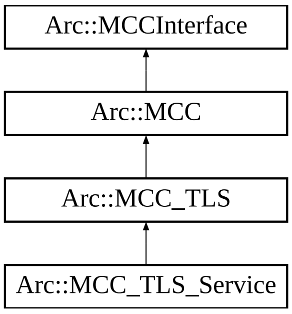
The documentation for this class was generated from the following file:

- MCCTLS.h

5.39 Arc::MCC_TLS_Service Class Reference

`#include <MCCTLS.h>`

Inheritance diagram for Arc::MCC_TLS_Service::



Public Member Functions

- **MCC_TLS_Service** (Arc::Config ∗cfg)
- virtual MCC_Status process (Message &, Message &)

5.39.1 Detailed Description

This two classed are MCCs implementing TLS functionality. Upon creation this object creats SSL_CTX object and configures SSL_CTX object with some environment information about credential. Because we cannot know the "socket" when the creation of MCC_TLS_Service/MCC_TLS_Client object (not like MCC_TCP_Client, which can creat socket in the constructor method by using information in configuration file), we can only creat "ssl" object which is binded to specified "socket", when MCC_HTTP_Client calls the process() method of MCC_TLS_Client object, or MCC_TCP_Service calls the process() method of MCC_TLS_Service object. The "ssl" object is embeded in a payload called PayloadTLSSocket.

The process() method of MCC_TLS_Service is passed payload implementing PayloadStreamInterface (actually PayloadTCPSocket), and the method return empty payload in "outmsg", just as MCC_HTTP_Service does. The ssl object is created and binded to socket object when constructing the PayloadTLSSocket in the process() method.

The process() method of MCC_TLS_Client is also passed payload impementing PayloadStreamInterface and return empty payload.

So the MCC_TLS_Service and MCC_TLS_Client will only keep the imformation about SSL_CTX, nothing else. It is the PayloadTLSSocket some keeps some information about ssl session. And the PayloadTLSSocket which implements the PayloadStreamInterface will be used by PayloadHTTP.

5.39.2 Member Function Documentation

5.39.2.1 virtual MCC_Status Arc::MCC_TLS_Service::process (Message &, Message &) `[virtual]`

Method for processing of requests and responses. This method is called by preceeding MCC in chain when a request needs to be processed. This method must call similar method of next MCC in chain unless any failure happens. Result returned by call to next MCC should be processed and passed back to previous

MCC. In case of failure this method is expected to generate valid error response and return it back to previous MCC without calling the next one.

Parameters:

request The request that needs to be processed.

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

Returns:

An object representing the status of the call.

Implements Arc::MCCInterface.

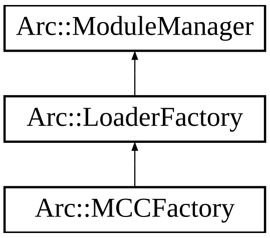
The documentation for this class was generated from the following file:

- MCCTLS.h

5.40 Arc::MCCFactory Class Reference

```
#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)

5.40.1 Detailed Description

This class handles shared libraries containing MCCs

5.40.2 Constructor & Destructor Documentation

5.40.2.1 Arc::MCCFactory::MCCFactory (Config ∗ *cfg*)

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

5.40.3 Member Function Documentation

5.40.3.1 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.

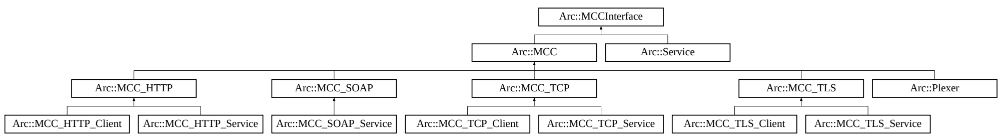
The documentation for this class was generated from the following file:

- MCCFactory.h

5.41 Arc::MCCInterface Class Reference

`#include <MCC.h>`

Inheritance diagram for Arc::MCCInterface::



Public Member Functions

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

5.41.1 Detailed Description

This class defines interface for communication between MCC, Service and Plexer objects. Interface is made of method process() which is called by previous MCC in chain. For memory management policies please read description of Message class.

5.41.2 Member Function Documentation

5.41.2.1 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, Arc::MCC_HTTP_Service, Arc::MCC_HTTP_Client, Arc::MCC_SOAP_-Service, Arc::MCC_TCP_Service, Arc::MCC_TCP_Client, Arc::MCC_TLS_Service, and Arc::MCC_-TLS_Client.

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

- MCC.h

5.42 Arc::Message Class Reference

```
#include <Message.h>
```

Public Member Functions

- Message (void)
- Message (Message &msg)
- ∼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)

5.42.1 Detailed Description

Message is passed through chain of MCCs. It refers to objects with main content (payload), authentication/authorization information and common purpose attributes. Message class does not manage pointers to objects and theur content. it only serves for grouping those objects. Message objects are supposed to be processed by objects' 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 mus tnot 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.

5.42.2 Constructor & Destructor Documentation

5.42.2.1 Arc::Message::Message (void) `[inline]`

Dummy constructor

5.42.2.2 Arc::Message::Message (Message & *msg*) `[inline]`

Copy constructor. Ensures shallow copy.

5.42.2.3 Arc::Message::∼Message (void) `[inline]`

Destructor does not affect refered objects

5.42.3 Member Function Documentation

5.42.3.1 MessageAttributes∗ Arc::Message::Attributes (void) `[inline]`

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

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

Assignment. Ensures shallow copy.

5.42.3.3 MessagePayload∗ Arc::Message::Payload (MessagePayload ∗ *new_payload*) `[inline]`

Replace payload with new one

5.42.3.4 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

5.43 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

Protected Attributes

- AttrMap attributes_

5.43.1 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.

5.43.2 Constructor & Destructor Documentation

5.43.2.1 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.

5.43.3 Member Function Documentation

5.43.3.1 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.

5.43.3.2 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.

5.43.3.3 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.

5.43.3.4 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.

5.43.3.5 void Arc::MessageAttributes::remove (const std::string & *key*, const std::string & *value*)

Removes one value of an attribute.

This method removes a certain value from the attribute that matches a certain key.

Parameters:

key The key of the attribute from which the value shall be removed.

value The value to remove.

5.43.3.6 void Arc::MessageAttributes::removeAll (const std::string & *key*)

Removes all attributes with a certain key.

This method removes all attributes that match a certain key.

Parameters:

key The key of the attributes to remove.

5.43.3.7 void Arc::MessageAttributes::set (const std::string & *key*, const std::string & *value*)

Sets a unique value of an attribute.

This method removes any previous value of an attribute and sets the new value as the only value.

Parameters:

key The key of the attribute.

value The (new) value of the attribute.

5.43.4 Member Data Documentation

5.43.4.1 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

5.44 Arc::MessageAuth Class Reference

```
#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)

5.44.1 Detailed Description

Class MessageAuth will contain authencity information, authorization tokens and decisions.

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

- MessageAuth.h

5.45 Arc::MessageContext Class Reference

```
#include <Message.h>
```

Public Member Functions

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

5.45.1 Detailed Description

Handler for context of message associated to persistent conenction.

5.45.2 Member Function Documentation

5.45.2.1 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

5.46 Arc::MessageContextElement Class Reference

```
#include <Message.h>
```

5.46.1 Detailed Description

Just a top class for elements contained in context - needed for destruction to work.

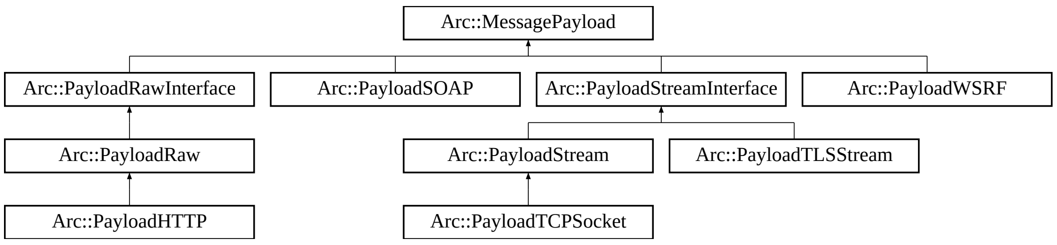
The documentation for this class was generated from the following file:

- Message.h

5.47 Arc::MessagePayload Class Reference

```
#include <Message.h>
```

Inheritance diagram for Arc::MessagePayload::



5.47.1 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.

