

KnowARC Reference Manual

Generated by Doxygen 1.5.1

Sun Sep 23 09:45:48 2007

Contents

| | | |
|----------|-----------------------------------------------------------|----------|
| 1 | KnowARC Hierarchical Index | 1 |
| 1.1 | KnowARC Class Hierarchy | 1 |
| 2 | KnowARC Class Index | 3 |
| 2.1 | KnowARC Class List | 3 |
| 3 | KnowARC Class Documentation | 5 |
| 3.1 | Arc::AttributeIterator Class Reference | 5 |
| 3.2 | Arc::ChainContext Class Reference | 9 |
| 3.3 | Arc::Checksum Class Reference | 10 |
| 3.4 | Arc::ChecksumAny Class Reference | 11 |
| 3.5 | Arc::Config Class Reference | 13 |
| 3.6 | Arc::Counter Class Reference | 15 |
| 3.7 | Arc::CounterTicket Class Reference | 22 |
| 3.8 | Arc::CRC32Sum Class Reference | 24 |
| 3.9 | Arc::DataBufferPar Class Reference | 25 |
| 3.10 | Arc::DataHandle Class Reference | 32 |
| 3.11 | Arc::DataPoint Class Reference | 33 |
| 3.12 | Arc::DataPointDirect Class Reference | 41 |
| 3.13 | Arc::DataPointDirect::analyze_t Class Reference | 46 |
| 3.14 | Arc::DataPointIndex Class Reference | 47 |
| 3.15 | Arc::DataPointIndex::Location Class Reference | 51 |
| 3.16 | Arc::DataSpeed Class Reference | 52 |
| 3.17 | Arc::DelegationConsumer Class Reference | 56 |
| 3.18 | Arc::DelegationProvider Class Reference | 57 |
| 3.19 | dmc_descriptor Struct Reference | 58 |
| 3.20 | Arc::DMCFactory Class Reference | 59 |
| 3.21 | Arc::ExpirationReminder Class Reference | 60 |
| 3.22 | Arc::FileInfo Class Reference | 62 |

| | |
|------------------------------------------------------------|-----|
| 3.23 Arc::InformationContainer Class Reference | 63 |
| 3.24 Arc::InformationInterface Class Reference | 65 |
| 3.25 Arc::InformationRequest Class Reference | 67 |
| 3.26 Arc::InformationResponse Class Reference | 69 |
| 3.27 Arc::IntraProcessCounter Class Reference | 70 |
| 3.28 Arc::Loader Class Reference | 74 |
| 3.29 Arc::loader_descriptor Struct Reference | 76 |
| 3.30 Arc::LoaderFactory Class Reference | 77 |
| 3.31 Arc::LogDestination Class Reference | 79 |
| 3.32 Arc::Logger Class Reference | 80 |
| 3.33 Arc::LogMessage Class Reference | 83 |
| 3.34 Arc::LogStream Class Reference | 85 |
| 3.35 Arc::MCC Class Reference | 87 |
| 3.36 mcc_descriptor Struct Reference | 89 |
| 3.37 Arc::MCC_Status Class Reference | 90 |
| 3.38 Arc::MCCFactory Class Reference | 93 |
| 3.39 Arc::MCCInterface Class Reference | 94 |
| 3.40 Arc::MD5Sum Class Reference | 95 |
| 3.41 Arc::Message Class Reference | 96 |
| 3.42 Arc::MessageAttributes Class Reference | 99 |
| 3.43 Arc::MessageAuth Class Reference | 102 |
| 3.44 Arc::MessageContext Class Reference | 103 |
| 3.45 Arc::MessageContextElement Class Reference | 104 |
| 3.46 Arc::MessagePayload Class Reference | 105 |
| 3.47 Arc::ModuleManager Class Reference | 106 |
| 3.48 Arc::PayloadRaw Class Reference | 107 |
| 3.49 Arc::PayloadRawInterface Class Reference | 110 |
| 3.50 Arc::PayloadSOAP Class Reference | 112 |
| 3.51 Arc::PayloadStream Class Reference | 113 |
| 3.52 Arc::PayloadStreamInterface Class Reference | 116 |
| 3.53 Arc::PayloadWSRF Class Reference | 118 |
| 3.54 pdp_descriptor Struct Reference | 120 |
| 3.55 Arc::PDPFactory Class Reference | 121 |
| 3.56 Arc::Plexer Class Reference | 122 |
| 3.57 Arc::PlexerEntry Class Reference | 124 |
| 3.58 Arc::RegularExpression Class Reference | 125 |

| | | |
|------|------------------------------------------------------------------------|-----|
| 3.59 | sechandler_descriptor Struct Reference | 127 |
| 3.60 | Arc::SecHandlerFactory Class Reference | 128 |
| 3.61 | Arc::Service Class Reference | 129 |
| 3.62 | service_descriptor Struct Reference | 131 |
| 3.63 | Arc::ServiceFactory Class Reference | 132 |
| 3.64 | Arc::SimpleCondition Class Reference | 133 |
| 3.65 | Arc::SOAPEnvelope Class Reference | 135 |
| 3.66 | Arc::SOAPFault Class Reference | 137 |
| 3.67 | Arc::SOAPMessage Class Reference | 140 |
| 3.68 | Arc::Time Class Reference | 142 |
| 3.69 | Arc::URL Class Reference | 145 |
| 3.70 | Arc::URLLocation Class Reference | 151 |
| 3.71 | Arc::WSAEndpointReference Class Reference | 153 |
| 3.72 | Arc::WSAHeader Class Reference | 155 |
| 3.73 | Arc::WSRF Class Reference | 158 |
| 3.74 | Arc::WSRFBaseFault Class Reference | 160 |
| 3.75 | Arc::WSRP Class Reference | 162 |
| 3.76 | Arc::WSRPFault Class Reference | 164 |
| 3.77 | Arc::WSRPResourcePropertyChangeFailure Class Reference | 165 |
| 3.78 | Arc::XMLNode Class Reference | 166 |

Chapter 1

KnowARC Hierarchical Index

1.1 KnowARC Class Hierarchy

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

| | |
|-------------------------------------------|----|
| Arc::AttributeIterator | 5 |
| Arc::ChainContext | 9 |
| Arc::Checksum | 10 |
| Arc::ChecksumAny | 11 |
| Arc::CRC32Sum | 24 |
| Arc::MD5Sum | 95 |
| Arc::Counter | 15 |
| Arc::IntraProcessCounter | 70 |
| Arc::CounterTicket | 22 |
| Arc::DataBufferPar | 25 |
| Arc::DataHandle | 32 |
| Arc::DataPoint | 33 |
| Arc::DataPointDirect | 41 |
| Arc::DataPointIndex | 47 |
| Arc::DataPointDirect::analyze_t | 46 |
| Arc::DataPointIndex::Location | 51 |
| Arc::DataSpeed | 52 |
| Arc::DelegationConsumer | 56 |
| Arc::DelegationProvider | 57 |
| dmc_descriptor | 58 |
| Arc::ExpirationReminder | 60 |
| Arc::FileInfo | 62 |
| Arc::InformationInterface | 65 |
| Arc::InformationContainer | 63 |
| Arc::InformationRequest | 67 |
| Arc::InformationResponse | 69 |
| Arc::Loader | 74 |
| Arc::loader_descriptor | 76 |
| Arc::LogDestination | 79 |
| Arc::LogStream | 85 |
| Arc::Logger | 80 |
| Arc::LogMessage | 83 |

| | |
|--------------------------------------------------|-----|
| mcc_descriptor | 89 |
| Arc::MCC_Status | 90 |
| Arc::MCCInterface | 94 |
| Arc::MCC | 87 |
| Arc::Plexer | 122 |
| Arc::Service | 129 |
| Arc::Message | 96 |
| Arc::MessageAttributes | 99 |
| Arc::MessageAuth | 102 |
| Arc::MessageContext | 103 |
| Arc::MessageContextElement | 104 |
| Arc::MessagePayload | 105 |
| Arc::PayloadRawInterface | 110 |
| Arc::PayloadRaw | 107 |
| Arc::PayloadSOAP | 112 |
| Arc::PayloadStreamInterface | 116 |
| Arc::PayloadStream | 113 |
| Arc::PayloadWSRF | 118 |
| Arc::ModuleManager | 106 |
| Arc::LoaderFactory | 77 |
| Arc::DMCFactory | 59 |
| Arc::MCCFactory | 93 |
| Arc::PDPFactory | 121 |
| Arc::SecHandlerFactory | 128 |
| Arc::ServiceFactory | 132 |
| pdp_descriptor | 120 |
| Arc::PlexerEntry | 124 |
| Arc::RegularExpression | 125 |
| sechandler_descriptor | 127 |
| service_descriptor | 131 |
| Arc::SimpleCondition | 133 |
| Arc::SOAPFault | 137 |
| Arc::SOAPMessage | 140 |
| Arc::Time | 142 |
| Arc::URL | 145 |
| Arc::URLLocation | 151 |
| Arc::WSAEndpointReference | 153 |
| Arc::WSAHeader | 155 |
| Arc::WSRF | 158 |
| Arc::WSRFBBaseFault | 160 |
| Arc::WSRPFault | 164 |
| Arc::WSRPResourcePropertyChangeFailure | 165 |
| Arc::WSRP | 162 |
| Arc::XMLNode | 166 |
| Arc::Config | 13 |
| Arc::SOAPEnvelope | 135 |
| Arc::PayloadSOAP | 112 |

Chapter 2

KnowARC Class Index

2.1 KnowARC Class List

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

| | |
|-----------------------------------------------------------------------------------------------------------------------------|----|
| Arc::AttributeIterator (An iterator class for accessing multiple values of an attribute) | 5 |
| Arc::ChainContext (Interface to chain specific functionality) | 9 |
| Arc::CheckSum (Defines interface for variuos checksum manipulations) | 10 |
| Arc::CheckSumAny (Wrapper for CheckSum class) | 11 |
| Arc::Config (Configuration element - represents (sub)tree of ARC configuration) | 13 |
| Arc::Counter (A class defining a common interface for counters) | 15 |
| Arc::CounterTicket (A class for "tickets" that correspond to counter reservations) | 22 |
| Arc::CRC32Sum (Implementation of CRC32 checksum) | 24 |
| Arc::DataBufferPar (Represents set of buffers) | 25 |
| Arc::DataHandle (This clas is a wrapper around the DataPoint class) | 32 |
| Arc::DataPoint (This class is an abstraction of URL) | 33 |
| Arc::DataPointDirect (This is kind of generalized file handle) | 41 |
| Arc::DataPointDirect::analyze_t | 46 |
| Arc::DataPointIndex (Complements DataPoint with attributes common for meta-URLs) | 47 |
| Arc::DataPointIndex::Location | 51 |
| Arc::DataSpeed (Keeps track of average and instantaneous transfer speed) | 52 |
| Arc::DelegationConsumer (Manages private key of delegation procedure) | 56 |
| Arc::DelegationProvider (Manages creddenials of delegation issuer) | 57 |
| dmc_descriptor | 58 |
| Arc::DMCFactory | 59 |
| Arc::ExpirationReminder (A class intended for internal use within counters) | 60 |
| Arc::FileInfo (FileInfo stores information about files (metadata)) | 62 |
| Arc::InformationContainer (Information System document container and processor) | 63 |
| Arc::InformationInterface (Information System message processor) | 65 |
| Arc::InformationRequest (Request for information in InfoSystem) | 67 |
| Arc::InformationResponse (Informational response from InfoSystem) | 69 |
| Arc::IntraProcessCounter (A class for counters used by threads within a single process) | 70 |
| Arc::Loader (Creator of Message Component Chains (MCC)) | 74 |
| Arc::loader_descriptor (Identifier of plugin) | 76 |
| Arc::LoaderFactory (Plugin handler) | 77 |
| Arc::LogDestination (A base class for log destinations) | 79 |
| Arc::Logger (A logger class) | 80 |
| Arc::LogMessage (A class for log messages) | 83 |

| | |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| Arc::LogStream (A class for logging to ostreams) | 85 |
| Arc::MCC (Message Chain Component - base class for every MCC plugin) | 87 |
| mcc_descriptor (Identifier of Message Chain Component (MCC) plugin) | 89 |
| Arc::MCC_Status (A class for communication of MCC processing results) | 90 |
| Arc::MCCFactory (MCC Plugins handler) | 93 |
| Arc::MCCInterface (Interface for communication between MCC , Service and Plexer objects) | 94 |
| Arc::MD5Sum (Implementation of MD5 checksum) | 95 |
| Arc::Message (Object being passed through chain of MCC s) | 96 |
| Arc::MessageAttributes (A class for storage of attribute values) | 99 |
| Arc::MessageAuth (Contains authenticity information, authorization tokens and decisions) | 102 |
| Arc::MessageContext (Handler for context of message context) | 103 |
| Arc::MessageContextElement (Top class for elements contained in message context) | 104 |
| Arc::MessagePayload (Base class for content of message passed through chain) | 105 |
| Arc::ModuleManager (Manager of shared libraries) | 106 |
| Arc::PayloadRaw (Raw byte multi-buffer) | 107 |
| Arc::PayloadRawInterface (Random Access Payload for Message objects) | 110 |
| Arc::PayloadSOAP (Payload of Message with SOAP content) | 112 |
| Arc::PayloadStream (POSIX handle as Payload) | 113 |
| Arc::PayloadStreamInterface (Stream-like Payload for Message object) | 116 |
| Arc::PayloadWSRF (This class combines MessagePayload with WSRF) | 118 |
| pdp_descriptor (Identifier of Policy Decision Point (PDP) plugin) | 120 |
| Arc::PDPFactory (PDP Plugins handler) | 121 |
| Arc::Plexer (The Plexer class, used for routing messages to services) | 122 |
| Arc::PlexerEntry (A pair of label (regex) and pointer to service) | 124 |
| Arc::RegularExpression (A regular expression class) | 125 |
| sechandler_descriptor (Identifier of SecHandler plugin) | 127 |
| Arc::SecHandlerFactory (SecHandler Plugins handler) | 128 |
| Arc::Service (Service - last component in a Message Chain) | 129 |
| service_descriptor (Identifier of Service plugin) | 131 |
| Arc::ServiceFactory (Service Plugins handler) | 132 |
| Arc::SimpleCondition (Simple triggered condition) | 133 |
| Arc::SOAPEnvelope (Extends XMLNode class to support structures of SOAP message) | 135 |
| Arc::SOAPFault (Interface to SOAP Fault message) | 137 |
| Arc::SOAPMessage (Message restricted to SOAP payload) | 140 |
| Arc::Time (A class for storing and manipulating times) | 142 |
| Arc::URL (Class to hold general URL's) | 145 |
| Arc::URLLocation (Class to hold a resolved URL location) | 151 |
| Arc::WSAEndpointReference (Interface for manipulation of WS-Adressing Endpoint Reference) | 153 |
| Arc::WSAHeader (Interface for manipulation WS-Addressing information in SOAP header) | 155 |
| Arc::WSRF (Base class for every WSRF message) | 158 |
| Arc::WSRFBASEFault (Base class for WSRF fault messages) | 160 |
| Arc::WSRP (Base class for WS-ResourceProperties structures) | 162 |
| Arc::WSRPFault (Base class for WS-ResourceProperties faults) | 164 |
| Arc::WSRPResourcePropertyChangeFailure | 165 |
| Arc::XMLNode (Wrapper for LibXML library Tree interface) | 166 |

Chapter 3

KnowARC Class Documentation

3.1 Arc::AttributeIterator Class Reference

An iterator class for accessing multiple values of an attribute.

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

Public Member Functions

- [AttributeIterator](#) ()
- const std::string & [operator *](#) () const
- const std::string * [operator →](#) () const
- const [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](#)

3.1.1 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;
```

3.1.2 Constructor & Destructor Documentation

3.1.2.1 Arc::AttributeIterator::AttributeIterator ()

Default constructor.

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

3.1.2.2 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.

3.1.3 Member Function Documentation

3.1.3.1 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.

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

The arrow operator.

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

3.1.3.3 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.

3.1.3.4 [AttributeIterator](#) Arc::AttributeIterator::operator++ (int)

The postfix advance operator.

Advances the iterator to the next value. Works intuitively.

Returns:

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

3.1.3.5 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.

3.1.4 Friends And Related Function Documentation

3.1.4.1 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.

3.1.5 Member Data Documentation

3.1.5.1 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.

3.1.5.2 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`

3.2 Arc::ChainContext Class Reference

Interface to chain specific functionality.

```
#include <Loader.h>
```

Public Member Functions

- [operator ServiceFactory * \(\)](#)
- [operator MCCFactory * \(\)](#)
- [operator SecHandlerFactory * \(\)](#)
- [operator PDPFactory * \(\)](#)

Friends

- class **Loader**

3.2.1 Detailed Description

Interface to chain specific functionality.

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

3.2.2 Member Function Documentation

3.2.2.1 Arc::ChainContext::operator [ServiceFactory](#) * () [inline]

Returns associated [ServiceFactory](#) object

3.2.2.2 Arc::ChainContext::operator [MCCFactory](#) * () [inline]

Returns associated [MCCFactory](#) object

3.2.2.3 Arc::ChainContext::operator [SecHandlerFactory](#) * () [inline]

Returns associated [SecHandlerFactory](#) object

3.2.2.4 Arc::ChainContext::operator [PDPFactory](#) * () [inline]

Returns associated [PDPFactory](#) object

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

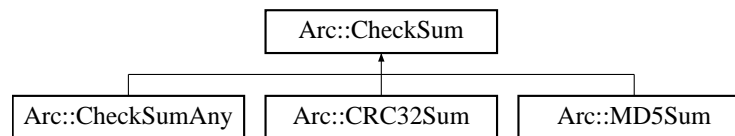
- [Loader.h](#)

3.3 Arc::Checksum Class Reference

Defines interface for variuos checksum manipulations.

```
#include <Checksum.h>
```

Inheritance diagram for Arc::Checksum::



Public Member Functions

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

3.3.1 Detailed Description

Defines interface for variuos checksum manipulations.

This class is used during data transfers through [DataBufferPar](#) class

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

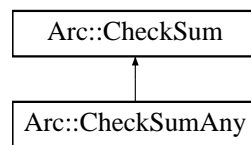
- CheckSum.h

3.4 Arc::ChecksumAny Class Reference

Wrapper for [Checksum](#) class.

```
#include <Checksum.h>
```

Inheritance diagram for Arc::ChecksumAny::



Public Types

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

Public Member Functions

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

Static Public Member Functions

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

3.4.1 Detailed Description

Wrapper for [Checksum](#) class.

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

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

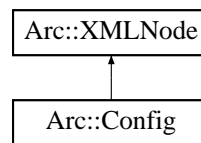
- CheckSum.h

3.5 Arc::Config Class Reference

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

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

Inheritance diagram for Arc::Config::



Public Member Functions

- [Config](#) ()
- [Config](#) (const char *filename)
- [Config](#) (const std::string &xml_str)
- [Config](#) (Arc::XMLNode xml)
- void [print](#) (void)
- void [parse](#) (const char *filename)

3.5.1 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.

3.5.2 Constructor & Destructor Documentation

3.5.2.1 Arc::Config::Config () [inline]

Dummy constructor - produces empty structure

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

Loads configuration document from file 'filename'

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

Parse configuration document from memory

3.5.2.4 `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.

3.5.3 Member Function Documentation

3.5.3.1 `void Arc::Config::print` (void)

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

3.5.3.2 `void Arc::Config::parse` (const char **filename*)

Parse configuration document from file 'filename'

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

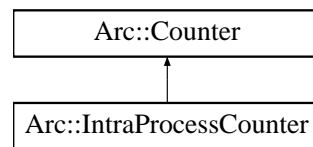
- ArcConfig.h

3.6 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`

3.6.1 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(...);
```

3.6.2 Member Typedef Documentation

3.6.2.1 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 numbers (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.

3.6.3 Constructor & Destructor Documentation

3.6.3.1 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.

3.6.3.2 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.

3.6.4 Member Function Documentation

3.6.4.1 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](#).

3.6.4.2 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](#).

3.6.4.3 `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](#).

3.6.4.4 `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](#).

3.6.4.5 `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](#).

3.6.4.6 `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](#).

3.6.4.7 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 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](#).

3.6.4.8 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](#).

3.6.4.9 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.

3.6.4.10 **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.

3.6.4.11 **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.

3.6.4.12 **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.

3.6.4.13 **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](#) (but 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.

3.6.4.14 [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](#) (but 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.

3.6.5 Friends And Related Function Documentation**3.6.5.1** friend class [CounterTicket](#) [friend]

The [CounterTicket](#) class needs to be a friend.

3.6.5.2 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

3.7 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](#)

3.7.1 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();
```

3.7.2 Constructor & Destructor Documentation

3.7.2.1 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](#).

3.7.3 Member Function Documentation

3.7.3.1 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.

3.7.3.2 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.

3.7.3.3 void Arc::CounterTicket::cancel ()

Cancels a reservation.

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.

3.7.4 Friends And Related Function Documentation**3.7.4.1 friend class [Counter](#) [friend]**

The [Counter](#) class needs to be a friend.

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

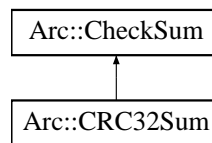
- Counter.h

3.8 Arc::CRC32Sum Class Reference

Implementation of CRC32 checksum.

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

Inheritance diagram for Arc::CRC32Sum::



Public Member Functions

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

3.8.1 Detailed Description

Implementation of CRC32 checksum.

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

- CheckSum.h

3.9 Arc::DataBufferPar Class Reference

Represents set of buffers.

```
#include <DataBufferPar.h>
```

Public Member Functions

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

Public Attributes

- [DataSpeed speed](#)

Classes

- struct `buf_desc`

3.9.1 Detailed Description

Represents set of buffers.

This class is used during data transfer using [DataPoint](#) classes.

3.9.2 Constructor & Destructor Documentation

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

Constructor

Parameters:

size size of every buffer in bytes.

blocks number of buffers.

3.9.2.2 `Arc::DataBufferPar::DataBufferPar (Checksum * cksum, unsigned int size = 65536, int blocks = 3)`

Constructor

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.

3.9.2.3 `Arc::DataBufferPar::~~DataBufferPar ()`

Destructor.

3.9.3 Member Function Documentation

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

Check if [DataBufferPar](#) object is initialized.

3.9.3.2 `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.

3.9.3.3]

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

Direct access to buffer by number.

3.9.3.4 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

3.9.3.5 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.

3.9.3.6 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.

3.9.3.7 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.

3.9.3.8 **bool Arc::DataBufferPar::for_write (int & *handle*, unsigned int & *length*, unsigned 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.

3.9.3.9 **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.

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

Informs object that data was written from buffer.

Parameters:

handle buffer's number.

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

Informs object that data was written from buffer.

Parameters:

buf - address of buffer

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

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

Parameters:

handle buffer's number.

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

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

Parameters:

buf - address of buffer

3.9.3.14 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.

3.9.3.15 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.

3.9.3.16 void Arc::DataBufferPar::error_read (bool v)

Informs object if error accured on 'read' side.

Parameters:

v true if error.

3.9.3.17 void Arc::DataBufferPar::error_write (bool v)

Informs object if error accured on 'write' side.

Parameters:

v true if error.

3.9.3.18 bool Arc::DataBufferPar::eof_read ()

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

3.9.3.19 bool Arc::DataBufferPar::eof_write ()

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

3.9.3.20 bool Arc::DataBufferPar::error_read ()

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

3.9.3.21 bool Arc::DataBufferPar::error_write ()

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

3.9.3.22 bool Arc::DataBufferPar::error_transfer ()

Returns true if eror occured inside object.

3.9.3.23 bool Arc::DataBufferPar::error ()

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

3.9.3.24 bool Arc::DataBufferPar::wait ()

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

3.9.3.25 bool Arc::DataBufferPar::wait_used ()

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

3.9.3.26 bool Arc::DataBufferPar::checksum_valid ()

Returns true if checksum was successfully computed.

3.9.3.27 const [Checksum](#)* Arc::DataBufferPar::checksum_object ()

Returns [Checksum](#) object specified in constructor.

3.9.3.28 bool Arc::DataBufferPar::wait_eof_read ()

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

3.9.3.29 bool Arc::DataBufferPar::wait_read ()

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

3.9.3.30 bool Arc::DataBufferPar::wait_eof_write ()

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

3.9.3.31 bool Arc::DataBufferPar::wait_write ()

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

3.9.3.32 bool Arc::DataBufferPar::wait_eof ()

Wait till end of transfer happens on any side.

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

Returns offset following last piece of data transfered.

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

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

3.9.4 Member Data Documentation

3.9.4.1 [DataSpeed](#) Arc::DataBufferPar::speed

This object controls transfer speed.

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

- DataBufferPar.h

3.10 Arc::DataHandle Class Reference

This class is a wrapper around the [DataPoint](#) class.

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

Public Member Functions

- **DataHandle** (const [URL](#) &url)
- [DataPoint](#) * **operator** → ()
- bool **operator!** ()
- **operator bool** ()

3.10.1 Detailed Description

This class is a wrapper around the [DataPoint](#) class.

It simplifies the construction and use and destruction of [DataPoint](#) objects.

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

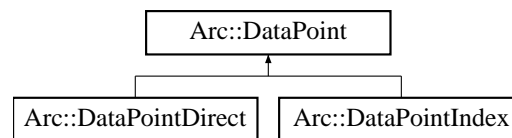
- DataHandle.h

3.11 Arc::DataPoint Class Reference

This class is an abstraction of [URL](#).

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

Inheritance diagram for Arc::DataPoint::



Public Member Functions

- [DataPoint](#) (const [URL](#) &url)
- virtual bool [meta_resolve](#) (bool)
- virtual bool [meta_preregister](#) (bool, bool=false)
- virtual bool [meta_postregister](#) (bool)
- virtual bool [meta_register](#) (bool replication)
- virtual bool [meta_preunregister](#) (bool)
- virtual bool [meta_unregister](#) (bool)
- virtual bool [list_files](#) (std::list< [FileInfo](#) > &, bool=true)
- virtual bool [get_info](#) ([FileInfo](#) &)
- 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](#) &)
- virtual int [tries](#) ()
- virtual void [tries](#) (int n)
- virtual const [URL](#) & [base_url](#) () const
- virtual bool [add_location](#) (const std::string &, const [URL](#) &)

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_](#)

3.11.1 Detailed Description

This class is an abstraction of [URL](#).

It can handle URLs of type [file://](#), [ftp://](#), [gsiftp://](#), [http://](#), [https://](#), [http://](#) (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.

3.11.2 Constructor & Destructor Documentation

3.11.2.1 [Arc::DataPoint::DataPoint](#) (const [URL](#) & [url](#))

Constructor requires [URL](#) or meta-URL to be provided.

3.11.3 Member Function Documentation

3.11.3.1 [virtual bool Arc::DataPoint::meta_resolve](#) (bool) [inline, virtual]

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

Parameters:

source true if [DataPoint](#) object represents source of information

3.11.3.2 virtual bool Arc::DataPoint::meta_preregister (bool, bool = 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](#).

3.11.3.3 virtual bool Arc::DataPoint::meta_postregister (bool) [inline, virtual]

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

Parameters:

replication if true then file is being replicated between 2 locations registered in Indexing [Service](#) under same name.

3.11.3.4 virtual bool Arc::DataPoint::meta_preunregister (bool) [inline, virtual]

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

3.11.3.5 virtual bool Arc::DataPoint::meta_unregister (bool) [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.

3.11.3.6 virtual bool Arc::DataPoint::list_files (std::list< [FileInfo](#) > &, bool = 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 properties of objects.

Reimplemented in [Arc::DataPointDirect](#).

3.11.3.7 virtual bool Arc::DataPoint::get_info (FileInfo &) [inline, virtual]

Retrieve properties of object pointed by meta-URL of [DataPoint](#) object. It works only for meta-URL.

Parameters:

fi contains retrieved information.

Reimplemented in [Arc::DataPointIndex](#).

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

Check if meta-information 'size' is available.

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

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

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

Set value of meta-information 'size'.

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

Get value of meta-information 'size'.

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

Check if meta-information 'checksum' is available.

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

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

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

Set value of meta-information 'checksum'.

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

Get value of meta-information 'checksum'.

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

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

3.11.3.17 virtual void Arc::DataPoint::meta_created (Time val) [inline, virtual]

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

3.11.3.18 virtual void Arc::DataPoint::meta_created_force (Time val) [inline, virtual]

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

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

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

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

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

3.11.3.21 virtual void Arc::DataPoint::meta_validtill (Time val) [inline, virtual]

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

3.11.3.22 virtual void Arc::DataPoint::meta_validtill_force (Time val) [inline, virtual]

Set value of meta-information 'validity time'.

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

Get value of meta-information 'validity time'.

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

Check if [URL](#) is meta-URL.

Reimplemented in [Arc::DataPointDirect](#), and [Arc::DataPointIndex](#).

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

If endpoint can have any use from meta information.

Reimplemented in [Arc::DataPointIndex](#).

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

If endpoint can provide at least some meta information directly.

Reimplemented in [Arc::DataPointIndex](#).

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

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

Parameters:

p object from which information is taken.

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

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

Parameters:

p object to which compare.

3.11.3.29 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](#).

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

Check if file is local ([URL](#) is something like `file://`).

3.11.3.31 virtual const [URL](#)& Arc::DataPoint::current_location () const [inline, virtual]

Returns current (resolved) [URL](#).

Reimplemented in [Arc::DataPointIndex](#).

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

Returns meta information used to create current [URL](#). For RC that is location's name. For RLS that is equal to pfn.

Reimplemented in [Arc::DataPointIndex](#).

3.11.3.33 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](#).

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

Returns false if out of retries.

Reimplemented in [Arc::DataPointIndex](#).

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

Returns true if number of resolved URLs is not 0.

Reimplemented in [Arc::DataPointIndex](#).

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

Remove current [URL](#) from list.

Reimplemented in [Arc::DataPointIndex](#).

3.11.3.37 virtual bool Arc::DataPoint::remove_locations (const [DataPoint](#) &) [inline, virtual]

Remove locations present in another [DataPoint](#) object.

Reimplemented in [Arc::DataPointIndex](#).

3.11.3.38 virtual int Arc::DataPoint::tries () [virtual]

Returns number of retries left.

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

Set number of retries.

Reimplemented in [Arc::DataPointIndex](#).

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

Returns [URL](#) which was passed to constructor.

3.11.3.41 virtual bool Arc::DataPoint::add_location (const std::string &, const [URL](#) &) [inline, virtual]

Add [URL](#) to list.

Parameters:

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

loc [URL](#).

Reimplemented in [Arc::DataPointIndex](#).

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

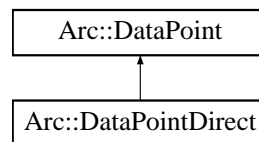
- DataPoint.h

3.12 Arc::DataPointDirect Class Reference

This is kind of generalized file handle.

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

Inheritance diagram for Arc::DataPointDirect::



Public Types

- 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](#)

3.12.1 Detailed Description

This is kind of generalized file handle.

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

3.12.2 Member Enumeration Documentation

3.12.2.1 enum [Arc::DataPointDirect::failure_reason_t](#)

Reason of transfer failure.

3.12.3 Constructor & Destructor Documentation

3.12.3.1 [Arc::DataPointDirect::DataPointDirect](#) (const [URL](#) & *url*)

Constructor

Parameters:

url [URL](#).

3.12.3.2 [virtual Arc::DataPointDirect::~~DataPointDirect](#) () [virtual]

Destructor. No comments.

3.12.4 Member Function Documentation

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

Check if [URL](#) is meta-URL.

Reimplemented from [Arc::DataPoint](#).

3.12.4.2 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.

3.12.4.3 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.

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

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

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

Same as stop_reading but for corresponding start_writing.

3.12.4.6 virtual bool Arc::DataPointDirect::analyze (analyze_t & arg) [virtual]

Analyze url and provide hints.

Parameters:

arg returns suggested values.

3.12.4.7 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](#).

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

Remove/delete object at [URL](#).

3.12.4.9 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.

Reimplemented from [Arc::DataPoint](#).

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

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

3.12.4.11 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.

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

Allow/disallow to make check for existence of remote file (and probably other checks too) before initiating 'reading' and 'writing' operations.

Parameters:

v true if allowed (default is true).

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

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

3.12.4.14 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://).

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

Check if heavy security during data transfer is allowed.

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

Request passive transfers for FTP-like protocols.

Parameters:

true to request.

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

Returns reason of transfer failure.

3.12.4.18 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.

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

- DataPointDirect.h

3.13 Arc::DataPointDirect::analyze_t Class Reference

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

Public Attributes

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

3.13.1 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:

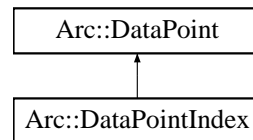
- DataPointDirect.h

3.14 Arc::DataPointIndex Class Reference

Complements [DataPoint](#) with attributes common for meta-URLs.