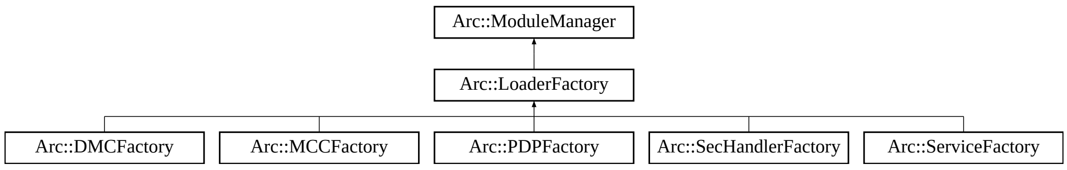
The documentation for this class was generated from the following file:

- Message.h

5.48 Arc::ModuleManager Class Reference

```
#include <ModuleManager.h>
```

Inheritance diagram for Arc::ModuleManager::



Public Member Functions

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

5.48.1 Detailed Description

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.

5.48.2 Constructor & Destructor Documentation

5.48.2.1 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.

5.48.3 Member Function Documentation

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

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

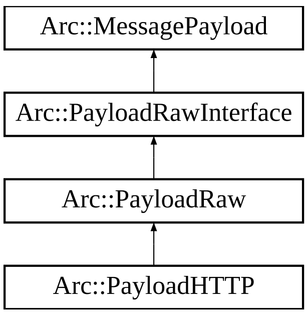
The documentation for this class was generated from the following file:

- ModuleManager.h

5.49 Arc::PayloadHTTP Class Reference

```
#include <PayloadHTTP.h>
```

Inheritance diagram for Arc::PayloadHTTP::



Public Member Functions

- PayloadHTTP (PayloadStreamInterface &stream)
- PayloadHTTP (const std::string &method, const std::string &url, PayloadStreamInterface &stream)
- PayloadHTTP (int code, const std::string &reason, PayloadStreamInterface &stream)
- PayloadHTTP (const std::string &method, const std::string &url)
- PayloadHTTP (int code, const std::string &reason)
- virtual **operator bool** (void)
- virtual bool **operator!** (void)
- virtual const std::string & Attribute (const std::string &name)
- virtual const std::map< std::string, std::string > & Attributes (void)
- virtual void Attribute (const std::string &name, const std::string &value)
- virtual bool Flush (void)
- virtual std::string **Method** ()
- virtual std::string **Endpoint** ()
- virtual std::string **Reason** ()
- virtual int **Code** ()

Protected Member Functions

- bool readline (std::string &line)
- bool read (char ∗buf, int &size)
- bool parse_header (void)
- bool get_body (void)

Protected Attributes

- bool **valid_**
- PayloadStreamInterface & **stream_**
- std::string uri_
- int version_major_
- int version_minor_

- std::string method_
- int code_
- std::string reason_
- int length_
- bool chunked_
- std::map< std::string, std::string > attributes_
- char **tbuf_** [1024]
- int **tbuflen_**

5.49.1 Detailed Description

This class implements parsing and generation of HTTP messages. It implements only subset of HTTP/1.1 and also provides an PayloadRawInterface for including as payload into Message passed through MCC chains.

5.49.2 Constructor & Destructor Documentation

5.49.2.1 Arc::PayloadHTTP::PayloadHTTP (PayloadStreamInterface & *stream*)

Constructor - creates object by parsing HTTP request or response from stream. Supplied stream is associated with object for later use.

5.49.2.2 Arc::PayloadHTTP::PayloadHTTP (const std::string & *method*, const std::string & *url*, PayloadStreamInterface & *stream*)

Constructor - creates HTTP request to be sent through stream. HTTP message is not sent yet.

5.49.2.3 Arc::PayloadHTTP::PayloadHTTP (int *code*, const std::string & *reason*, PayloadStreamInterface & *stream*)

Constructor - creates HTTP response to be sent through stream. HTTP message is not sent yet.

5.49.2.4 Arc::PayloadHTTP::PayloadHTTP (const std::string & *method*, const std::string & *url*)

Constructor - creates HTTP request to be rendered through Raw interface.

5.49.2.5 Arc::PayloadHTTP::PayloadHTTP (int *code*, const std::string & *reason*)

Constructor - creates HTTP response to be rendered through Raw interface.

5.49.3 Member Function Documentation

5.49.3.1 virtual void Arc::PayloadHTTP::Attribute (const std::string & *name*, const std::string & *value*) `[virtual]`

Sets HTTP header attribute 'name' to 'value'

5.49.3.2 virtual const std::string& Arc::PayloadHTTP::Attribute (const std::string & *name*) [virtual]

Returns HTTP header attribute with specified name. Empty string if no such attribute.

5.49.3.3 virtual const std::map<std::string,std::string>& Arc::PayloadHTTP::Attributes (void) [virtual]

Returns HTTP all header attributes.

5.49.3.4 virtual bool Arc::PayloadHTTP::Flush (void) [virtual]

Send created object through associated stream. If there is no stream associated then HTTP specific data is inserted into Raw buffers of this object.

5.49.3.5 bool Arc::PayloadHTTP::get_body (void) [protected]

Read Body of HTTP message and attach it to inherited PayloadRaw object

5.49.3.6 bool Arc::PayloadHTTP::parse_header (void) [protected]

Read HTTP header and fill internal variables

5.49.3.7 bool Arc::PayloadHTTP::read (char * *buf*, int & *size*) [protected]

Read up to 'size' bytes from stream_

5.49.3.8 bool Arc::PayloadHTTP::readline (std::string & *line*) [protected]

Read from stream till

5.49.4 Member Data Documentation

5.49.4.1 std::map<std::string,std::string> Arc::PayloadHTTP::attributes_ [protected]

true if content is chunked

5.49.4.2 bool Arc::PayloadHTTP::chunked_ [protected]

Content-length of HTTP message

5.49.4.3 int Arc::PayloadHTTP::code_ [protected]

HTTP method being used or requested

5.49.4.4 int Arc::PayloadHTTP::length_ `[protected]`

HTTP reason being sent or supplied

5.49.4.5 std::string Arc::PayloadHTTP::method_ `[protected]`

minor number of HTTP version - must be 0 or 1

5.49.4.6 std::string Arc::PayloadHTTP::reason_ `[protected]`

HTTP code being sent or supplied

5.49.4.7 std::string Arc::PayloadHTTP::uri_ `[protected]`

stream used to comminicate to outside

5.49.4.8 int Arc::PayloadHTTP::version_major_ `[protected]`

URI being contacted

5.49.4.9 int Arc::PayloadHTTP::version_minor_ `[protected]`

major number of HTTP version - must be 1

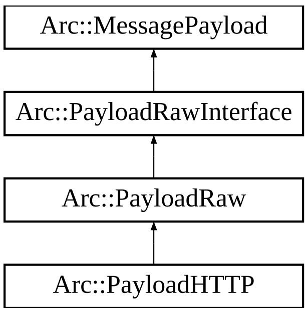
The documentation for this class was generated from the following file:

- PayloadHTTP.h

5.50 Arc::PayloadRaw Class Reference

`#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

Protected Attributes

- std::vector< PayloadRawBuf > **buf_**

5.50.1 Detailed Description

Implementation of PayloadRawInterface - raw byte multi-buffer.

5.50.2 Constructor & Destructor Documentation

5.50.2.1 Arc::PayloadRaw::PayloadRaw (void) `[inline]`

Constructor. Created object contains no buffers.

5.50.2.2 virtual Arc::PayloadRaw::∼PayloadRaw (void) `[virtual]`

Destructor. Frees allocated buffers.

5.50.3 Member Function Documentation

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

Returns pointer to num'th buffer

Implements Arc::PayloadRawInterface.

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

Returns length of num'th buffer

Implements Arc::PayloadRawInterface.

5.50.3.3 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.

5.50.3.4 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 aontent at 's' is expected to be null-terminated.

Implements Arc::PayloadRawInterface.

5.50.3.5 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.

5.50.3.6]

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.

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

Returns cumulative length of all buffers

Implements Arc::PayloadRawInterface.

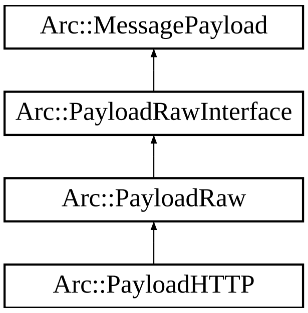
The documentation for this class was generated from the following file:

- PayloadRaw.h

5.51 Arc::PayloadRawInterface Class Reference

`#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

5.51.1 Detailed Description

Virtual interface for managing arbitrarily accessible Message payload. This class implements memory-resident or memory-mapped content made of optionally multiple chunks/buffers. This calss is purely virtual.

5.51.2 Member Function Documentation

5.51.2.1 virtual char∗ Arc::PayloadRawInterface::Buffer (unsigned int *num*) `[pure virtual]`

Returns pointer to num'th buffer

Implemented in Arc::PayloadRaw.

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

Returns length of num'th buffer

Implemented in Arc::PayloadRaw.

5.51.2.3 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.

5.51.2.4 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 aontent at 's' is expected to be null-terminated.

Implemented in Arc::PayloadRaw.

5.51.2.5 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.

5.51.2.6]

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.

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

Returns cumulative length of all buffers

Implemented in Arc::PayloadRaw.

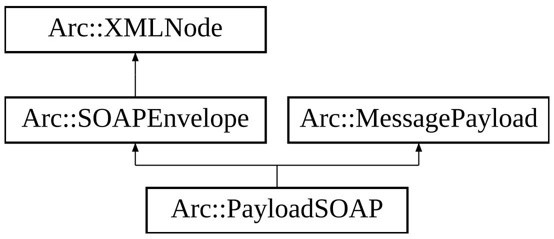
The documentation for this class was generated from the following file:

- PayloadRaw.h

5.52 Arc::PayloadSOAP Class Reference

`#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)

5.52.1 Detailed Description

This class combines MessagePayload with SOAPEnvelope to make it possible to pass SOAP messages through MCC chain

5.52.2 Constructor & Destructor Documentation

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

Constructor - creates new Message payload

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

Constructor - creates Message payload from SOAP message. Used SOAP message must exist as long as created object exists.

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

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

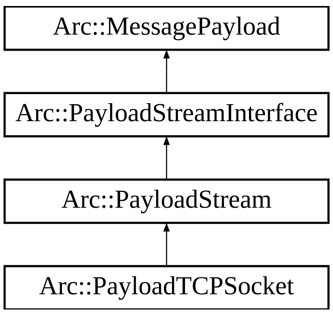
The documentation for this class was generated from the following file:

- PayloadSOAP.h

5.53 Arc::PayloadStream Class Reference

```
#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_

5.53.1 Detailed Description

Implemetation of PayloadStreamInterface for generic POSIX handle.

5.53.2 Constructor & Destructor Documentation

5.53.2.1 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.

5.53.2.2 virtual Arc::PayloadStream::∼PayloadStream (void) `[inline, virtual]`

Destructor.

5.53.3 Member Function Documentation

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

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

Implements Arc::PayloadStreamInterface.

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

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

Implements Arc::PayloadStreamInterface.

5.53.3.3 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.

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

Returns true if stream is valid.

Implements Arc::PayloadStreamInterface.

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

Returns true if stream is invalid.

Implements Arc::PayloadStreamInterface.

5.53.3.6 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.

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

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

Implements Arc::PayloadStreamInterface.

5.53.3.8 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.

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

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

Implements Arc::PayloadStreamInterface.

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

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

Implements Arc::PayloadStreamInterface.

5.53.4 Member Data Documentation

5.53.4.1 int Arc::PayloadStream::handle_ `[protected]`

Timeout for read/write operations

5.53.4.2 bool Arc::PayloadStream::seekable_ `[protected]`

Handle for operations

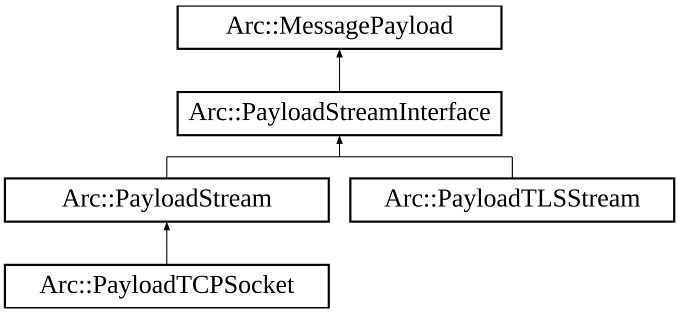
The documentation for this class was generated from the following file:

- PayloadStream.h

5.54 Arc::PayloadStreamInterface Class Reference

`#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

5.54.1 Detailed Description

Virtual interface for managing stream-like source and destination. It's supposed to passed through MCC chain as payload of Message. It must be treated by MCC as dynamic payload. This class is purely virtual.

5.54.2 Member Function Documentation

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

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

Implemented in Arc::PayloadStream, and Arc::PayloadTLSStream.

5.54.2.2 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, and Arc::PayloadTLSStream.

5.54.2.3 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, and Arc::PayloadTLSStream.

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

Returns true if stream is valid.

Implemented in Arc::PayloadStream, and Arc::PayloadTLSStream.

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

Returns true if stream is invalid.

Implemented in Arc::PayloadStream, and Arc::PayloadTLSStream.

5.54.2.6 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, and Arc::PayloadTLSStream.

5.54.2.7 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, and Arc::PayloadTLSStream.

5.54.2.8 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, and Arc::PayloadTLSStream.

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

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

Implemented in Arc::PayloadStream, and Arc::PayloadTLSStream.

5.54.2.10 virtual int Arc::PayloadStreamInterface::Timeout (void) const [pure virtual]

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

Implemented in Arc::PayloadStream, and Arc::PayloadTLSStream.

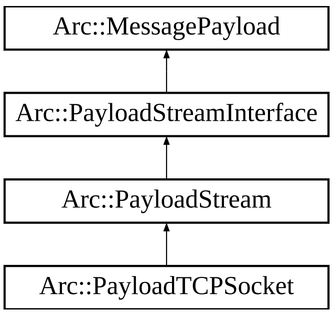
The documentation for this class was generated from the following file:

- PayloadStream.h

5.55 Arc::PayloadTCPSocket Class Reference