```
#include <DataPointIndex.h>
```

Inheritance diagram for Arc::DataPointIndex::



Public Member Functions

- **DataPointIndex** (const [URL](#) &url)
- virtual bool [get_info](#) ([FileInfo](#) &fi)
- virtual const [URL](#) & [current_location](#) () const
- virtual const std::string & [current_meta_location](#) () const
- virtual bool [next_location](#) ()
- virtual bool [have_location](#) () const
- virtual bool [have_locations](#) () const
- virtual bool [remove_location](#) ()
- virtual bool [remove_locations](#) (const [DataPoint](#) &p)
- virtual bool [add_location](#) (const std::string &meta, const [URL](#) &loc)
- virtual bool [meta](#) () const
- virtual bool [accepts_meta](#) ()
- virtual bool [provides_meta](#) ()
- virtual bool [meta_stored](#) ()
- virtual void [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](#)

3.14.1 Detailed Description

Complements [DataPoint](#) with attributes common for meta-URLs.

It should never be used directly. Instead inherit from it to provide class for specific Indexing [Service](#).

3.14.2 Member Function Documentation

3.14.2.1 `virtual bool Arc::DataPointIndex::get_info (FileInfo &fi) [virtual]`

Retrieve properties of object pointed by meta-URL of [DataPoint](#) object. It works only for meta-URL.

Parameters:

fi contains retrieved information.

Reimplemented from [Arc::DataPoint](#).

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

Returns current (resolved) [URL](#).

Reimplemented from [Arc::DataPoint](#).

3.14.2.3 `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](#).

3.14.2.4 `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](#).

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

Returns false if out of retries.

Reimplemented from [Arc::DataPoint](#).

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

Returns true if number of resolved URLs is not 0.

Reimplemented from [Arc::DataPoint](#).

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

Remove current [URL](#) from list.

Reimplemented from [Arc::DataPoint](#).

3.14.2.8 virtual bool Arc::DataPointIndex::remove_locations (const [DataPoint](#) & p) [virtual]

Remove locations present in another [DataPoint](#) object.

Reimplemented from [Arc::DataPoint](#).

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

Add [URL](#) to list.

Parameters:

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

loc [URL](#).

Reimplemented from [Arc::DataPoint](#).

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

Check if [URL](#) is meta-URL.

Reimplemented from [Arc::DataPoint](#).

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

If endpoint can have any use from meta information.

Reimplemented from [Arc::DataPoint](#).

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

If endpoint can provide at least some meta information directly.

Reimplemented from [Arc::DataPoint](#).

3.14.2.13 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](#).

3.14.2.14 virtual void Arc::DataPointIndex::tries (int n) [virtual]

Set number of retries.

Reimplemented from [Arc::DataPoint](#).

3.14.3 Member Data Documentation

3.14.3.1 `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:

- DataPointIndex.h

3.15 Arc::DataPointIndex::Location Class Reference

```
#include <DataPointIndex.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**

3.15.1 Detailed Description

[DataPointIndex::Location](#) represents physical service at which files are located aka "base URL" including 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:

- DataPointIndex.h

3.16 Arc::DataSpeed Class Reference

Keeps track of average and instantaneous transfer speed.

```
#include <DataSpeed.h>
```

Public Types

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

Public Member Functions

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

3.16.1 Detailed Description

Keeps track of average and instantaneous transfer speed.

Also detects data transfer inactivity and other transfer timeouts.

3.16.2 Constructor & Destructor Documentation

3.16.2.1 Arc::DataSpeed::DataSpeed (time_t base = DATASPEED_AVERAGING_PERIOD)

Constructor

Parameters:

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

3.16.2.2 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.

3.16.2.3 Arc::DataSpeed::~DataSpeed (void)

Destructor.

3.16.3 Member Function Documentation

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

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

3.16.3.2 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.

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

Check if speed information is going to be printed.

3.16.3.4 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

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

Set minimal average speed.

Parameters:

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

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

Set inactivity timeout.

Parameters:

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

3.16.3.7 void Arc::DataSpeed::set_base (time_t *base* = DATASPEED_AVERAGING_PERIOD)

Set averaging time period.

Parameters:

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

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

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

Parameters:

max - amount of data in bytes.

3.16.3.9 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.

3.16.3.10 void Arc::DataSpeed::reset (void)

Reset all counters and triggers.

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

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

Parameters:

n amount of data transfered (bytes).

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

Turn off speed control.

Parameters:

disable true to turn off.

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

Check if minimal speed error was triggered.

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

Check if minimal average speed error was triggered.

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

Check if maximal inactivity time error was triggered.

3.16.3.16 unsigned long long int Arc::DataSpeed::transferred_size (void) [inline]

Returns amount of data this object knows about.

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

- DataSpeed.h

3.17 Arc::DelegationConsumer Class Reference

Manages private key of delegation procedure.

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

Public Member Functions

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

Protected Member Functions

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

Protected Attributes

- void * **key_**

3.17.1 Detailed Description

Manages private key of delegation procedure.

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

- DelegationInterface.h

3.18 Arc::DelegationProvider Class Reference

Manages credentials of delegation issuer.

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

Public Member Functions

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

3.18.1 Detailed Description

Manages credentials of delegation issuer.

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

- DelegationInterface.h

3.19 dmc_descriptor Struct Reference

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

Public Attributes

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

3.19.1 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.

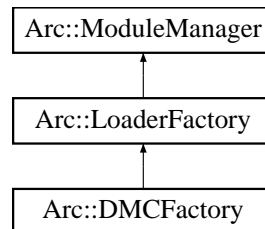
The documentation for this struct was generated from the following file:

- DMCLoader.h

3.20 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)

3.20.1 Detailed Description

This class handles shared libraries containing DMCs

3.20.2 Constructor & Destructor Documentation

3.20.2.1 Arc::DMCFactory::DMCFactory ([Config](#) * *cfg*)

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

3.20.3 Member Function Documentation

3.20.3.1 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

3.21 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](#)

3.21.1 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.

3.21.2 Member Function Documentation

3.21.2.1 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.

3.21.2.2 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.

3.21.2.3 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.

3.21.3 Friends And Related Function Documentation

3.21.3.1 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

3.22 Arc::FileInfo Class Reference

[FileInfo](#) stores information about files (metadata).

```
#include <FileInfo.h>
```

Public Types

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

Public Member Functions

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

3.22.1 Detailed Description

[FileInfo](#) stores information about files (metadata).

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

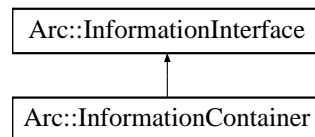
- FileInfo.h

3.23 Arc::InformationContainer Class Reference

Information System document container and processor.

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

Inheritance diagram for Arc::InformationContainer::



Public Member Functions

- **InformationContainer** ([XMLNode](#) doc, bool copy=false)
- [XMLNode Acquire](#) (void)
- void **Release** (void)

Protected Member Functions

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

Protected Attributes

- [XMLNode doc_](#)

3.23.1 Detailed Description

Information System document container and processor.

This class inherits from [InformationInterface](#) and offers container for storing informational XML document.

3.23.2 Member Function Documentation

3.23.2.1 virtual std::list<[XMLNode](#)> Arc::InformationContainer::Get (const std::list< std::string > &path) [protected, virtual]

This method is called by this object's Process method. Real implementation of this class should return (sub)tree of XML document. This method may be called multiple times per single Process call.

Reimplemented from [Arc::InformationInterface](#).

3.23.2.2 [XMLNode](#) Arc::InformationContainer::Acquire (void)

Get a lock on contained XML document. To be used in multi-threaded environment. Do not forget to release it with Release()

3.23.3 Member Data Documentation

3.23.3.1 [XMLNode Arc::InformationContainer::doc_](#) [protected]

Either link or container of XML document

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

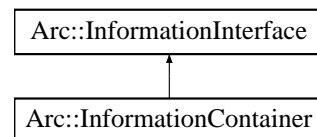
- InformationInterface.h

3.24 Arc::InformationInterface Class Reference

Information System message processor.

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

Inheritance diagram for Arc::InformationInterface::



Public Member Functions

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

Protected Member Functions

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

Protected Attributes

- Glib::Mutex [lock_](#)
- bool [to_lock_](#)

3.24.1 Detailed Description

Information System message processor.

This class provides callback for 2 operations of WS-ResourceProperties and convenient parsing/generation of corresponding SOAP messages. In a future it may extend range of supported specifications.

3.24.2 Constructor & Destructor Documentation

3.24.2.1 Arc::InformationInterface::InformationInterface (bool *safe* = true)

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

3.24.3 Member Function Documentation

3.24.3.1 virtual std::list<[XMLNode](#)> Arc::InformationInterface::Get (const std::list< std::string > &*path*) [protected, virtual]

This method is called by this object's Process method. Real implementation of this class should return (sub)tree of XML document. This method may be called multiple times per single Process call.

Reimplemented in [Arc::InformationContainer](#).

3.24.4 Member Data Documentation

3.24.4.1 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

3.25 Arc::InformationRequest Class Reference

Request for information in InfoSystem.

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

Public Member Functions

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

3.25.1 Detailed Description

Request for information in InfoSystem.

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

3.25.2 Constructor & Destructor Documentation

3.25.2.1 Arc::InformationRequest::InformationRequest (void)

Dummy constructor

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

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

3.25.2.3 Arc::InformationRequest::InformationRequest (const std::list< std::list< std::string > > &paths)

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

3.25.2.4 Arc::InformationRequest::InformationRequest ([XMLNode](#) query)

Request for attributes specified by XPath query.

3.25.3 Member Function Documentation

3.25.3.1 [SOAPEnvelope](#)* Arc::InformationRequest::SOAP (void)

Returns generated SOAP message

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

- [InformationInterface.h](#)

3.26 Arc::InformationResponse Class Reference

Informational response from InfoSystem.

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

Public Member Functions

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

3.26.1 Detailed Description

Informational response from InfoSystem.

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

3.26.2 Constructor & Destructor Documentation

3.26.2.1 Arc::InformationResponse::InformationResponse ([SOAPEnvelope](#) & *soap*)

Constructor parses WS-ResourceProperties response. Provided [SOAPEnvelope](#) object must be valid as long as this object is in use.

3.26.3 Member Function Documentation

3.26.3.1 `std::list<XMLNode> Arc::InformationResponse::Result` (void)

Returns set of attributes which were in SOAP message passed to constructor.

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

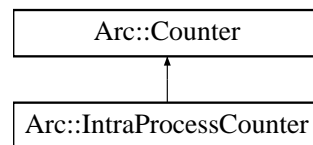
- InformationInterface.h

3.27 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)

3.27.1 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.

3.27.2 Constructor & Destructor Documentation

3.27.2.1 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.

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

Destructor.

This is the destructor of the [IntraProcessCounter](#) class. Does not need to do anything.

3.27.3 Member Function Documentation**3.27.3.1 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](#).

3.27.3.2 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](#).

3.27.3.3 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](#).

3.27.3.4 `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](#).

3.27.3.5 `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](#).

3.27.3.6 `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](#).

3.27.3.7 `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](#).

3.27.3.8 `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](#).

3.27.3.9 `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.

3.27.3.10 `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.

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

- IntraProcessCounter.h

3.28 Arc::Loader Class Reference

Creator of [Message](#) Component Chains ([MCC](#)).

```
#include <Loader.h>
```

Public Types

- typedef std::map< std::string, [MCC](#) * > **mcc_container_t**
- typedef std::map< std::string, [Service](#) * > **service_container_t**
- typedef std::map< std::string, 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**

3.28.1 Detailed Description

Creator of [Message](#) Component Chains ([MCC](#)).

This class processes XML configuration and creates message chains. Accepted configuration is defined by XML schema mcc.xsd. Supported components are of types [MCC](#), [Service](#) and [Plexer](#). [MCC](#) and [Service](#) are loaded from dynamic libraries. For [Plexer](#) only internal implementation is supported. This object is also a container for loaded componets. All components and chains are destroyed if this object is destroyed. Chains are created in 2 steps. First all components are loaded and corresponding objects are created. Constructors are supplied with corresponding configuration subtrees. During next step components are linked together by calling their Next() methods. Each call creates labeled link to next component in a chain. 2 step method has an advantage over single step because it allows loops in chains and makes loading procedure more simple. But that also means during short period of time components are only partly configured. Components in such state must produce proper error response if [Message](#) arrives. Note: Current implementation requires all components and links to be labeled. All labels must be unique. Future implementation will be able to assign labels automatically.

3.28.2 Constructor & Destructor Documentation

3.28.2.1 Arc::Loader::Loader ([Config](#) * cfg)

Constructor that takes whole XML configuration and creates component chains

3.28.2.2 Arc::Loader::~~Loader ()

Destructor destroys all components created by constructor

3.28.3 Member Function Documentation

3.28.3.1]

[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

3.29 Arc::loader_descriptor Struct Reference

Identifier of plugin.

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

Public Attributes

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

3.29.1 Detailed Description

Identifier of plugin.

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

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

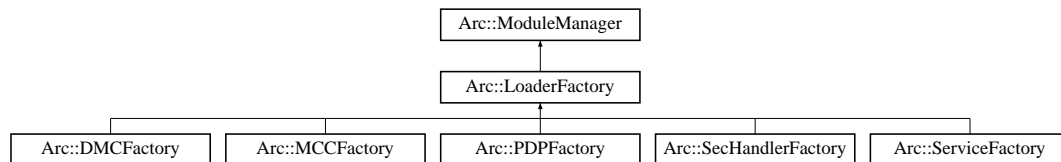
- LoaderFactory.h

3.30 Arc::LoaderFactory Class Reference

Plugin handler.

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

Inheritance diagram for Arc::LoaderFactory::



Public Member Functions

- void [load_all_instances](#) (const std::string &libname)

Protected Member Functions

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

3.30.1 Detailed Description

Plugin handler.

This class handles shared libraries containing loadable classes

3.30.2 Constructor & Destructor Documentation

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

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

3.30.3 Member Function Documentation

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

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

Reimplemented in [Arc::DMCFactory](#), [Arc::MCCFactory](#), [Arc::PDPFactory](#), [Arc::SecHandlerFactory](#), and [Arc::ServiceFactory](#).

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

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

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

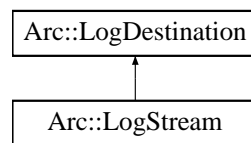
- LoaderFactory.h

3.31 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](#) ()

3.31.1 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.

3.31.2 Constructor & Destructor Documentation

3.31.2.1 Arc::LogDestination::LogDestination () [protected]

Default constructor.

The only constructor needed by subclasses, since the [LogDestination](#) class has no attributes.

3.31.3 Member Function Documentation

3.31.3.1 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](#)

3.32 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

3.32.1 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.

3.32.2 Constructor & Destructor Documentation

3.32.2.1 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.

3.32.2.2 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.

3.32.3 Member Function Documentation

3.32.3.1 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 LogDestinatoin 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](#) itself.

3.32.3.2 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

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

Returns the threshold.

Returns the threshold.

Returns:

The threshold of this [Logger](#).

3.32.3.4 void Arc::Logger::msg (LogMessage message)

Sends a [LogMessage](#).

Sends a [LogMessage](#).

Parameters:

The [LogMessage](#) to send.

3.32.3.5 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.

3.32.4 Member Data Documentation

3.32.4.1 [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](#)

3.33 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)

3.33.1 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.

3.33.2 Constructor & Destructor Documentation

3.33.2.1 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.

3.33.2.2 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](#).

3.33.3 Member Function Documentation

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

Returns the level of the [LogMessage](#).

Returns the level of the [LogMessage](#).

Returns:

The level of the [LogMessage](#).

3.33.3.2 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.

3.33.4 Friends And Related Function Documentation

3.33.4.1 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.

3.33.4.2 std::ostream& operator<< (std::ostream & os, const [LogMessage](#) & message) [friend]

Printing of LogMessages to ostream.

Output operator so that LogMessages can be printed conveniently by LogDestinations.

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

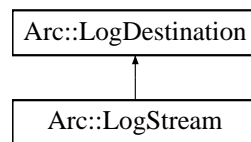
- [Logger.h](#)

3.34 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)

3.34.1 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.

3.34.2 Constructor & Destructor Documentation

3.34.2.1 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 write LogMessages.

3.34.3 Member Function Documentation

3.34.3.1 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:

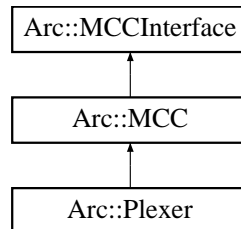
- `Logger.h`

3.35 Arc::MCC Class Reference

[Message](#) Chain Component - base class for every [MCC](#) plugin.

```
#include <MCC.h>
```

Inheritance diagram for Arc::MCC::



Public Member Functions

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

Protected Member Functions

- [Arc::MCCInterface](#) * [Next](#) (const std::string &label="")

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](#)

3.35.1 Detailed Description

[Message](#) Chain Component - base class for every [MCC](#) plugin.

This is partially virtual class which defines interface and common functionality for every [MCC](#) plugin needed for managing of component in a chain.

3.35.2 Constructor & Destructor Documentation

3.35.2.1 Arc::MCC::MCC ([Arc::Config](#) *) [inline]

Example constructor - [MCC](#) takes at least it's configuration subtree

3.35.3 Member Function Documentation

3.35.3.1 `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](#).

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

SecHandler

3.35.3.3 `virtual void Arc::MCC::Unlink (void)` [virtual]

Removing all links. Useful for destroying chains.

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

Dummy [Message](#) processing method. Just a placeholder.

Implements [Arc::MCCInterface](#).

Reimplemented in [Arc::Plexer](#).

3.35.4 Member Data Documentation

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

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

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

Set of labeled authentication and authorization handlers. [MCC](#) calls sequence of handlers at specific point depending on associated identifier. In most cases those are "in" and "out" for incoming and outgoing messages correspondingly.

3.35.4.3 `Arc::Logger Arc::MCC::logger` [static, protected]

A logger for MCCs.

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

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

- [MCC.h](#)

3.36 mcc_descriptor Struct Reference

Identifier of Message Chain Componet (MCC) plugin.

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

Public Attributes

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

3.36.1 Detailed Description

Identifier of Message Chain Componet (MCC) plugin.

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

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

- MCCLoader.h

3.37 Arc::MCC_Status Class Reference

A class for communication of [MCC](#) processing results.

```
#include <MCC_Status.h>
```

Public Member Functions

- [MCC_Status](#) (StatusKind kind=STATUS_UNDEFINED, const std::string &origin="???", const std::string &explanation="No explanation.")
- bool [isOk](#) () const
- StatusKind [getKind](#) () const
- const std::string & [getOrigin](#) () const
- const std::string & [getExplanation](#) () const
- [operator std::string](#) () const
- [operator bool](#) (void) const
- bool [operator!](#) (void) const

3.37.1 Detailed Description

A class for communication of [MCC](#) processing results.

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

3.37.2 Constructor & Destructor Documentation

3.37.2.1 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.")

3.37.3 Member Function Documentation

3.37.3.1 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

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

Returns the status kind.

Returns the status kind of this object.

Returns:

The status kind of this object.

3.37.3.3 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.

3.37.3.4 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.

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

Conversion to string.

This operator converts a [MCC_Status](#) object to a string.

3.37.3.6 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

3.37.3.7 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:

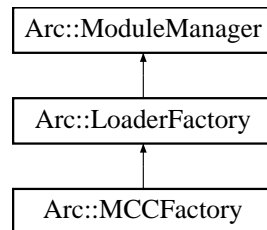
- `MCC_Status.h`

3.38 Arc::MCCFactory Class Reference

MCC Plugins handler.

```
#include <MCCFactory.h>
```

Inheritance diagram for Arc::MCCFactory::



Public Member Functions

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

3.38.1 Detailed Description

MCC Plugins handler.

This class handles shared libraries containing MCCs

3.38.2 Constructor & Destructor Documentation

3.38.2.1 Arc::MCCFactory::MCCFactory ([Config](#) * *cfg*)

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

3.38.3 Member Function Documentation

3.38.3.1 [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:

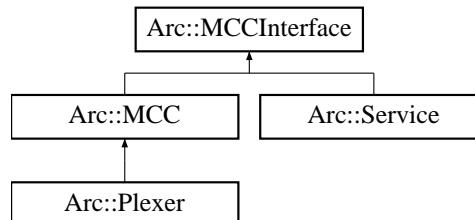
- MCCFactory.h

3.39 Arc::MCCInterface Class Reference

Interface for communication between [MCC](#), [Service](#) and [Plexer](#) objects.

```
#include <MCC.h>
```

Inheritance diagram for Arc::MCCInterface::



Public Member Functions

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

3.39.1 Detailed Description

Interface for communication between [MCC](#), [Service](#) and [Plexer](#) objects.

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

3.39.2 Member Function Documentation

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

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

Parameters:

request The request that needs to be processed.

response A [Message](#) object that will contain the response of the request when the method returns.

Returns:

An object representing the status of the call.

Implemented in [Arc::Plexer](#), and [Arc::MCC](#).

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

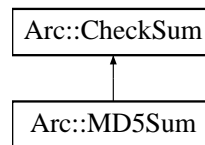
- MCC.h

3.40 Arc::MD5Sum Class Reference

Implementation of MD5 checksum.

```
#include <Checksum.h>
```

Inheritance diagram for Arc::MD5Sum::



Public Member Functions

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

3.40.1 Detailed Description

Implementation of MD5 checksum.

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

- CheckSum.h

3.41 Arc::Message Class Reference

Object being passed through chain of MCCs.

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

Public Member Functions

- [Message](#) (void)
- [Message](#) ([Message](#) &msg)
- [Message](#) (long msg_ptr_addr)
- [~Message](#) (void)
- [Message](#) & [operator=](#) ([Message](#) &msg)
- [MessagePayload](#) * [Payload](#) (void)
- [MessagePayload](#) * [Payload](#) ([MessagePayload](#) *new_payload)
- [MessageAttributes](#) * [Attributes](#) (void)
- void [Attributes](#) ([MessageAttributes](#) *attributes)
- [MessageAuth](#) * [Auth](#) (void)
- void [Auth](#) ([MessageAuth](#) *auth)
- [MessageContext](#) * [Context](#) (void)
- void [Context](#) ([MessageContext](#) *context)

3.41.1 Detailed Description

Object being passed through chain of MCCs.

An instance of this class refers to objects with main content ([MessagePayload](#)), authentication/authorization information ([MessageAuth](#)) and common purpose attributes ([MessageAttributes](#)). [Message](#) class does not manage pointers to objects and their content. It only serves for grouping those objects. [Message](#) objects are supposed to be processed by MCCs and Services implementing [MCCInterface](#) method process(). All objects constituting content of [Message](#) object are subject to following policies:

1. All objects created inside call to process() method using new command must be explicitly 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 explicitly destroyed within that call with following exception. a) Objects which are part of 'response' Method returned from call to next's process() method. Unless those objects are passed further to calling process(), of course.
3. It is not allowed to make 'response' point to same objects as 'request' does on entry to process() method. That is needed to avoid double destruction of same object. (Note: if in a future such need arises it may be solved by storing additional flags in [Message](#) object).
4. It is allowed to change content of pointers of 'request' [Message](#). Calling process() method must not rely on that object to stay intact.
5. Called process() method should either fill 'response' [Message](#) with pointers to valid objects or to keep them intact. This makes it possible for calling process() to preload 'response' with valid error message.

3.41.2 Constructor & Destructor Documentation

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

Dummy constructor

3.41.2.2 Arc::Message::Message (Message & msg) [inline]

Copy constructor. Ensures shallow copy.

3.41.2.3 Arc::Message::Message (long msg_ptr_addr)

Copy constructor. Used by language bindings

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

Destructor does not affect referred objects

3.41.3 Member Function Documentation

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

Assignment. Ensures shallow copy.

3.41.3.2 MessagePayload* Arc::Message::Payload (void) [inline]

Returns pointer to current payload or NULL if no payload assigned.

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

Replaces payload with new one. Returns the old one.

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

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

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

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

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

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

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

Assigns message context object

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

- Message.h

3.42 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_](#)

3.42.1 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.

3.42.2 Constructor & Destructor Documentation

3.42.2.1 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.

3.42.3 Member Function Documentation

3.42.3.1 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.

3.42.3.2 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.

3.42.3.3 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.

3.42.3.4 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.

3.42.3.5 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.

3.42.3.6 `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.

3.42.3.7 `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.

3.42.4 Member Data Documentation**3.42.4.1 `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

3.43 Arc::MessageAuth Class Reference

Contains authenticity information, authorization tokens and decisions.

```
#include <MessageAuth.h>
```

Public Member Functions

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

3.43.1 Detailed Description

Contains authenticity information, authorization tokens and decisions.

Functionality of this class is not defined yet.

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

- MessageAuth.h

3.44 Arc::MessageContext Class Reference

Handler for context of message context.

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

Public Member Functions

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

3.44.1 Detailed Description

Handler for context of message context.

This class is a container for objects derived from [MessageContextElement](#). It gets associated with [Message](#) object usually by first [MCC](#) in a chain and is kept as long as connection persists.

3.44.2 Member Function Documentation

3.44.2.1 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

3.45 Arc::MessageContextElement Class Reference

Top class for elements contained in message context.

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

3.45.1 Detailed Description

Top class for elements contained in message context.

Objects of classes inherited with this one may be stored in [MessageContext](#) container.

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

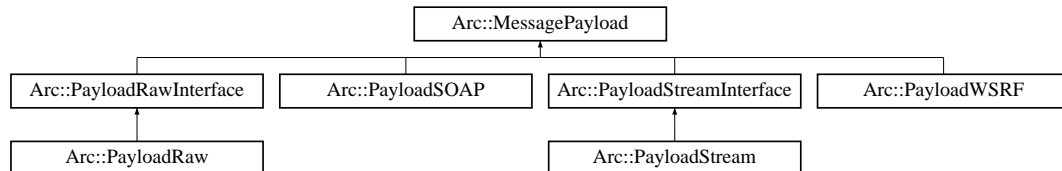
- Message.h

3.46 Arc::MessagePayload Class Reference

Base class for content of message passed through chain.

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

Inheritance diagram for Arc::MessagePayload::



3.46.1 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:

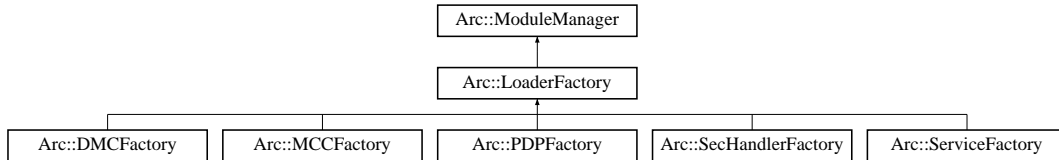
- Message.h

3.47 Arc::ModuleManager Class Reference

Manager of shared libraries.

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

Inheritance diagram for Arc::ModuleManager::



Public Member Functions

- [ModuleManager](#) ([Arc::Config](#) *cfg)
- [Glib::Module](#) * [load](#) (const std::string &name)

3.47.1 Detailed Description

Manager of shared libraries.

This class loads shared libraries/modules. There supposed to be created one instance of it per executable. In such circumstances it would cache handles to loaded modules and not load them multiple times.

3.47.2 Constructor & Destructor Documentation

3.47.2.1 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.

3.47.3 Member Function Documentation

3.47.3.1 [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:

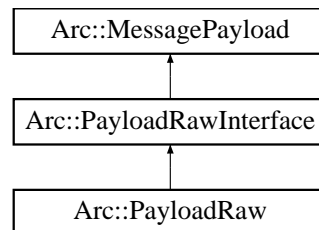
- ModuleManager.h

3.48 Arc::PayloadRaw Class Reference

Raw byte multi-buffer.

```
#include <PayloadRaw.h>
```

Inheritance diagram for Arc::PayloadRaw::



Public Member Functions

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

Protected Attributes

- int [offset_](#)
- int [size_](#)
- std::vector< PayloadRawBuf > [buf_](#)

3.48.1 Detailed Description

Raw byte multi-buffer.

This is implementation of [PayloadRawInterface](#). Buffers are memory blocks logically placed one after another.

3.48.2 Constructor & Destructor Documentation

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

Constructor. Created object contains no buffers.

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

Destructor. Frees allocated buffers.

3.48.3 Member Function Documentation**3.48.3.1]**

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](#).

3.48.3.2 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](#).

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

Returns logical size of whole structure.

Implements [Arc::PayloadRawInterface](#).

3.48.3.4 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](#).

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

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

Implements [Arc::PayloadRawInterface](#).

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

Returns pointer to num'th buffer

Implements [Arc::PayloadRawInterface](#).

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

Returns length of num'th buffer

Implements [Arc::PayloadRawInterface](#).

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

Returns position of num'th buffer

Implements [Arc::PayloadRawInterface](#).

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

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

Implements [Arc::PayloadRawInterface](#).

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

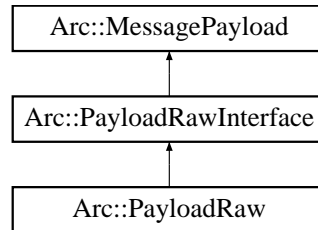
- PayloadRaw.h

3.49 Arc::PayloadRawInterface Class Reference

Random Access Payload for [Message](#) objects.

```
#include <PayloadRaw.h>
```

Inheritance diagram for Arc::PayloadRawInterface::



Public Member Functions

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

3.49.1 Detailed Description

Random Access Payload for [Message](#) objects.

This class is a virtual interface for managing [Message](#) payload with arbitrarily accessible content. Inheriting classes are supposed to implement memory-resident or memory-mapped content made of optionally multiple chunks/buffers. Every buffer has own size and offset. This class is purely virtual.

3.49.2 Member Function Documentation

3.49.2.1 `operator[]`

```
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](#).

3.49.2.2 `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](#).

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

Returns logical size of whole structure.

Implemented in [Arc::PayloadRaw](#).

3.49.2.4 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](#).

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

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

Implemented in [Arc::PayloadRaw](#).

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

Returns pointer to num'th buffer

Implemented in [Arc::PayloadRaw](#).

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

Returns length of num'th buffer

Implemented in [Arc::PayloadRaw](#).

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

Returns position of num'th buffer

Implemented in [Arc::PayloadRaw](#).

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

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

Implemented in [Arc::PayloadRaw](#).

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

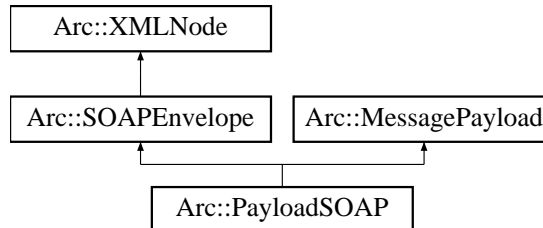
- PayloadRaw.h

3.50 Arc::PayloadSOAP Class Reference

Payload of [Message](#) with SOAP content.

```
#include <PayloadSOAP.h>
```

Inheritance diagram for Arc::PayloadSOAP::



Public Member Functions

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

3.50.1 Detailed Description

Payload of [Message](#) with SOAP content.

This class combines [MessagePayload](#) with [SOAPEnvelope](#) to make it possible to pass SOAP messages through [MCC](#) chain.

3.50.2 Constructor & Destructor Documentation

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

Constructor - creates new [Message](#) payload

3.50.2.2 Arc::PayloadSOAP::PayloadSOAP (const [Arc::SOAPEnvelope](#) & soap)

Constructor - creates [Message](#) payload from SOAP document. Provided SOAP document must exist as long as created object exists.

3.50.2.3 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:

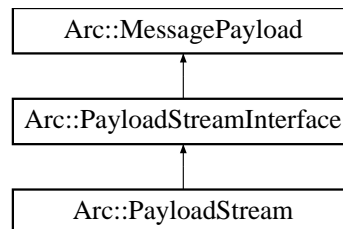
- PayloadSOAP.h

3.51 Arc::PayloadStream Class Reference

POSIX handle as Payload.