```
#include <PayloadTCPSocket.h>
```

Inheritance diagram for Arc::PayloadTCPSocket::



Public Member Functions

- PayloadTCPSocket (const char ∗hostname, int port, Logger &logger)
- PayloadTCPSocket (const std::string endpoint, Logger &logger)
- PayloadTCPSocket (int s, Logger &logger)
- **PayloadTCPSocket** (PayloadTCPSocket &s, Logger &logger)
- **PayloadTCPSocket** (PayloadStream &s, Logger &logger)

5.55.1 Detailed Description

This class extends PayloadStream with TCP socket specific features

5.55.2 Constructor & Destructor Documentation

5.55.2.1 Arc::PayloadTCPSocket::PayloadTCPSocket (const char ∗ *hostname*, int *port*, Logger & *logger*)

Constructor - connects to TCP server at specified hostname:port

5.55.2.2 Arc::PayloadTCPSocket::PayloadTCPSocket (const std::string *endpoint*, Logger & *logger*)

Constructor - connects to TCP server at specified endpoint - hostname:port

5.55.2.3 Arc::PayloadTCPSocket::PayloadTCPSocket (int *s*, Logger & *logger*) `[inline]`

Constructor - creates object of already connected socket

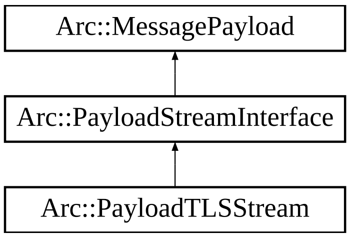
The documentation for this class was generated from the following file:

- PayloadTCPSocket.h

5.56 Arc::PayloadTLSStream Class Reference

`#include <PayloadTLSStream.h>`

Inheritance diagram for Arc::PayloadTLSStream::



Public Member Functions

- PayloadTLSStream (SSL ∗ssl=NULL)
- virtual ∼PayloadTLSStream (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)
- X509 ∗ GetPeercert (void)

Protected Attributes

- int **timeout_**
- SSL ∗ ssl_

5.56.1 Detailed Description

Implemetation of PayloadStreamInterface for "ssl" handle.

5.56.2 Constructor & Destructor Documentation

5.56.2.1 Arc::PayloadTLSStream::PayloadTLSStream (SSL ∗ *ssl* = `NULL`)

Constructor. Attaches to already open handle. Handle is not managed by this class and must be closed by external code.

5.56.2.2 virtual Arc::PayloadTLSStream::∼PayloadTLSStream (void) `[inline, virtual]`

Destructor.

5.56.3 Member Function Documentation

5.56.3.1 virtual std::string Arc::PayloadTLSStream::Get (void) `[inline, virtual]`

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

Implements Arc::PayloadStreamInterface.

5.56.3.2 virtual bool Arc::PayloadTLSStream::Get (std::string & *buf*) `[virtual]`

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

Implements Arc::PayloadStreamInterface.

5.56.3.3 virtual bool Arc::PayloadTLSStream::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.

5.56.3.4 X509∗ Arc::PayloadTLSStream::GetPeercert (void)

Getting peer certificate from the established ssl

5.56.3.5 virtual Arc::PayloadTLSStream::operator bool (void) `[inline, virtual]`

Returns true if stream is valid.

Implements Arc::PayloadStreamInterface.

5.56.3.6 virtual bool Arc::PayloadTLSStream::operator! (void) `[inline, virtual]`

Returns true if stream is invalid.

Implements Arc::PayloadStreamInterface.

5.56.3.7 virtual bool Arc::PayloadTLSStream::Put (const char ∗ *buf*) `[inline, virtual]`

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

Implements Arc::PayloadStreamInterface.

5.56.3.8 virtual bool Arc::PayloadTLSStream::Put (const std::string & *buf*) `[inline, virtual]`

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

Implements Arc::PayloadStreamInterface.

5.56.3.9 virtual bool Arc::PayloadTLSStream::Put (const char ∗ *buf*, int *size*) `[virtual]`

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

Implements Arc::PayloadStreamInterface.

5.56.3.10 virtual void Arc::PayloadTLSStream::Timeout (int *to*) `[inline, virtual]`

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

Implements Arc::PayloadStreamInterface.

5.56.3.11 virtual int Arc::PayloadTLSStream::Timeout (void) const `[inline, virtual]`

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

Implements Arc::PayloadStreamInterface.

5.56.4 Member Data Documentation

5.56.4.1 SSL∗ Arc::PayloadTLSStream::ssl_ `[protected]`

Timeout for read/write operations

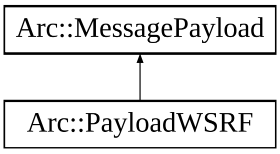
The documentation for this class was generated from the following file:

- PayloadTLSStream.h

5.57 Arc::PayloadWSRF Class Reference

```
#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_**

5.57.1 Detailed Description

This class combines MessagePayload with WSRF to make it possible to pass WSRF messages through MCC chain

5.57.2 Constructor & Destructor Documentation

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

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

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

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

5.57.2.3 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

5.58 pdp_descriptor Struct Reference

```
#include <PDPLoader.h>
```

Public Attributes

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

5.58.1 Detailed Description

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.

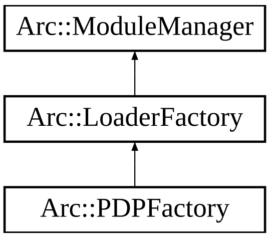
The documentation for this struct was generated from the following file:

- PDPLoader.h

5.59 Arc::PDPFactory Class Reference

```
#include <PDPFactory.h>
```

Inheritance diagram for Arc::PDPFactory::



Public Member Functions

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

5.59.1 Detailed Description

This class handles shared libraries containing PDPs

5.59.2 Constructor & Destructor Documentation

5.59.2.1 Arc::PDPFactory::PDPFactory (Config ∗ *cfg*)

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

5.59.3 Member Function Documentation

5.59.3.1 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:

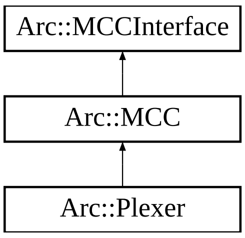
- PDPFactory.h

5.60 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 Protected Attributes

- static Arc::Logger logger

5.60.1 Detailed Description

The Plexer class, used for routing messages to services.

This is the Plexer class. Its purpose is to rout incoming messages to appropriate services.

5.60.2 Constructor & Destructor Documentation

5.60.2.1 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.

5.60.2.2 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.

5.60.3 Member Function Documentation

5.60.3.1 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.

5.60.3.2 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.

Implements Arc::MCCInterface.

5.60.4 Member Data Documentation

5.60.4.1 Arc::Logger Arc::Plexer::logger [static, protected]

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

5.61 Arc::PlexerEntry Class Reference

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

`#include <Plexer.h>`

Friends

- class **Plexer**

5.61.1 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

5.62 Arc::RegularExpression Class Reference

A regular expression class.

`#include <Plexer.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) const
- std::string getPattern ()

5.62.1 Detailed Description

A regular expression class.

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

5.62.2 Constructor & Destructor Documentation

5.62.2.1 Arc::RegularExpression::RegularExpression (std::string *pattern*)

Creates a reges from a pattern string.

5.62.2.2 Arc::RegularExpression::RegularExpression (const RegularExpression & *regex*)

Copy constructor.

5.62.2.3 Arc::RegularExpression::∼RegularExpression ()

Destructor.

5.62.3 Member Function Documentation

5.62.3.1 std::string Arc::RegularExpression::getPattern ()

Returns patter.

5.62.3.2 bool Arc::RegularExpression::hasPattern (std::string *str*)

Returns true if this regex has the pattern provided.

5.62.3.3 bool Arc::RegularExpression::isOk ()

Returns true if the pattern of this regex is ok.

5.62.3.4 bool Arc::RegularExpression::match (const std::string & *str*, std::list< std::string > & *unmatched*) const

Returns true if this regex matches the string provided. Unmatched parts of the string are stored in 'unmatched'.

5.62.3.5 bool Arc::RegularExpression::match (const std::string & *str*) const

Returns true if this regex matches whole string provided.

5.62.3.6 const RegularExpression& Arc::RegularExpression::operator= (const RegularExpression & *regex*)

Assignment operator.

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

- Plexer.h

5.63 sechandler_descriptor Struct Reference

`#include <SecHandlerLoader.h>`

Public Attributes

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

5.63.1 Detailed Description

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.

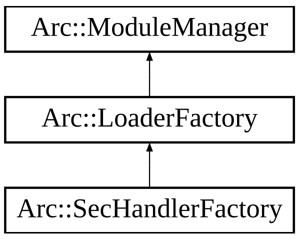
The documentation for this struct was generated from the following file:

- SecHandlerLoader.h

5.64 Arc::SecHandlerFactory Class Reference

```
#include <SecHandlerFactory.h>
```

Inheritance diagram for Arc::SecHandlerFactory::



Public Member Functions

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

5.64.1 Detailed Description

This class handles shared libraries containing SecHandlers

5.64.2 Constructor & Destructor Documentation

5.64.2.1 Arc::SecHandlerFactory::SecHandlerFactory (Config ∗ *cfg*)

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

5.64.3 Member Function Documentation

5.64.3.1 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

5.65 Arc::Security Class Reference

Common stuff used by security related slasses.

```
#include <Security.h>
```

Friends

- class **SecHandler**
- class **PDP**

5.65.1 Detailed Description

Common stuff used by security related slasses.