```
#include <PayloadStream.h>
```

Inheritance diagram for Arc::PayloadStream::



Public Member Functions

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

Protected Attributes

- int [timeout_](#)
- int [handle_](#)
- bool [seekable_](#)

3.51.1 Detailed Description

POSIX handle as Payload.

This is an implementation of [PayloadStreamInterface](#) for generic POSIX handle.

3.51.2 Constructor & Destructor Documentation

3.51.2.1 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.

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

Destructor.

3.51.3 Member Function Documentation

3.51.3.1 virtual bool Arc::PayloadStream::Get (char * *buf*, int & *size*) [virtual]

Extracts information from stream up to 'size' bytes. 'size' contains number of read bytes on exit. Returns true in case of success.

Implements [Arc::PayloadStreamInterface](#).

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

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

Implements [Arc::PayloadStreamInterface](#).

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

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

Implements [Arc::PayloadStreamInterface](#).

3.51.3.4 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](#).

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

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

Implements [Arc::PayloadStreamInterface](#).

3.51.3.6 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](#).

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

Returns true if stream is valid.

Implements [Arc::PayloadStreamInterface](#).

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

Returns true if stream is invalid.

Implements [Arc::PayloadStreamInterface](#).

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

Query current timeout for [Get\(\)](#) and [Put\(\)](#) operations.

Implements [Arc::PayloadStreamInterface](#).

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

Set current timeout for [Get\(\)](#) and [Put\(\)](#) operations.

Implements [Arc::PayloadStreamInterface](#).

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

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

3.51.4 Member Data Documentation**3.51.4.1 int Arc::PayloadStream::handle_** [protected]

Timeout for read/write operations

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

Handle for operations

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

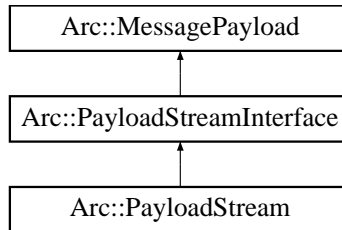
- PayloadStream.h

3.52 Arc::PayloadStreamInterface Class Reference

Stream-like Payload for [Message](#) object.

```
#include <PayloadStream.h>
```

Inheritance diagram for Arc::PayloadStreamInterface::



Public Member Functions

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

3.52.1 Detailed Description

Stream-like Payload for [Message](#) object.

This class is a virtual interface for managing stream-like source and destination. It's supposed to be passed through [MCC](#) chain as payload of [Message](#). It must be treated by MCCs and Services as dynamic payload. This class is purely virtual.

3.52.2 Member Function Documentation

3.52.2.1 virtual bool Arc::PayloadStreamInterface::Get (char * *buf*, int & *size*) [pure virtual]

Extracts information from stream up to 'size' bytes. 'size' contains number of read bytes on exit. Returns true in case of success.

Implemented in [Arc::PayloadStream](#).

3.52.2.2 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](#).

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

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

Implemented in [Arc::PayloadStream](#).

3.52.2.4 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](#).

3.52.2.5 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](#).

3.52.2.6 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](#).

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

Returns true if stream is valid.

Implemented in [Arc::PayloadStream](#).

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

Returns true if stream is invalid.

Implemented in [Arc::PayloadStream](#).

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

Query current timeout for [Get\(\)](#) and [Put\(\)](#) operations.

Implemented in [Arc::PayloadStream](#).

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

Set current timeout for [Get\(\)](#) and [Put\(\)](#) operations.

Implemented in [Arc::PayloadStream](#).

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

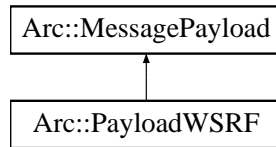
- [PayloadStream.h](#)

3.53 Arc::PayloadWSRF Class Reference

This class combines [MessagePayload](#) with [WSRF](#).

```
#include <PayloadWSRF.h>
```

Inheritance diagram for Arc::PayloadWSRF::



Public Member Functions

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

Protected Attributes

- [WSRF](#) & **wsrf_**
- bool **owner_**

3.53.1 Detailed Description

This class combines [MessagePayload](#) with [WSRF](#).

It's intention is to make it possible to pass [WSRF](#) messages through [MCC](#) chain as one more Payload type.

3.53.2 Constructor & Destructor Documentation

3.53.2.1 Arc::PayloadWSRF::PayloadWSRF (const [SOAPEnvelope](#) & soap)

Constructor - creates [Message](#) payload from SOAP message. Returns invalid [WSRF](#) if SOAP does not represent WS-ResourceProperties

3.53.2.2 Arc::PayloadWSRF::PayloadWSRF ([WSRF](#) & *wsrp*)

Constructor - creates [Message](#) payload with acquired [WSRF](#) message. [WSRF](#) message will be destroyed by destructor of this object.

3.53.2.3 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

3.54 pdp_descriptor Struct Reference

Identifier of Policy Decision Point (PDP) plugin.

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

Public Attributes

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

3.54.1 Detailed Description

Identifier of Policy Decision Point (PDP) plugin.

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

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

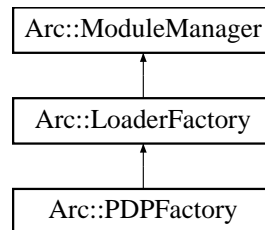
- PDPLoader.h

3.55 Arc::PDPFactory Class Reference

PDP Plugins handler.

```
#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)

3.55.1 Detailed Description

PDP Plugins handler.

This class handles shared libraries containing PDPs

3.55.2 Constructor & Destructor Documentation

3.55.2.1 Arc::PDPFactory::PDPFactory ([Config](#) * cfg)

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

3.55.3 Member Function Documentation

3.55.3.1 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:

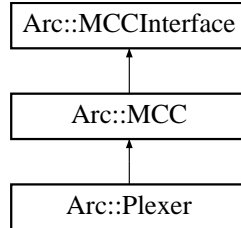
- PDPFactory.h

3.56 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)

3.56.1 Detailed Description

The [Plexer](#) class, used for routing messages to services.

This is the [Plexer](#) class. Its purpose is to route incoming messages to appropriate Services and [MCC](#) chains.

3.56.2 Constructor & Destructor Documentation

3.56.2.1 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.

3.56.2.2 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.

3.56.3 Member Function Documentation

3.56.3.1 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](#).

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

Rout request messages to appropriate services.

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

Reimplemented from [Arc::MCC](#).

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

- [Plexer.h](#)

3.57 Arc::PlexerEntry Class Reference

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

```
#include <Plexer.h>
```

Friends

- class **Plexer**

3.57.1 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

3.58 Arc::RegularExpression Class Reference

A regular expression class.

```
#include <ArcRegex.h>
```

Public Member Functions

- [RegularExpression](#) (std::string pattern)
- [RegularExpression](#) (const [RegularExpression](#) ®ex)
- [~RegularExpression](#) ()
- const [RegularExpression](#) & operator= (const [RegularExpression](#) ®ex)
- 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](#) ()

3.58.1 Detailed Description

A regular expression class.

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

3.58.2 Constructor & Destructor Documentation

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

Creates a regex from a pattern string.

3.58.2.2 Arc::RegularExpression::RegularExpression (const [RegularExpression](#) & *regex*)

Copy constructor.

3.58.2.3 Arc::RegularExpression::~~RegularExpression ()

Destructor.

3.58.3 Member Function Documentation

3.58.3.1 const [RegularExpression](#)& Arc::RegularExpression::operator= (const [RegularExpression](#) & *regex*)

Assignment operator.

3.58.3.2 bool Arc::RegularExpression::isOk ()

Returns true if the pattern of this regex is ok.

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

Returns true if this regex has the pattern provided.

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

Returns true if this regex matches whole string provided.

3.58.3.5 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'.

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

Returns patter.

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

- ArcRegex.h

3.59 sechandler_descriptor Struct Reference

Identifier of SecHandler plugin.

```
#include <SecHandlerLoader.h>
```

Public Attributes

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

3.59.1 Detailed Description

Identifier of SecHandler plugin.

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

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

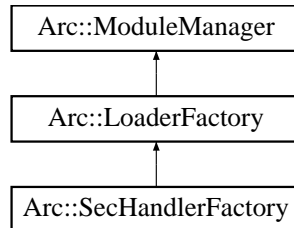
- SecHandlerLoader.h

3.60 Arc::SecHandlerFactory Class Reference

SecHandler Plugins handler.

```
#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)

3.60.1 Detailed Description

SecHandler Plugins handler.

This class handles shared libraries containing SecHandlers

3.60.2 Constructor & Destructor Documentation

3.60.2.1 Arc::SecHandlerFactory::SecHandlerFactory ([Config](#) * cfg)

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

3.60.3 Member Function Documentation

3.60.3.1 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:

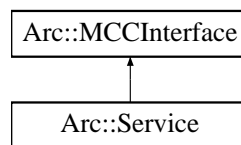
- SecHandlerFactory.h

3.61 Arc::Service Class Reference

[Service](#) - last component in a [Message](#) Chain.

```
#include <Service.h>
```

Inheritance diagram for Arc::Service::



Public Member Functions

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

Protected Attributes

- std::map< std::string, std::list< Arc::SecHandler * > > [sechandlers_](#)

Static Protected Attributes

- static [Logger](#) [logger](#)

3.61.1 Detailed Description

[Service](#) - last component in a [Message](#) Chain.

This is virtual class which defines interface (in a future also common functionality) for every [Service](#) plugin. Interface is made of method [process\(\)](#) which is called by [Plexer](#) or [MCC](#) class. There is one [Service](#) object created for every service description processed by [Loader](#) class objects. Classes derived from [Service](#) class must implement [process\(\)](#) method of [MCCInterface](#). It is up to developer how internal state of service is stored and communicated to other services and external utilities. [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 be linked to SOAP [MCC](#) it must accept and generate messages with [PayloadSOAP](#) payload. Method [process\(\)](#) of class derived from [Service](#) class may be called concurrently in multiple threads. Developers must take that into account and write thread-safe implementation. Simple example of service is provided in /src/tests/echo/echo.cpp of source tree. The way to write client counterpart of corresponding service is undefined yet. For example see /src/tests/echo/test.cpp

3.61.2 Constructor & Destructor Documentation

3.61.2.1 Arc::Service::Service ([Arc::Config](#) *) [inline]

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

3.61.3 Member Function Documentation

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

SecHandler

3.61.4 Member Data Documentation

3.61.4.1 `std::map<std::string,std::list<Arc::SecHandler*> >` [Arc::Service::sechandlers_](#)
[protected]

Set of labeled authentication and authorization handlers. [MCC](#) calls sequence of handlers at specific point depending on associated identifier. in most aces those are "in" and "out" for incoming and outgoing messages correspondingly.

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

- Service.h

3.62 service_descriptor Struct Reference

Identifier of Service plugin.

```
#include <ServiceLoader.h>
```

Public Attributes

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

3.62.1 Detailed Description

Identifier of Service plugin.

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

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

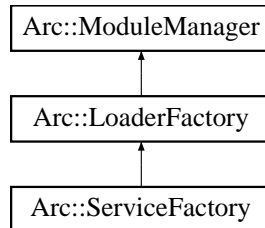
- ServiceLoader.h

3.63 Arc::ServiceFactory Class Reference

[Service](#) Plugins handler.

```
#include <ServiceFactory.h>
```

Inheritance diagram for Arc::ServiceFactory::



Public Member Functions

- [ServiceFactory](#) ([Config](#) *cfg)
- [Service](#) * [get_instance](#) (const std::string &name, [Config](#) *cfg, [ChainContext](#) *ctx)
- [Service](#) * [get_instance](#) (const std::string &name, int version, [Config](#) *cfg, [ChainContext](#) *ctx)
- [Service](#) * [get_instance](#) (const std::string &name, int min_version, int max_version, [Config](#) *cfg, [ChainContext](#) *ctx)

3.63.1 Detailed Description

[Service](#) Plugins handler.

This class handles shared libraries containing Services

3.63.2 Constructor & Destructor Documentation

3.63.2.1 Arc::ServiceFactory::ServiceFactory ([Config](#) * cfg)

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

3.63.3 Member Function Documentation

3.63.3.1 [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

3.64 Arc::SimpleCondition Class Reference

Simple triggered condition.

```
#include <Thread.h>
```

Public Member Functions

- void [lock](#) (void)
- void [unlock](#) (void)
- void [signal](#) (void)
- void [signal_nonblock](#) (void)
- void [broadcast](#) (void)
- void [wait](#) (void)
- void [wait_nonblock](#) (void)
- void [wait](#) (int t)
- void [reset](#) (void)

3.64.1 Detailed Description

Simple triggered condition.

Provides condition and semaphor objects in one element.

3.64.2 Member Function Documentation

3.64.2.1 void Arc::SimpleCondition::lock (void) [inline]

Acquire semaphor

3.64.2.2 void Arc::SimpleCondition::unlock (void) [inline]

Release semaphor

3.64.2.3 void Arc::SimpleCondition::signal (void) [inline]

Signal about condition

3.64.2.4 void Arc::SimpleCondition::signal_nonblock (void) [inline]

Signal about condition without using semaphor

3.64.2.5 void Arc::SimpleCondition::broadcast (void) [inline]

Signal about condition to all waiting threads

3.64.2.6 void Arc::SimpleCondition::wait (void) [inline]

Wait for condition

3.64.2.7 void Arc::SimpleCondition::wait_nonblock (void) [inline]

Wait for condition without using semaphor

3.64.2.8 void Arc::SimpleCondition::wait (int t) [inline]

Wait for condition no longer than t milliseconds

3.64.2.9 void Arc::SimpleCondition::reset (void) [inline]

Reset object to initial state

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

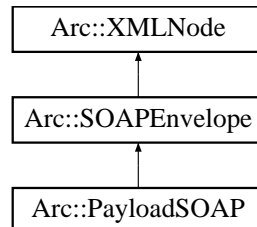
- Thread.h

3.65 Arc::SOAPEnvelope Class Reference

Extends [XMLNode](#) class to support structures of SOAP message.

```
#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)

3.65.1 Detailed Description

Extends [XMLNode](#) class to support structures of SOAP message.

All [XMLNode](#) methods are exposed by inheriting from [XMLNode](#) and node itself is translated into Envelope part of SOAP.

3.65.2 Constructor & Destructor Documentation

3.65.2.1 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.

3.65.2.2 Arc::SOAPEnvelope::SOAPEnvelope (const char * xml, int len = -1)

Same as previous

3.65.2.3 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.

3.65.2.4 Arc::SOAPEnvelope::SOAPEnvelope (XMLNode doc)

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

3.65.3 Member Function Documentation

3.65.3.1 SOAPEnvelope* Arc::SOAPEnvelope::New (void)

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

3.65.3.2 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](#).

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

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

Reimplemented from [Arc::XMLNode](#).

3.65.3.4 XMLNode Arc::SOAPEnvelope::Header (void) [inline]

Get SOAP header as XML node

3.65.3.5 bool Arc::SOAPEnvelope::IsFault (void) [inline]

Returns true if message is Fault

3.65.3.6 SOAPFault* Arc::SOAPEnvelope::Fault (void) [inline]

Get Fault part of message. Returns NULL if message is not Fault.

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

- SOAPEnvelope.h

3.66 Arc::SOAPFault Class Reference

Interface to SOAP Fault message.

```
#include <SOAPEnvelope.h>
```

Public Types

- 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](#)

3.66.1 Detailed Description

Interface to SOAP Fault message.

[SOAPFault](#) class provides a convenience interface for accessing elements of SOAP faults. It also tries to expose single interface for both version 1.0 and 1.2 faults. This class is not intended to 'own' any information stored. It's purpose is to manipulate information which is kept under control of [XMLNode](#) or [SOAPEnvelope](#) classes. If instance does not refer to valid SOAP Fault structure all manipulation methods will have no effect.

3.66.2 Member Enumeration Documentation

3.66.2.1 enum [Arc::SOAPFault::SOAPFaultCode](#)

Fault codes of SOAP specs

3.66.3 Constructor & Destructor Documentation

3.66.3.1 Arc::SOAPFault::SOAPFault ([XMLNode](#) & *body*)

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

3.66.4 Member Function Documentation

3.66.4.1 Arc::SOAPFault::operator bool (void) [inline]

Returns true if instance refers to SOAP Fault

3.66.4.2 [SOAPFaultCode](#) Arc::SOAPFault::Code (void)

Returns Fault Code element

3.66.4.3 void Arc::SOAPFault::Code ([SOAPFaultCode](#) *code*)

Set Fault Code element

3.66.4.4 std::string Arc::SOAPFault::Subcode (int *level*)

Returns Fault Subcode element at various levels (0 is for Code)

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

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

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

Returns content of Fault Reason element at various levels

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

Set Fault Reason content at various levels to 'reason'

3.66.4.8 void Arc::SOAPFault::Reason (const char * *reason*) [inline]

Set Fault Reason element at top level

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

Returns content of Fault Node element

3.66.4.10 void Arc::SOAPFault::Node (const char * *node*)

Set content of Fault Node element to 'node'

3.66.4.11 `std::string Arc::SOAPFault::Role (void)`

Returns content of Fault Role element

3.66.4.12 `void Arc::SOAPFault::Role (const char * role)`

Set content of Fault Role element to '*role*'

3.66.4.13 [XMLNode](#) `Arc::SOAPFault::Detail (bool create = false)`

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

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

- SOAPEnvelope.h

3.67 Arc::SOAPMessage Class Reference

[Message](#) restricted to SOAP payload.

```
#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)

3.67.1 Detailed Description

[Message](#) restricted to SOAP payload.

This is a special [Message](#) intended to be used in language bindings for programming languages which are not flexible enough to support all kinds of Payloads. It is passed through chain of MCCs and works like the [Message](#) but can carry only SOAP content.

3.67.2 Constructor & Destructor Documentation

3.67.2.1 Arc::SOAPMessage::SOAPMessage (void) [inline]

Dummy constructor

3.67.2.2 Arc::SOAPMessage::SOAPMessage (long msg_ptr_addr)

Copy constructor. Used by language bindings

3.67.2.3 Arc::SOAPMessage::SOAPMessage ([SOAPMessage](#) & msg) [inline]

Copy constructor. Ensures shallow copy.

3.67.2.4 Arc::SOAPMessage::SOAPMessage ([Arc::Message](#) & msg)

Copy constructor. Ensures shallow copy.

3.67.2.5 Arc::SOAPMessage::~~SOAPMessage (void) [inline]

Destructor does not affect refered objects

3.67.3 Member Function Documentation**3.67.3.1 SOAPMessage& Arc::SOAPMessage::operator= (SOAPMessage & msg)** [inline]

Assignment. Ensures shallow copy.

3.67.3.2 Arc::PayloadSOAP* Arc::SOAPMessage::Payload (void) [inline]

Returns pointer to current payload or NULL if no payload assigned.

3.67.3.3 Arc::PayloadSOAP* Arc::SOAPMessage::Payload (Arc::PayloadSOAP * new_payload)
[inline]

Replace payload with new one

3.67.3.4 Arc::MessageAttributes* Arc::SOAPMessage::Attributes (void) [inline]

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

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

- SOAPMessage.h

3.68 Arc::Time Class Reference

A class for storing and manipulating times.

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

Public Member Functions

- [Time](#) ()
- [Time](#) (const time_t &)
- [Time](#) (const std::string &)
- [Time](#) & [operator=](#) (const time_t &)
- [Time](#) & [operator=](#) (const [Time](#) &)
- void [SetTime](#) (const time_t &)
- time_t [GetTime](#) () const
- [operator std::string](#) () const
- std::string [str](#) (const TimeFormat &=time_format) const
- bool [operator<](#) (const [Time](#) &) const
- bool [operator>](#) (const [Time](#) &) const
- bool [operator<=](#) (const [Time](#) &) const
- bool [operator>=](#) (const [Time](#) &) const
- bool [operator==](#) (const [Time](#) &) const
- bool [operator!=](#) (const [Time](#) &) const
- [Time](#) [operator+](#) (const Period &) const
- [Time](#) [operator-](#) (const Period &) const

Static Public Member Functions

- static void [SetFormat](#) (const TimeFormat &)
- static TimeFormat [GetFormat](#) ()

3.68.1 Detailed Description

A class for storing and manipulating times.

3.68.2 Constructor & Destructor Documentation

3.68.2.1 Arc::Time::Time ()

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

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

Constructor that takes a time_t variable and stores it.

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

Constructor that tries to convert a string into a time_t.

3.68.3 Member Function Documentation

3.68.3.1 **Time& Arc::Time::operator= (const time_t &)**

Assignment operator from a time_t.

3.68.3.2 **Time& Arc::Time::operator= (const Time &)**

Assignment operator from a Time.

3.68.3.3 **void Arc::Time::SetTime (const time_t &)**

sets the time

3.68.3.4 **time_t Arc::Time::GetTime () const**

gets the time

3.68.3.5 **Arc::Time::operator std::string () const**

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

3.68.3.6 **std::string Arc::Time::str (const TimeFormat & = time_format) const**

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

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

Sets the default format for time strings.

3.68.3.8 **static TimeFormat Arc::Time::GetFormat () [static]**

Gets the default format for time strings.

3.68.3.9 **bool Arc::Time::operator< (const Time &) const**

Comparing two Time objects.

3.68.3.10 **bool Arc::Time::operator> (const Time &) const**

Comparing two Time objects.

3.68.3.11 **bool Arc::Time::operator<= (const Time &) const**

Comparing two Time objects.

3.68.3.12 `bool Arc::Time::operator>= (const Time &) const`

Comparing two `Time` objects.

3.68.3.13 `bool Arc::Time::operator== (const Time &) const`

Comparing two `Time` objects.

3.68.3.14 `bool Arc::Time::operator!= (const Time &) const`

Comparing two `Time` objects.

3.68.3.15 `Time Arc::Time::operator+ (const Period &) const`

Adding `Time` object with `Period` object.

3.68.3.16 `Time Arc::Time::operator- (const Period &) const`

Subtracting `Period` object from `Time` object.

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

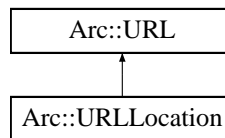
- `DateTime.h`

3.69 Arc::URL Class Reference

Class to hold general URL's.

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

Inheritance diagram for Arc::URL::



Public Member Functions

- [URL](#) ()
- [URL](#) (const std::string &url)
- virtual [~URL](#) ()
- const std::string & [Protocol](#) () const
- 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 [fullstr](#) () const
- virtual std::string [ConnectionURL](#) () const
- bool [operator<](#) (const [URL](#) &url) const
- bool [operator==](#) (const [URL](#) &url) const
- [operator bool](#) () const
- bool [operator!](#) () const

Static Protected Member Functions

- static std::string [BaseDN2Path](#) (const std::string &)
- static std::string [Path2BaseDN](#) (const std::string &)

Protected Attributes

- std::string [protocol](#)
- std::string [username](#)
- std::string [passwd](#)
- std::string [host](#)
- int [port](#)
- std::string [path](#)
- std::map< std::string, std::string > [httpoptions](#)
- std::map< std::string, std::string > [urloptions](#)
- std::list< [URLLocation](#) > [locations](#)
- std::map< std::string, std::string > [commonlocoptions](#)

Friends

- std::ostream & [operator<<](#) (std::ostream &out, const [URL](#) &u)

3.69.1 Detailed Description

Class to hold general URL's.

A [URL](#) is constructed from a string representation and split into protocol, hostname, port and path.

3.69.2 Constructor & Destructor Documentation

3.69.2.1 Arc::URL::URL ()

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

3.69.2.2 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.

3.69.2.3 virtual Arc::URL::~~URL () [virtual]

[URL](#) Destructor

3.69.3 Member Function Documentation

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

Returns the protocol of the [URL](#).

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

Returns the username of the [URL](#).

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

Returns the password of the [URL](#).

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

Returns the hostname of the [URL](#).

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

Returns the port of the [URL](#).

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

Returns the path of the [URL](#).

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

In case of ldap-protocol, return the basedn of the [URL](#).

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

Returns HTTP options if any.

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

Returns the value of an HTTP option.

Parameters:

option The option whose value is returned.

undefined This value is returned if the HTTP option is not defined.

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

Returns [URL](#) options if any.

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

Returns the value of a [URL](#) option.

Parameters:

option The option whose value is returned.

undefined This value is returned if the [URL](#) option is not defined.

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

Adds a [URL](#) option.

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

Returns the locations if any.

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

Returns the common location options if any.

3.69.3.15 `const std::string& Arc::URL::CommonLocOption (const std::string & option, const std::string & undefined = "") const`

Returns the value of a common location option.

Parameters:

option The option whose value is returned.

undefined This value is returned if the common location option is not defined.

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

Returns a string representation of the [URL](#).

Reimplemented in [Arc::URLLocation](#).

3.69.3.17 `virtual std::string Arc::URL::fullstr () const` [virtual]

Returns a string representation including options and locations

Reimplemented in [Arc::URLLocation](#).

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

Returns a string representation with protocol, host and port only

3.69.3.19 `bool Arc::URL::operator< (const URL & url) const`

Compares one [URL](#) to another

3.69.3.20 `bool Arc::URL::operator== (const URL & url) const`

Is one [URL](#) equal to another?

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

Check if instance holds valid [URL](#)

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

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

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

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

3.69.4 Friends And Related Function Documentation

3.69.4.1 std::ostream& operator<< (std::ostream & out, const [URL](#) & u) [friend]

Overloaded operator << to print a [URL](#).

3.69.5 Member Data Documentation

3.69.5.1 std::string [Arc::URL::protocol](#) [protected]

the url protocol.

3.69.5.2 std::string [Arc::URL::username](#) [protected]

username of the url.

3.69.5.3 std::string [Arc::URL::passwd](#) [protected]

password of the url.

3.69.5.4 std::string [Arc::URL::host](#) [protected]

hostname of the url.

3.69.5.5 int [Arc::URL::port](#) [protected]

portnumber of the url.

3.69.5.6 std::string [Arc::URL::path](#) [protected]

the url path.

3.69.5.7 `std::map<std::string, std::string>` [Arc::URL::httpoptions](#) [protected]

http-options of the url.

3.69.5.8 `std::map<std::string, std::string>` [Arc::URL::urloptions](#) [protected]

options of the url.

3.69.5.9 `std::list<URLLocation>` [Arc::URL::locations](#) [protected]

locations for index server URLs.

3.69.5.10 `std::map<std::string, std::string>` [Arc::URL::commonlocoptions](#) [protected]

common location options for index server URLs.

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

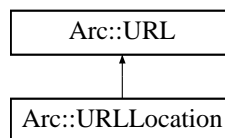
- URL.h

3.70 Arc::URLLocation Class Reference

Class to hold a resolved [URL](#) location.

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

Inheritance diagram for Arc::URLLocation::



Public Member Functions

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

Protected Attributes

- std::string [name](#)

3.70.1 Detailed Description

Class to hold a resolved [URL](#) location.

It is specific for an RC or RLS registration.

3.70.2 Constructor & Destructor Documentation

3.70.2.1 Arc::URLLocation::URLLocation (const std::string & url)

Creates a [URL](#) Location from a [URL](#).

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

Creates a [URL](#) Location from a name and an option string.

3.70.2.3 virtual Arc::URLLocation::~~URLLocation () [virtual]

[URL](#) Location destructor.

3.70.3 Member Function Documentation

3.70.3.1 `std::string Arc::URLLocation::Name () const`

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

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

Returns a string representation of the [URL](#) Location.

Reimplemented from [Arc::URL](#).

3.70.3.3 `virtual std::string Arc::URLLocation::fullstr () const` [virtual]

Returns a string representation including options and locations

Reimplemented from [Arc::URL](#).

3.70.4 Member Data Documentation

3.70.4.1 `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](#)

3.71 Arc::WSAEndpointReference Class Reference

Interface for manipulation of WS-Adressing Endpoint Reference.

```
#include <WSA.h>
```

Public Member Functions

- [WSAEndpointReference](#) ([XMLNode](#) epr)
- [WSAEndpointReference](#) (const std::string &address)
- [WSAEndpointReference](#) (void)
- [~WSAEndpointReference](#) (void)
- std::string [Address](#) (void) const
- void [Address](#) (const std::string &uri)
- [WSAEndpointReference](#) & [operator=](#) (const std::string &address)
- [XMLNode](#) [ReferenceParameters](#) (void)
- [XMLNode](#) [MetaData](#) (void)
- [operator XMLNode](#) (void)

Protected Attributes

- [XMLNode](#) epr_

3.71.1 Detailed Description

Interface for manipulation of WS-Adressing Endpoint Reference.

It works on Endpoint Reference stored in XML tree. No information is stored in this object except reference to corresponding XML subtree.

3.71.2 Constructor & Destructor Documentation

3.71.2.1 Arc::WSAEndpointReference::WSAEndpointReference ([XMLNode](#) epr)

Linking to existing EPR in XML tree

3.71.2.2 Arc::WSAEndpointReference::WSAEndpointReference (const std::string & address)

Creating independent EPR - not implemented

3.71.2.3 Arc::WSAEndpointReference::WSAEndpointReference (void)

Dummy constructor - creates invalid instance

3.71.2.4 Arc::WSAEndpointReference::~~WSAEndpointReference (void)

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

3.71.3 Member Function Documentation

3.71.3.1 `std::string Arc::WSAEndpointReference::Address (void) const`

Returns Address ([URL](#)) encoded in EPR

3.71.3.2 `void Arc::WSAEndpointReference::Address (const std::string & uri)`

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

3.71.3.3 [WSAEndpointReference](#)& `Arc::WSAEndpointReference::operator= (const std::string & address)`

Same as Address(uri)

3.71.3.4 [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.

3.71.3.5 [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.

3.71.3.6 `Arc::WSAEndpointReference::operator XMLNode (void)`

Returns reference to EPR top XML node

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

- WSA.h

3.72 Arc::WSAHeader Class Reference

Interface for manipulation WS-Addressing information in SOAP header.

```
#include <WSA.h>
```

Public Member Functions

- [WSAHeader](#) ([SOAPEnvelope](#) &soap)
- [WSAHeader](#) (const std::string &action)
- std::string [To](#) (void) const
- void [To](#) (const std::string &uri)
- [WSAEndpointReference From](#) (void)
- [WSAEndpointReference ReplyTo](#) (void)
- [WSAEndpointReference FaultTo](#) (void)
- std::string [Action](#) (void) const
- void [Action](#) (const std::string &uri)
- std::string [MessageID](#) (void) const
- void [MessageID](#) (const std::string &uri)
- std::string [RelatesTo](#) (void) const
- void [RelatesTo](#) (const std::string &uri)
- std::string [RelationshipType](#) (void) const
- void [RelationshipType](#) (const std::string &uri)
- [XMLNode ReferenceParameter](#) (int n)
- [XMLNode ReferenceParameter](#) (const std::string &name)
- [XMLNode NewReferenceParameter](#) (const std::string &name)
- [operator XMLNode](#) (void)

Static Public Member Functions

- static bool [Check](#) ([SOAPEnvelope](#) &soap)

Protected Attributes

- [XMLNode header_](#)
- bool [header_allocated_](#)

3.72.1 Detailed Description

Interface for manipulation WS-Addressing information in SOAP header.

It works on Endpoint Reference stored in XML tree. No information is stored in this object except reference to corresponding XML subtree.

3.72.2 Constructor & Destructor Documentation

3.72.2.1 Arc::WSAHeader::WSAHeader ([SOAPEnvelope](#) & soap)

Linking to a header of existing SOAP message

3.72.2.2 Arc::WSAHeader::WSAHeader (const std::string & action)

Creating independent SOAP header - not implemented

3.72.3 Member Function Documentation**3.72.3.1 std::string Arc::WSAHeader::To (void) const**

Returns content of To element of SOAP Header.

3.72.3.2 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.

3.72.3.3 [WSAEndpointReference](#) Arc::WSAHeader::From (void)

Returns From element of SOAP Header. If such element does not exist it's created. Obtained element may be manipulated.

3.72.3.4 [WSAEndpointReference](#) Arc::WSAHeader::ReplyTo (void)

Returns ReplyTo element of SOAP Header. If such element does not exist it's created. Obtained element may be manipulated.

3.72.3.5 [WSAEndpointReference](#) Arc::WSAHeader::FaultTo (void)

Returns FaultTo element of SOAP Header. If such element does not exist it's created. Obtained element may be manipulated.

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

Returns content of Action element of SOAP Header.

3.72.3.7 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.

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

Returns content of MessageID element of SOAP Header.

3.72.3.9 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.

3.72.3.10 `std::string Arc::WSAHeader::RelatesTo (void) const`

Returns content of RelatesTo element of SOAP Header.

3.72.3.11 `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.

3.72.3.12 `std::string Arc::WSAHeader::RelationshipType (void) const`

Returns content of RelationshipType element of SOAP Header.

3.72.3.13 `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.

3.72.3.14 `XMLNode Arc::WSAHeader::ReferenceParameter (int n)`

Return n-th ReferenceParameter element

3.72.3.15 `XMLNode Arc::WSAHeader::ReferenceParameter (const std::string & name)`

Returns first ReferenceParameter element with specified name

3.72.3.16 `XMLNode Arc::WSAHeader::NewReferenceParameter (const std::string & name)`

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

3.72.3.17 `Arc::WSAHeader::operator XMLNode (void)`

Returns reference to SOAP Header - not implemented

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

Tells if specified SOAP message has WSA header

3.72.4 Member Data Documentation**3.72.4.1** `bool Arc::WSAHeader::header_allocated_ [protected]`

SOAP header element

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

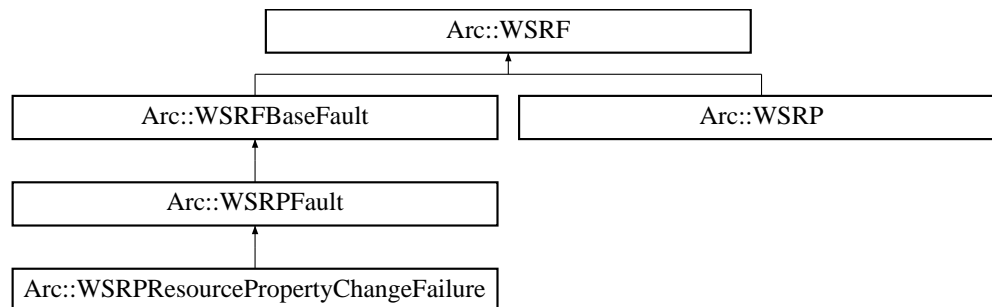
- WSA.h

3.73 Arc::WSRF Class Reference

Base class for every [WSRF](#) message.