This class is just a place where to put common stuff that is used by security related slasses. So far it only contains a logger.

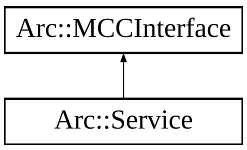
The documentation for this class was generated from the following file:

- Security.h

5.66 Arc::Service Class Reference

`#include <Service.h>`

Inheritance diagram for Arc::Service::



Public Member Functions

- Service (Arc::Config ∗cfg __attribute__((unused)))
- virtual void AddSecHandler (Arc::Config ∗cfg, Arc::SecHandler ∗sechandler, const std::string &label="")

Protected Attributes

- std::map< std::string, std::list< Arc::SecHandler ∗ > > sechandlers_

Static Protected Attributes

- static Logger **logger**

5.66.1 Detailed Description

Service - last plugin 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 accespt 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 . The way to write client couterpart of corresponding service is undefined. For example see /src/tests/echo/test.cpp .

5.66.2 Constructor & Destructor Documentation

5.66.2.1 Arc::Service::Service (Arc::Config ∗cfg ___attribute___((unused))) `[inline]`

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

5.66.3 Member Function Documentation

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

SecHandler

5.66.4 Member Data Documentation

5.66.4.1 std::map<std::string,std::list<Arc::SecHandler*> > Arc::Service::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:

- Service.h

5.67 service_descriptor Struct Reference

`#include <ServiceLoader.h>`

Public Attributes

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

5.67.1 Detailed Description

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.

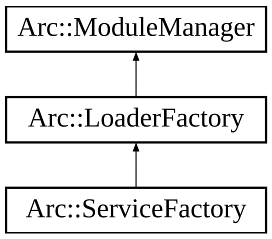
The documentation for this struct was generated from the following file:

- ServiceLoader.h

5.68 Arc::ServiceFactory Class Reference

```
#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)

5.68.1 Detailed Description

This class handles shared libraries containing Services

5.68.2 Constructor & Destructor Documentation

5.68.2.1 Arc::ServiceFactory::ServiceFactory (Config ∗ *cfg*)

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

5.68.3 Member Function Documentation

5.68.3.1 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.

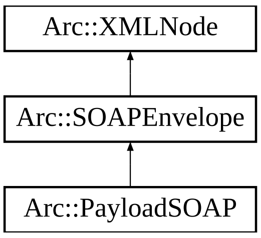
The documentation for this class was generated from the following file:

- ServiceFactory.h

5.69 Arc::SOAPEnvelope Class Reference

`#include <SOAPEnvelope.h>`

Inheritance diagram for Arc::SOAPEnvelope::



Public Member Functions

- SOAPEnvelope (const std::string &xml)
- SOAPEnvelope (const char ∗xml, int len=-1)
- SOAPEnvelope (const NS &ns, bool fault=false)
- SOAPEnvelope (XMLNode doc)
- SOAPEnvelope ∗ New (void)
- void Namespaces (const NS &namespaces)
- void GetXML (std::string &xml) const
- XMLNode **Header** (void)
- bool **IsFault** (void)
- SOAPFault ∗ **Fault** (void)

5.69.1 Detailed Description

SOAPEnvelope extends XMLNode class to support structures of SOAP message. All XMLNode methods are exposed with top node translated to Envelope part of SOAP.

5.69.2 Constructor & Destructor Documentation

5.69.2.1 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.

5.69.2.2 Arc::SOAPEnvelope::SOAPEnvelope (const char ∗ *xml*, int *len* = `-1`)

Same as previous

5.69.2.3 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.

5.69.2.4 Arc::SOAPEnvelope::SOAPEnvelope (XMLNode *doc*)

Acquire XML document as SOAP message. Created XML structure is NOT owned by this instance.

5.69.3 Member Function Documentation

5.69.3.1 void Arc::SOAPEnvelope::GetXML (std::string & *xml*) const

Fills argument with this instance XML (sub)tree textual representation

Reimplemented from Arc::XMLNode.

5.69.3.2 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.

5.69.3.3 SOAPEnvelope∗ Arc::SOAPEnvelope::New (void)

Creates complete copy of SOAP. Do not use New() method of XMLNode - use this one.

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

- SOAPEnvelope.h

5.70 Arc::SOAPFault Class Reference

```
#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**

5.70.1 Detailed Description

SOAPFault provides an interface to conveinet access to 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 under control of XMLNode or SOAPEnvelope classes. If instance does not refer to valid SOAP Fault structure all manipulation methods will have no effect.

5.70.2 Member Enumeration Documentation

5.70.2.1 enum Arc::SOAPFault::SOAPFaultCode

Fault codes of SOAP specs

5.70.3 Constructor & Destructor Documentation

5.70.3.1 Arc::SOAPFault::SOAPFault (XMLNode & *body*)

Parse Fault elements of SOAP Body or any other XML tree with Fault element

5.70.4 Member Function Documentation

5.70.4.1 void Arc::SOAPFault::Code (SOAPFaultCode *code*)

Set Fault Code element

5.70.4.2 SOAPFaultCode Arc::SOAPFault::Code (void)

Returns Fault Code element

5.70.4.3 XMLNode Arc::SOAPFault::Detail (bool *create* = `false`)

Access Fault Detail element. If create is set to true this element is creted if not present.

5.70.4.4 void Arc::SOAPFault::Node (const char $*$ *node*)

Set content of Fault Node element to 'node'

5.70.4.5 std::string Arc::SOAPFault::Node (void)

Returns content of Fault Node element

5.70.4.6 Arc::SOAPFault::operator bool (void) `[inline]`

Returns true if instance refers to SOAP Fault

5.70.4.7 void Arc::SOAPFault::Reason (const char $*$ *reason*) `[inline]`

Set Fault Reason element at top level

5.70.4.8 void Arc::SOAPFault::Reason (int *num*, const char $*$ *reason*)

Set Fault Reason content at various levels to 'reason'

5.70.4.9 std::string Arc::SOAPFault::Reason (int *num* = 0)

Returns content of Fault Reason element at various levels

5.70.4.10 void Arc::SOAPFault::Role (const char ∗ *role*)

Set content of Fault Role element to 'role'

5.70.4.11 std::string Arc::SOAPFault::Role (void)

Returns content of Fault Role element

5.70.4.12 void Arc::SOAPFault::Subcode (int *level*, const char ∗ *subcode*)

Set Fault Subcode element at various levels (0 is for Code) to 'subcode'

5.70.4.13 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

5.71 Arc::SOAPMessage Class Reference

```
#include <SOAPMessage.h>
```

Public Member Functions

- SOAPMessage (void)
- SOAPMessage (long msg_ptr_addr)
- SOAPMessage (SOAPMessage &msg)
- SOAPMessage (Arc::Message &msg)
- ∼SOAPMessage (void)
- SOAPMessage & operator= (SOAPMessage &msg)
- Arc::PayloadSOAP ∗ Payload (void)
- Arc::PayloadSOAP ∗ Payload (Arc::PayloadSOAP ∗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)

5.71.1 Detailed Description

Message is passed through chain of MCCs works like the Message but use only SOAP message

5.71.2 Constructor & Destructor Documentation

5.71.2.1 Arc::SOAPMessage::SOAPMessage (void) `[inline]`

Dummy constructor

5.71.2.2 Arc::SOAPMessage::SOAPMessage (long *msg_ptr_addr*)

Copy constructor. Used by language bindigs

5.71.2.3 Arc::SOAPMessage::SOAPMessage (SOAPMessage & *msg*) `[inline]`

Copy constructor. Ensures shallow copy.

5.71.2.4 Arc::SOAPMessage::SOAPMessage (Arc::Message & *msg*)

Copy constructor. Ensures shallow copy.