```
#include <WSRF.h>
```

Inheritance diagram for Arc::WSRF::



Public Member Functions

- [WSRF](#) ([SOAPEnvelope](#) &soap, const std::string &action="")
- [WSRF](#) (bool fault=false, const std::string &action="")
- virtual [SOAPEnvelope](#) & [SOAP](#) (void)
- virtual [operator bool](#) (void)
- virtual bool **operator!** (void)

Protected Member Functions

- void [set_namespaces](#) (void)

Protected Attributes

- [SOAPEnvelope](#) & soap_
- bool [allocated_](#)
- bool [valid_](#)

3.73.1 Detailed Description

Base class for every [WSRF](#) message.

This class is not intended to be used directly. Use it like reference while passing through unknown [WSRF](#) message or use classes derived from it.

3.73.2 Constructor & Destructor Documentation

3.73.2.1 Arc::WSRF::WSRF ([SOAPEnvelope](#) & soap, const std::string & action = "")

Constructor - creates object out of supplied SOAP tree.

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

Constructor - creates new [WSRF](#) object

3.73.3 Member Function Documentation

3.73.3.1 void Arc::WSRF::set_namespaces (void) [protected]

set WS Resource namespaces and default prefixes in SOAP message

Reimplemented in [Arc::WSRP](#), and [Arc::WSRFBBaseFault](#).

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

Direct access to underlying SOAP element

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

Returns true if instance is valid

3.73.4 Member Data Documentation

3.73.4.1 bool Arc::WSRF::allocated_ [protected]

Associated SOAP message - it's SOAP message after all

3.73.4.2 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:

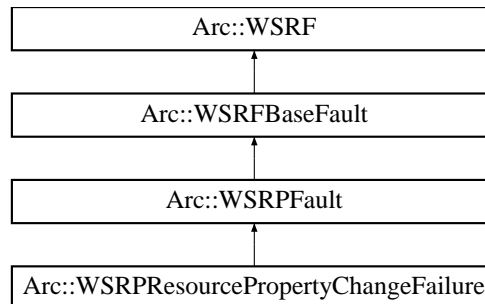
- WSRF.h

3.74 Arc::WSRFBaseFault Class Reference

Base class for [WSRF](#) fault messages.

```
#include <WSRFBaseFault.h>
```

Inheritance diagram for Arc::WSRFBaseFault::



Public Member Functions

- [WSRFBaseFault](#) ([SOAPEnvelope](#) &soap)
- [WSRFBaseFault](#) (const std::string &type)
- std::string **Type** (void)
- [Time](#) **Timestamp** (void)
- void **Timestamp** ([Time](#))
- [WSAEndpointReference](#) **Originator** (void)
- void **ErrorCode** (const std::string &dialect, const [XMLNode](#) &error)
- [XMLNode](#) **ErrorCode** (void)
- std::string **ErrorCodeDialect** (void)
- void **Description** (int pos, const std::string &desc, const std::string &lang)
- std::string **Description** (int pos)
- std::string **DescriptionLang** (int pos)
- void **FaultCause** (int pos, const [XMLNode](#) &cause)
- [XMLNode](#) **FaultCause** (int pos)

Protected Member Functions

- void [set_namespaces](#) (void)

3.74.1 Detailed Description

Base class for [WSRF](#) fault messages.

Use classes inherited from it for specific faults.

3.74.2 Constructor & Destructor Documentation

3.74.2.1 Arc::WSRFBaseFault::WSRFBaseFault ([SOAPEnvelope](#) & soap)

Constructor - creates object out of supplied SOAP tree.

3.74.2.2 Arc::WSRFBaseFault::WSRFBaseFault (const std::string & *type*)

Constructor - creates new [WSRF](#) fault

3.74.3 Member Function Documentation

3.74.3.1 void Arc::WSRFBaseFault::set_namespaces (void) [protected]

set WS-ResourceProperties namespaces and default prefixes in SOAP message

Reimplemented from [Arc::WSRF](#).

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

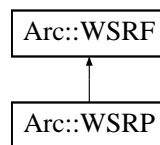
- WSRFBaseFault.h

3.75 Arc::WSRP Class Reference

Base class for WS-ResourceProperties structures.

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

Inheritance diagram for Arc::WSRP::



Public Member Functions

- [WSRP](#) (bool fault=false, const std::string &action="")
- [WSRP](#) ([SOAPEnvelope](#) &soap, const std::string &action="")

Protected Member Functions

- void [set_namespaces](#) (void)

3.75.1 Detailed Description

Base class for WS-ResourceProperties structures.

Inheriting classes implement specific WS-ResourceProperties messages and their properties/elements. Refer to WS-ResourceProperties specifications for things specific to every message.

3.75.2 Constructor & Destructor Documentation

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

Constructor - prepares object for creation of new [WSRP](#) request/response/fault

3.75.2.2 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.

3.75.3 Member Function Documentation

3.75.3.1 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:

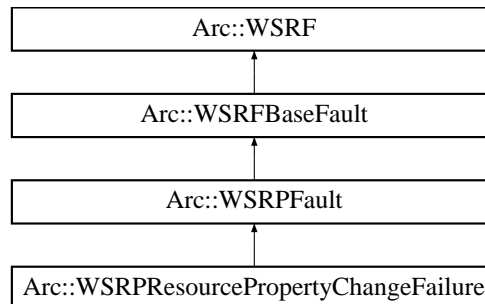
- [WSResourceProperties.h](#)

3.76 Arc::WSRPFault Class Reference

Base class for WS-ResourceProperties faults.

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

Inheritance diagram for Arc::WSRPFault::



Public Member Functions

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

3.76.1 Detailed Description

Base class for WS-ResourceProperties faults.

3.76.2 Constructor & Destructor Documentation

3.76.2.1 Arc::WSRPFault::WSRPFault ([SOAPEnvelope](#) & soap)

Constructor - creates object out of supplied SOAP tree.

3.76.2.2 Arc::WSRPFault::WSRPFault (const std::string & type)

Constructor - creates new [WSRP](#) fault

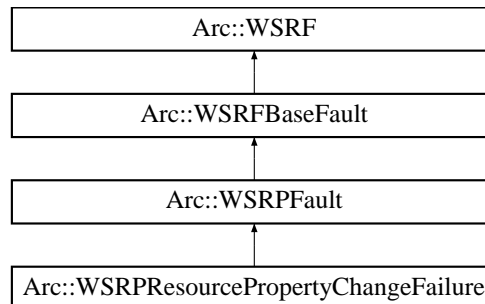
The documentation for this class was generated from the following file:

- WSResourceProperties.h

3.77 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)

3.77.1 Detailed Description

Base class for WS-ResourceProperties faults which contain ResourcePropertyChangeFailure

3.77.2 Constructor & Destructor Documentation

3.77.2.1 Arc::WSRPResourcePropertyChangeFailure::WSRPResourcePropertyChangeFailure ([SOAPEnvelope](#) & soap) [inline]

Constructor - creates object out of supplied SOAP tree.

3.77.2.2 Arc::WSRPResourcePropertyChangeFailure::WSRPResourcePropertyChangeFailure (const std::string & type) [inline]

Constructor - creates new [WSRP](#) fault

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

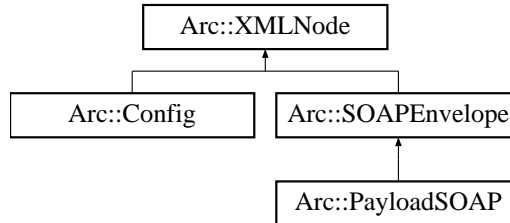
- WSResourceProperties.h

3.78 Arc::XMLNode Class Reference

Wrapper for LibXML library Tree interface.