5.71.2.5 Arc::SOAPMessage::∼SOAPMessage (void) `[inline]`

Destructor does not affect refered objects

5.71.3 Member Function Documentation

5.71.3.1 Arc::MessageAttributes∗ Arc::SOAPMessage::Attributes (void) `[inline]`

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

5.71.3.2 SOAPMessage& Arc::SOAPMessage::operator= (SOAPMessage & *msg*) `[inline]`

Assignment. Ensures shallow copy.

5.71.3.3 Arc::PayloadSOAP∗ Arc::SOAPMessage::Payload (Arc::PayloadSOAP ∗ *new_payload*) `[inline]`

Replace payload with new one

5.71.3.4 Arc::PayloadSOAP∗ Arc::SOAPMessage::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:

- SOAPMessage.h

5.72 Arc::Time Class Reference

```
#include <DateTime.h>
```

Public Member Functions

- Time ()
- Time (const time_t &)
- Time (const std::string &)
- Time & operator= (const time_t &)
- 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

Static Public Member Functions

- static void SetFormat (const TimeFormat &)
- static TimeFormat GetFormat ()

5.72.1 Detailed Description

A class for storing and manipulating times.

5.72.2 Constructor & Destructor Documentation

5.72.2.1 Arc::Time::Time ()

Default constructor. The time is put equal the current time.

5.72.2.2 Arc::Time::Time (const time_t &)

Constructor that takes a time_t variable and stores it.

5.72.2.3 Arc::Time::Time (const std::string &)

Constructor that tries to convert a string into a time_t.

5.72.3 Member Function Documentation

5.72.3.1 static TimeFormat Arc::Time::GetFormat () [static]

Gets the default format for time strings.

5.72.3.2 time_t Arc::Time::GetTime () const

gets the time

5.72.3.3 Arc::Time::operator std::string () const

Returns a string representation of the time, using the default format.

5.72.3.4 bool Arc::Time::operator!= (const Time &) const

Comparing two Time objects.

5.72.3.5 bool Arc::Time::operator< (const Time &) const

Comparing two Time objects.

5.72.3.6 bool Arc::Time::operator<= (const Time &) const

Comparing two Time objects.

5.72.3.7 Time& Arc::Time::operator= (const time_t &)

Assignment operator from a time_t.

5.72.3.8 bool Arc::Time::operator== (const Time &) const

Comparing two Time objects.

5.72.3.9 bool Arc::Time::operator> (const Time &) const

Comparing two Time objects.

5.72.3.10 bool Arc::Time::operator>= (const Time &) const

Comparing two Time objects.

5.72.3.11 static void Arc::Time::SetFormat (const TimeFormat &) [static]

Sets the default format for time strings.

5.72.3.12 void Arc::Time::SetTime (const time_t &)

sets the time

5.72.3.13 std::string Arc::Time::str (const TimeFormat & = `time_format`) const

Returns a string representation of the time, using the specified format.

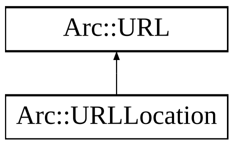
The documentation for this class was generated from the following file:

- DateTime.h

5.73 Arc::URL Class Reference

`#include <URL.h>`

Inheritance diagram for Arc::URL::



Public Member Functions

- URL ()
- URL (const std::string &url)
- virtual ~URL ()
- const std::string & Protocol () const
- const std::string & Username () const
- const std::string & Passwd () const
- const std::string & Host () const
- int Port () const
- const std::string & Path () const
- 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 CanonicalURL () 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)

5.73.1 Detailed Description

Class to hold general URL's. A URL is constructed from a string representation and split into protocol, hostname, port and path.

5.73.2 Constructor & Destructor Documentation

5.73.2.1 Arc::URL::URL ()

Empty constructor. Necessary when the class is part of another class and the like.

5.73.2.2 Arc::URL::URL (const std::string & *url*)

Constructs a new URL from a string representation. The string is split into protocol, hostname, port and path.

5.73.2.3 virtual Arc::URL::~URL () `[virtual]`

URL Destructor

5.73.3 Member Function Documentation

5.73.3.1 void Arc::URL::AddOption (const std::string & *option*, const std::string & *value*, bool *overwrite* = `true`)

Adds a URL option.

5.73.3.2 std::string Arc::URL::BaseDN () const

In case of ldap-protocol, return the basedn of the URL.

5.73.3.3 static std::string Arc::URL::BaseDN2Path (const std::string &) `[static, protected]`

a private method that converts an ldap basedn to a path.

5.73.3.4 virtual std::string Arc::URL::CanonicalURL () const `[virtual]`

Returns the URL string representation w/o options and locations

5.73.3.5 const std::string& Arc::URL::CommonLocOption (const std::string & *option*, const std::string & *undefined* = "") const

Returns the value of the common location option

Parameters:

option. Returns

undefined if the common location option is not defined.

5.73.3.6 const std::map<std::string, std::string>& Arc::URL::CommonLocOptions () const

Returns the common location options if any.

5.73.3.7 virtual std::string Arc::URL::ConnectionURL () const `[virtual]`

Returns a string representation with protocol, host and port only

5.73.3.8 const std::string& Arc::URL::Host () const

Returns the hostname of the URL.

5.73.3.9 const std::string& Arc::URL::HTTPOption (const std::string & *option*, const std::string & *undefined* = "") const

Returns the value of the HTTP option

Parameters:

option. Returns

undefined if the HTTP option is not defined.

5.73.3.10 const std::map<std::string, std::string>& Arc::URL::HTTPOptions () const

Returns HTTP options if any.

5.73.3.11 const std::list<URLLocation>& Arc::URL::Locations () const

Returns the locations if any.

5.73.3.12 Arc::URL::operator bool () const

Check if instance holds valid URL

5.73.3.13 bool Arc::URL::operator< (const URL & *url*) const

Compares one URL to another

5.73.3.14 bool Arc::URL::operator== (const URL & *url*) const

Is one URL equal to another?

5.73.3.15 const std::string& Arc::URL::Option (const std::string & *option*, const std::string & *undefined* = "") const

Returns the value of the URL option

Parameters:

option. Returns

undefined if the URL option is not defined.

5.73.3.16 const std::map<std::string, std::string>& Arc::URL::Options () const

Returns URL options if any.

5.73.3.17 const std::string& Arc::URL::Passwd () const

Returns the password of the URL.

5.73.3.18 const std::string& Arc::URL::Path () const

Returns the path of the URL.

5.73.3.19 static std::string Arc::URL::Path2BaseDN (const std::string &) `[static, protected]`

a private method that converts an ldap path to a basedn.

5.73.3.20 int Arc::URL::Port () const

Returns the port of the URL.

5.73.3.21 const std::string& Arc::URL::Protocol () const

Returns the protocol of the URL.

5.73.3.22 virtual std::string Arc::URL::str () const `[virtual]`

Returns a string representation of the URL.

Reimplemented in Arc::URLLocation.

5.73.3.23 const std::string& Arc::URL::Username () const

Returns the username of the URL.

5.73.4 Friends And Related Function Documentation

5.73.4.1 std::ostream& operator<< (std::ostream & *out*, const URL & *u*) `[friend]`

Overloaded operator << to print a URL.

5.73.5 Member Data Documentation

5.73.5.1 std::map<std::string, std::string> Arc::URL::commonlocoptions `[protected]`

common location options for index server URLs.

5.73.5.2 std::string Arc::URL::host `[protected]`

hostname of the url.

5.73.5.3 std::map<std::string, std::string> Arc::URL::httpoptions `[protected]`

http-options of the url.

5.73.5.4 std::list<URLLocation> Arc::URL::locations `[protected]`

locations for index server URLs.

5.73.5.5 std::string Arc::URL::passwd `[protected]`

password of the url.

5.73.5.6 std::string Arc::URL::path `[protected]`

the url path.

5.73.5.7 int Arc::URL::port `[protected]`

portnumber of the url.

5.73.5.8 std::string Arc::URL::protocol `[protected]`

the url protocol.

5.73.5.9 std::map<std::string, std::string> Arc::URL::urloptions `[protected]`

options of the url.

5.73.5.10 std::string Arc::URL::username `[protected]`

username of the url.

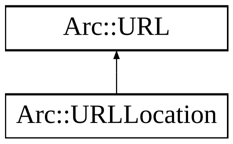
The documentation for this class was generated from the following file:

- URL.h

5.74 Arc::URLLocation Class Reference

```
#include <URL.h>
```

Inheritance diagram for Arc::URLLocation::



Public Member Functions

- URLLocation (const std::string &url)
- URLLocation (const std::string &name, const std::string &optstring)
- virtual ~URLLocation ()
- std::string Name () const
- virtual std::string str () const

Protected Attributes

- std::string name

5.74.1 Detailed Description

Class to hold a resolved URL location for an RC or RLS registration.

5.74.2 Constructor & Destructor Documentation

5.74.2.1 Arc::URLLocation::URLLocation (const std::string & *url*)

Creates a URL Location from a URL.

5.74.2.2 Arc::URLLocation::URLLocation (const std::string & *name*, const std::string & *optstring*)

Creates a URL Location from a name and an option string.

5.74.2.3 virtual Arc::URLLocation::~URLLocation () `[virtual]`

URL Location destructor.

5.74.3 Member Function Documentation

5.74.3.1 std::string Arc::URLLocation::Name () const

Returns the URL Location name (used for RC registrations).

5.74.3.2 virtual std::string Arc::URLLocation::str () const `[virtual]`

Returns a string representation of the URL Location.

Reimplemented from Arc::URL.

5.74.4 Member Data Documentation

5.74.4.1 std::string Arc::URLLocation::name `[protected]`

the URL Location name (used for RC registrations).

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

- URL.h

5.75 Arc::WSAEndpointReference Class 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_**

5.75.1 Detailed Description

This class implements interface for manipulating WS-Adressing Endpoint Reference stored in XML tree. Question: should there be some standalone class for storing EPR information?

5.75.2 Constructor & Destructor Documentation

5.75.2.1 Arc::WSAEndpointReference::WSAEndpointReference (XMLNode *epr*)

Linking to existing EPR in XML tree

5.75.2.2 Arc::WSAEndpointReference::WSAEndpointReference (const std::string & *address*)

Creating independent EPR - not implemented

5.75.2.3 Arc::WSAEndpointReference::WSAEndpointReference (void)

Dummy constructor - creates invalid instance

5.75.2.4 Arc::WSAEndpointReference::∼WSAEndpointReference (void)

Destructor. All empty elements of EPR XML are destroyed here too

5.75.3 Member Function Documentation

5.75.3.1 void Arc::WSAEndpointReference::Address (const std::string & *uri*)

Assigns new Address value. If EPR had no Address element it is created.

5.75.3.2 std::string Arc::WSAEndpointReference::Address (void) const

Returns Address (URL) encoded in EPR

5.75.3.3 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.

5.75.3.4 Arc::WSAEndpointReference::operator XMLNode (void)

Returns reference to EPR top XML node

5.75.3.5 WSAEndpointReference& Arc::WSAEndpointReference::operator= (const std::string & *address*)

Same as Address(uri)

5.75.3.6 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

5.76 Arc::WSAHeader Class Reference

```
#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_

5.76.1 Detailed Description

Interface to manipulate WS-Addressing related information in SOAP header

5.76.2 Constructor & Destructor Documentation

5.76.2.1 Arc::WSAHeader::WSAHeader (SOAPEnvelope & *soap*)

Linking to a header of existing SOAP message

5.76.2.2 Arc::WSAHeader::WSAHeader (const std::string & *action*)

Creating independent SOAP header - not implemented

5.76.3 Member Function Documentation

5.76.3.1 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.

5.76.3.2 std::string Arc::WSAHeader::Action (void) const

Returns content of Action element of SOAP Header.

5.76.3.3 static bool Arc::WSAHeader::Check (SOAPEnvelope & *soap*) `[static]`

Tells if specified SOAP message has WSA header

5.76.3.4 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.

5.76.3.5 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.

5.76.3.6 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.

5.76.3.7 std::string Arc::WSAHeader::MessageID (void) const

Returns content of MessageID element of SOAP Header.

5.76.3.8 XMLNode Arc::WSAHeader::NewReferenceParameter (const std::string & *name*)

Creates new ReferenceParameter element with specified name. Returns reference to created element.

5.76.3.9 Arc::WSAHeader::operator XMLNode (void)

Returns reference to SOAP Header - not implemented

5.76.3.10 XMLNode Arc::WSAHeader::ReferenceParameter (const std::string & *name*)

Returns first ReferenceParameter element with specified name

5.76.3.11 XMLNode Arc::WSAHeader::ReferenceParameter (int *n*)

Return n-th ReferenceParameter element

5.76.3.12 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.

5.76.3.13 std::string Arc::WSAHeader::RelatesTo (void) const

Returns content of RelatesTo element of SOAP Header.

5.76.3.14 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.

5.76.3.15 std::string Arc::WSAHeader::RelationshipType (void) const

Returns content of RelationshipType element of SOAP Header.

5.76.3.16 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.

5.76.3.17 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.

5.76.3.18 std::string Arc::WSAHeader::To (void) const

Returns content of To element of SOAP Header.

5.76.4 Member Data Documentation

5.76.4.1 bool Arc::WSAHeader::header_allocated_ [protected]

SOAP header element

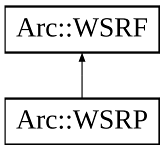
The documentation for this class was generated from the following file:

- WSA.h

5.77 Arc::WSRF Class Reference

```
#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_

5.77.1 Detailed Description

Base class for every WSRF message to be derived from

5.77.2 Constructor & Destructor Documentation

5.77.2.1 Arc::WSRF::WSRF (SOAPEnvelope & *soap*, const std::string & *action* = "")

Constructor - creates object out of supplied SOAP tree.

5.77.2.2 Arc::WSRF::WSRF (bool *fault* = false, const std::string & *action* = "")

Constructor - creates new WSRF object

5.77.3 Member Function Documentation

5.77.3.1 virtual Arc::WSRF::operator bool (void) `[inline, virtual]`

Returns true if instance is valid

5.77.3.2 void Arc::WSRF::set_namespaces (void) `[protected]`

set WS Resource namespaces and default prefixes in SOAP message

Reimplemented in Arc::WSRP.

5.77.3.3 virtual SOAPEnvelope& Arc::WSRF::SOAP (void) `[inline, virtual]`

Direct access to underlying SOAP element

5.77.4 Member Data Documentation

5.77.4.1 bool Arc::WSRF::allocated_ `[protected]`

Associated SOAP message - it's SOAP message after all

5.77.4.2 bool Arc::WSRF::valid_ `[protected]`

true if soap_ needs to be deleted in destructor

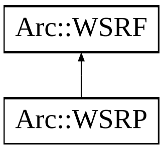
The documentation for this class was generated from the following file:

- WSRF.h

5.78 Arc::WSRP Class Reference

`#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)

5.78.1 Detailed Description

Base class for all 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.

5.78.2 Constructor & Destructor Documentation

5.78.2.1 Arc::WSRP::WSRP (bool *fault* = `false`, const std::string & *action* = `""`)

Constructor - prepares object for creation of new WSRP request/response/fault

5.78.2.2 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.

5.78.3 Member Function Documentation

5.78.3.1 void Arc::WSRP::set_namespaces (void) `[protected]`

set WS-ResourceProperties namespaces and default prefixes in SOAP message

Reimplemented from Arc::WSRF.

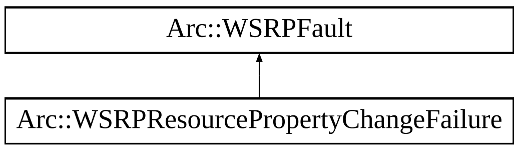
The documentation for this class was generated from the following file:

- WSResourceProperties.h

5.79 Arc::WSRPFault Class Reference

```
#include <WSResourceProperties.h>
```

Inheritance diagram for Arc::WSRPFault::



Public Member Functions

- WSRPFault (SOAPEnvelope &soap)
- WSRPFault (const std::string &type)

5.79.1 Detailed Description

Base class for all WS-ResourceProperties faults

5.79.2 Constructor & Destructor Documentation

5.79.2.1 Arc::WSRPFault::WSRPFault (SOAPEnvelope & *soap*)

Constructor - creates object out of supplied SOAP tree.

5.79.2.2 Arc::WSRPFault::WSRPFault (const std::string & *type*)

Constructor - creates new WSRP fault

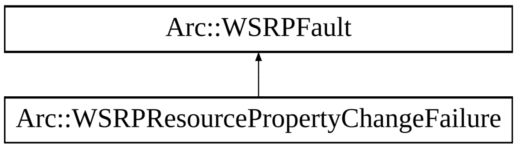
The documentation for this class was generated from the following file:

- WSResourceProperties.h

5.80 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)

5.80.1 Detailed Description

Base class for WS-ResourceProperties faults which contain ResourcePropertyChangeFailure

5.80.2 Constructor & Destructor Documentation

5.80.2.1 Arc::WSRPResourcePropertyChangeFailure::WSRPResourcePropertyChangeFailure (SOAPEnvelope & *soap*) `[inline]`

Constructor - creates object out of supplied SOAP tree.

5.80.2.2 Arc::WSRPResourcePropertyChangeFailure::WSRPResourcePropertyChangeFailure (const std::string & *type*) `[inline]`

Constructor - creates new WSRP fault

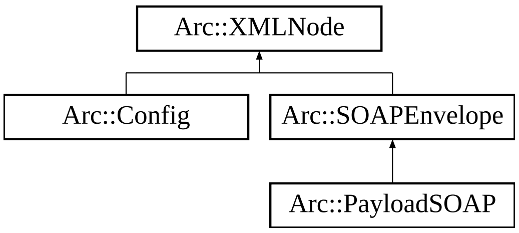
The documentation for this class was generated from the following file:

- WSResourceProperties.h

5.81 Arc::XMLNode Class Reference

`#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)
- ∼XMLNode (void)
- void New (XMLNode &new_node)
- 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
- std::string Name (void) const
- void Name (const std::string &name)
- void **Name** (const char ∗name)
- void GetXML (std::string &xml) const
- operator std::string (void) const
- XMLNode & operator= (const std::string &content)
- XMLNode & operator= (const char ∗content)
- XMLNode & operator= (const XMLNode &node)
- XMLNode Attribute (int n=0)
- XMLNode NewAttribute (const std::string &name)
- XMLNode NewAttribute (const char ∗name)
- XMLNode Attribute (const std::string &name)
- int AttributesSize (void)
- void Namespaces (const Arc::NS &namespaces)
- std::string NamespacePrefix (const char ∗urn)
- XMLNode NewChild (const std::string &name, int n=-1, bool global_order=false)
- XMLNode NewChild (const char ∗name, 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)

Protected Member Functions

- XMLNode (xmlNodePtr node)

Protected Attributes

- xmlNodePtr **node_**
- bool is_owner_
- bool is_temporary_

Friends

- bool MatchXMLName (const XMLNode &node1, const XMLNode &node2)
- bool MatchXMLName (const XMLNode &node, const char ∗name)

5.81.1 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.

5.81.2 Constructor & Destructor Documentation

5.81.2.1 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.

5.81.2.2 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.

5.81.2.3 Arc::XMLNode::XMLNode (const XMLNode & *node*) `[inline]`

Copies existing instance. Underlying XML element is NOT copied. Ownership is NOT inherited.

5.81.2.4 Arc::XMLNode::XMLNode (const std::string & *xml*) `[inline]`

Creates XML document structure from textual representation of XML document. Created structure is pointed and owned by constructed instance

5.81.2.5 Arc::XMLNode::XMLNode (const char ∗ *xml*, int *len* = `-1`) `[inline]`

Same as previous

5.81.2.6 Arc::XMLNode::XMLNode (const Arc::NS & *ns*) `[inline]`

Creates empty XML document structure with specified namespaces. Created structure is pointed and owned by constructed instance

5.81.2.7 Arc::XMLNode::∼XMLNode (void) `[inline]`

Destructor Also destroys underlying XML document if owned by this instance

5.81.3 Member Function Documentation

5.81.3.1 XMLNode Arc::XMLNode::Attribute (const std::string & *name*)

Returns XMLNode instance representing first attribute of node with specified by name

5.81.3.2 XMLNode Arc::XMLNode::Attribute (int *n* = `0`)

Returns XMLNode instance reresenting n-th attribute of node.

5.81.3.3 int Arc::XMLNode::AttributesSize (void)

Returns number of attributes of node

5.81.3.4 XMLNode Arc::XMLNode::Child (int *n* = `0`) const `[inline]`

Returns XMLNode instance representing n-th child of XML element. If such does not exist invalid XMLN-ode instance is returned

5.81.3.5 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

5.81.3.6 void Arc::XMLNode::GetXML (std::string & *xml*) const `[inline]`

Fills argument with this instance XML (sub)tree textual representation

Reimplemented in Arc::SOAPEnvelope.

5.81.3.7 void Arc::XMLNode::Name (const std::string & *name*)

Assign new name to XML node

5.81.3.8 std::string Arc::XMLNode::Name (void) const `[inline]`

Returns name of XML node

5.81.3.9 std::string Arc::XMLNode::NamespacePrefix (const char ∗ *urn*)

Returns prefix of specified namespace. Empty string if no such namespace.

5.81.3.10 void Arc::XMLNode::Namespaces (const Arc::NS & *namespaces*)

Assign 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.

5.81.3.11 void Arc::XMLNode::New (XMLNode & *new_node*)

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.

5.81.3.12 XMLNode Arc::XMLNode::NewAttribute (const char ∗ *name*)

Same as previous method

5.81.3.13 XMLNode Arc::XMLNode::NewAttribute (const std::string & *name*)

Creates new attribute with specified name.

5.81.3.14 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 on supplied one but not owned by returned instance

5.81.3.15 XMLNode Arc::XMLNode::NewChild (const char ∗ *name*, int *n* = `-1`, bool *global_order* = `false`)

Same as previous method

5.81.3.16 XMLNode Arc::XMLNode::NewChild (const std::string & *name*, int *n* = `-1`, bool *global_order* = `false`) `[inline]`

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

5.81.3.17 Arc::XMLNode::operator bool (void) const `[inline]`

Returns true if instance points to XML element - valid instance

5.81.3.18 Arc::XMLNode::operator std::string (void) const `[inline]`

Returns textual content of node excluding content of children nodes

5.81.3.19 bool Arc::XMLNode::operator! (void) const `[inline]`

Returns true if instance does not point to XML element - invalid instance

5.81.3.20 XMLNode& Arc::XMLNode::operator= (const XMLNode & *node*) `[inline]`

Make instance refer to another XML node. Ownership is not inherited.

5.81.3.21 XMLNode& Arc::XMLNode::operator= (const char ∗ *content*) `[inline]`

Same as previous method

5.81.3.22 XMLNode& Arc::XMLNode::operator= (const std::string & *content*) `[inline]`

Sets textual content of node. All existing children nodes are discarded.

5.81.3.23]

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]

5.81.3.24]

XMLNode Arc::XMLNode::operator[ ] (const std::string & *name*) const `[inline]`

Similar to previous method

5.81.3.25]

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

5.81.3.26 int Arc::XMLNode::Size (void) const `[inline]`

Returns number of children nodes

5.81.3.27 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").

5.81.4 Friends And Related Function Documentation

5.81.4.1 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

5.81.4.2 bool MatchXMLName (const XMLNode & *node1*, const XMLNode & *node2*) `[friend]`

Returns true if underlying XML elements have same names

5.81.5 Member Data Documentation

5.81.5.1 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.

5.81.5.2 bool Arc::XMLNode::is_temporary_ `[protected]`

This variable is for future

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

- XMLNode.h

Index