```
#include <XMLNode.h>
```

Inheritance diagram for Arc::XMLNode::



Public Member Functions

- [XMLNode](#) (void)
- [XMLNode](#) (const [XMLNode](#) &node)
- [XMLNode](#) (const std::string &xml)
- [XMLNode](#) (const char *xml, int len=-1)
- [XMLNode](#) (const Arc::NS &ns)
- [~XMLNode](#) (void)
- void [New](#) ([XMLNode](#) &new_node)
- [operator bool](#) (void) const
- [operator!](#) (void) const
- [XMLNode Child](#) (int n=0) const
- [XMLNode operator\[\]](#) (const char *name) const
- [XMLNode operator\[\]](#) (const std::string &name) const
- [XMLNode operator\[\]](#) (int n) const
- int [Size](#) (void) const
- [XMLNode Get](#) (const std::string &name) const
- std::string [Name](#) (void) const
- 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)
- void [Set](#) (const std::string &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)
- [XMLNode GetRoot](#) (void)

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)

3.78.1 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.

3.78.2 Constructor & Destructor Documentation

3.78.2.1 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.

3.78.2.2 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.

3.78.2.3 Arc::XMLNode::XMLNode (const [XMLNode](#) & node) [inline]

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

3.78.2.4 Arc::XMLNode::XMLNode (const std::string & xml)

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

3.78.2.5 Arc::XMLNode::XMLNode (const char * xml, int len = -1)

Same as previous

3.78.2.6 Arc::XMLNode::XMLNode (const Arc::NS & ns)

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

3.78.2.7 Arc::XMLNode::~~XMLNode (void)

Destructor Also destroys underlying XML document if owned by this instance

3.78.3 Member Function Documentation**3.78.3.1 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.

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

Returns true if instance points to XML element - valid instance

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

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

3.78.3.4 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 XMLNode instance is returned

3.78.3.5]

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

3.78.3.6 [\]](#)

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

Similar to previous method

3.78.3.7 [\]](#)

[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]

3.78.3.8 **int** Arc::XMLNode::Size (void) const [inline]

Returns number of children nodes

3.78.3.9 [XMLNode](#) Arc::XMLNode::Get (const std::string & *name*) const [inline]

More common get

3.78.3.10 **std::string** Arc::XMLNode::Name (void) const [inline]

Returns name of XML node

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

Assign new name to XML node

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

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

Reimplemented in [Arc::SOAPEnvelope](#).

3.78.3.13 **Arc::XMLNode::operator** std::string (void) const [inline]

Returns textual content of node excluding content of children nodes

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

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

3.78.3.15 [XMLNode&](#) Arc::XMLNode::operator= (const char * *content*) [inline]

Same as previous method

3.78.3.16 void Arc::XMLNode::Set (const std::string & *content*) [inline]

Common set method

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

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

3.78.3.18 XMLNode Arc::XMLNode::Attribute (int *n* = 0)

Returns XMLNode instance representing n-th attribute of node.

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

Creates new attribute with specified name.

3.78.3.20 XMLNode Arc::XMLNode::NewAttribute (const char * *name*)

Same as previous method

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

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

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

Returns number of attributes of node

3.78.3.23 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 useful to apply this method to XML being processed in order to refer to its elements by known prefix.

Reimplemented in Arc::SOAPEnvelope.

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

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

3.78.3.25 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

3.78.3.26 [XMLNode](#) Arc::XMLNode::NewChild (const char * *name*, int *n* = -1, bool *global_order* = false)

Same as previous method

3.78.3.27 [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 referring to new child. XML element is a copy of supplied one but not owned by returned instance

3.78.3.28 void Arc::XMLNode::Replace (const [XMLNode](#) & *node*)

Makes a copy of supplied node and place it to this one

3.78.3.29 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

3.78.3.30 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").

3.78.3.31 [XMLNode](#) Arc::XMLNode::GetRoot (void)

Get the root node from any child node of the tree

3.78.4 Friends And Related Function Documentation

3.78.4.1 bool MatchXMLName (const [XMLNode](#) & *node1*, const [XMLNode](#) & *node2*) [friend]

Returns true if underlying XML elements have same names

3.78.4.2 bool MatchXMLName (const [XMLNode](#) & *node*, const char * *name*) [friend]

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

3.78.5 Member Data Documentation

3.78.5.1 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.

3.78.5.2 **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

- ~Counter
 - Arc::Counter, [17](#)
- ~DataBufferPar
 - Arc::DataBufferPar, [26](#)
- ~DataPointDirect
 - Arc::DataPointDirect, [42](#)
- ~DataSpeed
 - Arc::DataSpeed, [53](#)
- ~IntraProcessCounter
 - Arc::IntraProcessCounter, [71](#)
- ~Loader
 - Arc::Loader, [74](#)
- ~Message
 - Arc::Message, [97](#)
- ~PayloadRaw
 - Arc::PayloadRaw, [107](#)
- ~PayloadStream
 - Arc::PayloadStream, [113](#)
- ~Plexer
 - Arc::Plexer, [122](#)
- ~RegularExpression
 - Arc::RegularExpression, [125](#)
- ~SOAPMessage
 - Arc::SOAPMessage, [140](#)
- ~URL
 - Arc::URL, [146](#)
- ~URLLocation
 - Arc::URLLocation, [151](#)
- ~WSAEndpointReference
 - Arc::WSAEndpointReference, [153](#)
- ~XMLNode
 - Arc::XMLNode, [168](#)
- accepts_meta
 - Arc::DataPoint, [37](#)
 - Arc::DataPointIndex, [49](#)
- Acquire
 - Arc::InformationContainer, [63](#)
- Action
 - Arc::WSAHeader, [156](#)
- Add
 - Arc::MessageContext, [103](#)
- add
 - Arc::MessageAttributes, [100](#)
- add_location
 - Arc::DataPoint, [39](#)
 - Arc::DataPointIndex, [49](#)
- addDestination
 - Arc::Logger, [81](#)
- additional_checks
 - Arc::DataPointDirect, [44](#)
- AddOption
 - Arc::URL, [147](#)
- Address
 - Arc::WSAEndpointReference, [154](#)
- AddSecHandler
 - Arc::MCC, [88](#)
 - Arc::Service, [130](#)
- allocated_
 - Arc::WSRF, [159](#)
- analyze
 - Arc::DataPointDirect, [43](#)
- Arc::AttributeIterator, [5](#)
- Arc::AttributeIterator
 - AttributeIterator, [6](#)
 - current_, [7](#)
 - end_, [7](#)
 - hasMore, [7](#)
 - MessageAttributes, [7](#)
 - operator *, [6](#)
 - operator++, [6, 7](#)
 - operator->, [6](#)
- Arc::ChainContext, [9](#)
- Arc::ChainContext
 - operator MCCFactory *, [9](#)
 - operator PDPFactory *, [9](#)
 - operator SecHandlerFactory *, [9](#)
 - operator ServiceFactory *, [9](#)
- Arc::Checksum, [10](#)
- Arc::ChecksumAny, [11](#)
- Arc::Config, [13](#)
 - Config, [13](#)
 - parse, [14](#)
 - print, [14](#)
- Arc::Counter, [15](#)
 - ~Counter, [17](#)
 - cancel, [19](#)
 - changeExcess, [18](#)
 - changeLimit, [18](#)
 - Counter, [17](#)

- CounterTicket, 21
- ExpirationReminder, 21
- extend, 19
- getCounterTicket, 20
- getCurrentTime, 20
- getExcess, 18
- getExpirationReminder, 21
- getExpiryTime, 20
- getLimit, 17
- getValue, 19
- IDType, 17
- reserve, 19
- setExcess, 18
- setLimit, 17
- Arc::CounterTicket, 22
- Arc::CounterTicket
 - cancel, 23
 - Counter, 23
 - CounterTicket, 22
 - extend, 23
 - isValid, 22
- Arc::CRC32Sum, 24
- Arc::DataBufferPar, 25
- Arc::DataBufferPar
 - ~DataBufferPar, 26
 - buffer_size, 30
 - checksum_object, 30
 - checksum_valid, 30
 - DataBufferPar, 26
 - eof_position, 30
 - eof_read, 28, 29
 - eof_write, 29
 - error, 29
 - error_read, 29
 - error_transfer, 29
 - error_write, 29
 - for_read, 27
 - for_write, 27, 28
 - is_notwritten, 28
 - is_read, 27
 - is_written, 28
 - operator bool, 26
 - operator[], 27
 - set, 26
 - speed, 31
 - wait, 29
 - wait_eof, 30
 - wait_eof_read, 30
 - wait_eof_write, 30
 - wait_read, 30
 - wait_used, 30
 - wait_write, 30
- Arc::DataHandle, 32
- Arc::DataPoint, 33
- Arc::DataPoint
 - accepts_meta, 37
 - add_location, 39
 - base_url, 39
 - current_location, 38
 - current_meta_location, 38
 - DataPoint, 34
 - get_info, 35
 - have_location, 39
 - have_locations, 39
 - list_files, 35
 - local, 38
 - meta, 37, 38
 - meta_checksum, 36
 - meta_checksum_available, 36
 - meta_checksum_force, 36
 - meta_compare, 38
 - meta_created, 37
 - meta_created_available, 36
 - meta_created_force, 37
 - meta_postregister, 35
 - meta_preregister, 34
 - meta_preunregister, 35
 - meta_resolve, 34
 - meta_size, 36
 - meta_size_available, 36
 - meta_size_force, 36
 - meta_stored, 38
 - meta_unregister, 35
 - meta_validtill, 37
 - meta_validtill_available, 37
 - meta_validtill_force, 37
 - next_location, 38
 - provides_meta, 37
 - remove_location, 39
 - remove_locations, 39
 - tries, 39
- Arc::DataPointDirect, 41
- Arc::DataPointDirect
 - ~DataPointDirect, 42
 - additional_checks, 44
 - analyze, 43
 - check, 43
 - DataPointDirect, 42
 - failure_reason, 45
 - failure_reason_t, 42
 - list_files, 44
 - meta, 43
 - out_of_order, 44
 - passive, 45
 - range, 45
 - remove, 43
 - secure, 44
 - start_reading, 43

- start_writing, 43
- stop_reading, 43
- stop_writing, 43
- Arc::DataPointDirect::analyze_t, 46
- Arc::DataPointIndex, 47
- Arc::DataPointIndex
 - accepts_meta, 49
 - add_location, 49
 - current_location, 48
 - current_meta_location, 48
 - get_info, 48
 - have_location, 48
 - have_locations, 48
 - locations, 50
 - meta, 49
 - meta_stored, 49
 - next_location, 48
 - provides_meta, 49
 - remove_location, 48
 - remove_locations, 49
 - tries, 49
- Arc::DataPointIndex::Location, 51
- Arc::DataSpeed, 52
- Arc::DataSpeed
 - ~DataSpeed, 53
 - DataSpeed, 52
 - hold, 55
 - max_inactivity_time_failure, 55
 - min_average_speed_failure, 55
 - min_speed_failure, 55
 - reset, 54
 - set_base, 54
 - set_max_data, 54
 - set_max_inactivity_time, 54
 - set_min_average_speed, 53
 - set_min_speed, 53
 - set_progress_indicator, 54
 - transfer, 54
 - transferred_size, 55
 - verbose, 53
- Arc::DelegationConsumer, 56
- Arc::DelegationProvider, 57
- Arc::DMCFactory, 59
 - DMCFactory, 59
 - get_instance, 59
- Arc::ExpirationReminder, 60
- Arc::ExpirationReminder
 - Counter, 61
 - getExpiryTime, 60
 - getReservationID, 60
 - operator<, 60
- Arc::FileInfo, 62
- Arc::InformationContainer, 63
- Arc::InformationContainer
 - Acquire, 63
 - doc_, 64
 - Get, 63
- Arc::InformationInterface, 65
- Arc::InformationInterface
 - Get, 65
 - InformationInterface, 65
 - lock_, 66
- Arc::InformationRequest, 67
- Arc::InformationRequest
 - InformationRequest, 67
 - SOAP, 67
- Arc::InformationResponse, 69
- Arc::InformationResponse
 - InformationResponse, 69
 - Result, 69
- Arc::IntraProcessCounter, 70
- Arc::IntraProcessCounter
 - ~IntraProcessCounter, 71
 - cancel, 73
 - changeExcess, 72
 - changeLimit, 71
 - extend, 73
 - getExcess, 71
 - getLimit, 71
 - getValue, 72
 - IntraProcessCounter, 70
 - reserve, 72
 - setExcess, 72
 - setLimit, 71
- Arc::Loader, 74
 - ~Loader, 74
 - Loader, 74
 - operator[], 75
- Arc::loader_descriptor, 76
- Arc::LoaderFactory, 77
- Arc::LoaderFactory
 - get_instance, 77
 - load_all_instances, 77
 - LoaderFactory, 77
- Arc::LogDestination, 79
- Arc::LogDestination
 - log, 79
 - LogDestination, 79
- Arc::Logger, 80
 - addDestination, 81
 - getThreshold, 81
 - Logger, 80
 - msg, 81
 - rootLogger, 82
 - setThreshold, 81
- Arc::LogMessage, 83
- Arc::LogMessage
 - getLevel, 84

- Logger, 84
- LogMessage, 83
- operator<<, 84
- setIdentifier, 84
- Arc::LogStream, 85
- Arc::LogStream
 - log, 85
 - LogStream, 85
- Arc::MCC, 87
 - AddSecHandler, 88
 - logger, 88
 - MCC, 87
 - Next, 88
 - next_, 88
 - process, 88
 - sechandlers_, 88
 - Unlink, 88
- Arc::MCC_Status, 90
 - getExplanation, 91
 - getKind, 90
 - getOrigin, 91
 - isOk, 90
 - MCC_Status, 90
 - operator bool, 91
 - operator std::string, 91
 - operator!, 91
- Arc::MCCFactory, 93
 - get_instance, 93
 - MCCFactory, 93
- Arc::MCCInterface, 94
 - process, 94
- Arc::MD5Sum, 95
- Arc::Message, 96
 - ~Message, 97
 - Attributes, 97
 - Auth, 97
 - Context, 97
 - Message, 97
 - operator=, 97
 - Payload, 97
- Arc::MessageAttributes, 99
- Arc::MessageAttributes
 - add, 100
 - attributes_, 101
 - count, 100
 - get, 101
 - getAll, 101
 - MessageAttributes, 99
 - remove, 100
 - removeAll, 100
 - set, 100
- Arc::MessageAuth, 102
- Arc::MessageContext, 103
- Arc::MessageContext
 - Add, 103
- Arc::MessageContextElement, 104
- Arc::MessagePayload, 105
- Arc::ModuleManager, 106
- Arc::ModuleManager
 - load, 106
 - ModuleManager, 106
- Arc::PayloadRaw, 107
- Arc::PayloadRaw
 - ~PayloadRaw, 107
 - Buffer, 108
 - BufferPos, 108
 - BufferSize, 108
 - Content, 108
 - Insert, 108
 - operator[], 108
 - PayloadRaw, 107
 - Size, 108
 - Truncate, 109
- Arc::PayloadRawInterface, 110
- Arc::PayloadRawInterface
 - Buffer, 111
 - BufferPos, 111
 - BufferSize, 111
 - Content, 110
 - Insert, 111
 - operator[], 110
 - Size, 111
 - Truncate, 111
- Arc::PayloadSOAP, 112
- Arc::PayloadSOAP
 - PayloadSOAP, 112
- Arc::PayloadStream, 113
- Arc::PayloadStream
 - ~PayloadStream, 113
 - Get, 114
 - GetHandle, 115
 - handle_, 115
 - operator bool, 114
 - operator!, 114
 - PayloadStream, 113
 - Put, 114
 - seekable_, 115
 - Timeout, 115
- Arc::PayloadStreamInterface, 116
- Arc::PayloadStreamInterface
 - Get, 116
 - operator bool, 117
 - operator!, 117
 - Put, 117
 - Timeout, 117
- Arc::PayloadWSRF, 118
- Arc::PayloadWSRF
 - PayloadWSRF, 118

- Arc::PDPFactory, 121
 - get_instance, 121
 - PDPFactory, 121
- Arc::Plexer, 122
 - ~Plexer, 122
 - Next, 122
 - Plexer, 122
 - process, 123
- Arc::PlexerEntry, 124
- Arc::RegularExpression, 125
- Arc::RegularExpression
 - ~RegularExpression, 125
 - getPattern, 126
 - hasPattern, 125
 - isOk, 125
 - match, 126
 - operator=, 125
 - RegularExpression, 125
- Arc::SecHandlerFactory, 128
- Arc::SecHandlerFactory
 - get_instance, 128
 - SecHandlerFactory, 128
- Arc::Service, 129
 - AddSecHandler, 130
 - sechandlers_, 130
 - Service, 129
- Arc::ServiceFactory, 132
- Arc::ServiceFactory
 - get_instance, 132
 - ServiceFactory, 132
- Arc::SimpleCondition, 133
- Arc::SimpleCondition
 - broadcast, 133
 - lock, 133
 - reset, 134
 - signal, 133
 - signal_nonblock, 133
 - unlock, 133
 - wait, 133, 134
 - wait_nonblock, 134
- Arc::SOAPEnvelope, 135
 - Fault, 136
 - GetXML, 136
 - Header, 136
 - IsFault, 136
 - Namespaces, 136
 - New, 136
 - SOAPEnvelope, 135, 136
- Arc::SOAPFault, 137
 - Code, 138
 - Detail, 139
 - Node, 138
 - operator bool, 138
 - Reason, 138
 - Role, 138, 139
 - SOAPFault, 138
 - SOAPFaultCode, 137
 - Subcode, 138
- Arc::SOAPMessage, 140
 - ~SOAPMessage, 140
 - Attributes, 141
 - operator=, 141
 - Payload, 141
 - SOAPMessage, 140
- Arc::Time, 142
 - GetFormat, 143
 - GetTime, 143
 - operator std::string, 143
 - operator!=, 144
 - operator+, 144
 - operator-, 144
 - operator<, 143
 - operator<=, 143
 - operator=, 143
 - operator==, 144
 - operator>, 143
 - operator>=, 143
 - SetFormat, 143
 - SetTime, 143
 - str, 143
 - Time, 142
- Arc::URL, 145
 - ~URL, 146
 - AddOption, 147
 - BaseDN, 147
 - BaseDN2Path, 149
 - CommonLocOption, 148
 - CommonLocOptions, 148
 - commonlocoptions, 150
 - ConnectionURL, 148
 - fullstr, 148
 - Host, 147
 - host, 149
 - HTTPOption, 147
 - HTTPOptions, 147
 - httpoptions, 149
 - Locations, 148
 - locations, 150
 - operator bool, 148
 - operator<, 148
 - operator<<, 149
 - operator==, 148
 - Option, 147
 - Options, 147
 - Passwd, 146
 - passwd, 149
 - Path, 147
 - path, 149

- Path2BaseDN, 149
- Port, 147
- port, 149
- Protocol, 146
- protocol, 149
- str, 148
- URL, 146
- urloptions, 150
- Username, 146
- username, 149
- Arc::URLLocation, 151
 - ~URLLocation, 151
 - fullstr, 152
 - Name, 152
 - name, 152
 - str, 152
 - URLLocation, 151
- Arc::WSAEndpointReference, 153
- Arc::WSAEndpointReference
 - ~WSAEndpointReference, 153
 - Address, 154
 - MetaData, 154
 - operator XMLNode, 154
 - operator=, 154
 - ReferenceParameters, 154
 - WSAEndpointReference, 153
- Arc::WSAHeader, 155
 - Action, 156
 - Check, 157
 - FaultTo, 156
 - From, 156
 - header_allocated_, 157
 - MessageID, 156
 - NewReferenceParameter, 157
 - operator XMLNode, 157
 - ReferenceParameter, 157
 - RelatesTo, 156, 157
 - RelationshipType, 157
 - ReplyTo, 156
 - To, 156
 - WSAHeader, 155
- Arc::WSRF, 158
 - allocated_, 159
 - operator bool, 159
 - set_namespaces, 159
 - SOAP, 159
 - valid_, 159
 - WSRF, 158
- Arc::WSRFBBaseFault, 160
- Arc::WSRFBBaseFault
 - set_namespaces, 161
 - WSRFBBaseFault, 160
- Arc::WSRP, 162
 - set_namespaces, 162
 - WSRP, 162
- Arc::WSRPFault, 164
 - WSRPFault, 164
- Arc::WSRPResourcePropertyChangeFailure, 165
- Arc::WSRPResourcePropertyChangeFailure
 - WSRPResourcePropertyChangeFailure, 165
- Arc::XMLNode, 166
 - ~XMLNode, 168
 - Attribute, 170
 - AttributesSize, 170
 - Child, 168
 - Destroy, 171
 - Get, 169
 - GetRoot, 171
 - GetXML, 169
 - is_owner_, 171
 - is_temporary_, 171
 - MatchXMLName, 171
 - Name, 169
 - NamespacePrefix, 170
 - Namespaces, 170
 - New, 168
 - NewAttribute, 170
 - NewChild, 170, 171
 - operator bool, 168
 - operator std::string, 169
 - operator!, 168
 - operator=, 169, 170
 - operator[], 168, 169
 - Replace, 171
 - Set, 169
 - Size, 169
 - XMLNode, 167, 168
 - XPathLookup, 171
- Attribute
 - Arc::XMLNode, 170
- AttributeIterator
 - Arc::AttributeIterator, 6
- Attributes
 - Arc::Message, 97
 - Arc::SOAPMessage, 141
- attributes_
 - Arc::MessageAttributes, 101
- AttributesSize
 - Arc::XMLNode, 170
- Auth
 - Arc::Message, 97
- base_url
 - Arc::DataPoint, 39
- BaseDN
 - Arc::URL, 147
- BaseDN2Path
 - Arc::URL, 149

- broadcast
 - Arc::SimpleCondition, 133
- Buffer
 - Arc::PayloadRaw, 108
 - Arc::PayloadRawInterface, 111
- buffer_size
 - Arc::DataBufferPar, 30
- BufferPos
 - Arc::PayloadRaw, 108
 - Arc::PayloadRawInterface, 111
- BufferSize
 - Arc::PayloadRaw, 108
 - Arc::PayloadRawInterface, 111
- cancel
 - Arc::Counter, 19
 - Arc::CounterTicket, 23
 - Arc::IntraProcessCounter, 73
- changeExcess
 - Arc::Counter, 18
 - Arc::IntraProcessCounter, 72
- changeLimit
 - Arc::Counter, 18
 - Arc::IntraProcessCounter, 71
- Check
 - Arc::WSAHeader, 157
- check
 - Arc::DataPointDirect, 43
- checksum_object
 - Arc::DataBufferPar, 30
- checksum_valid
 - Arc::DataBufferPar, 30
- Child
 - Arc::XMLNode, 168
- Code
 - Arc::SOAPFault, 138
- CommonLocOption
 - Arc::URL, 148
- CommonLocOptions
 - Arc::URL, 148
- commonlocoptions
 - Arc::URL, 150
- Config
 - Arc::Config, 13
- ConnectionURL
 - Arc::URL, 148
- Content
 - Arc::PayloadRaw, 108
 - Arc::PayloadRawInterface, 110
- Context
 - Arc::Message, 97
- count
 - Arc::MessageAttributes, 100
- Counter
 - Arc::Counter, 17
 - Arc::CounterTicket, 23
 - Arc::ExpirationReminder, 61
- CounterTicket
 - Arc::Counter, 21
 - Arc::CounterTicket, 22
- current_
 - Arc::AttributeIterator, 7
- current_location
 - Arc::DataPoint, 38
 - Arc::DataPointIndex, 48
- current_meta_location
 - Arc::DataPoint, 38
 - Arc::DataPointIndex, 48
- DataBufferPar
 - Arc::DataBufferPar, 26
- DataPoint
 - Arc::DataPoint, 34
- DataPointDirect
 - Arc::DataPointDirect, 42
- DataSpeed
 - Arc::DataSpeed, 52
- Destroy
 - Arc::XMLNode, 171
- Detail
 - Arc::SOAPFault, 139
- dmc_descriptor, 58
- DMCFactory
 - Arc::DMCFactory, 59
- doc_
 - Arc::InformationContainer, 64
- end_
 - Arc::AttributeIterator, 7
- eof_position
 - Arc::DataBufferPar, 30
- eof_read
 - Arc::DataBufferPar, 28, 29
- eof_write
 - Arc::DataBufferPar, 29
- error
 - Arc::DataBufferPar, 29
- error_read
 - Arc::DataBufferPar, 29
- error_transfer
 - Arc::DataBufferPar, 29
- error_write
 - Arc::DataBufferPar, 29
- ExpirationReminder
 - Arc::Counter, 21
- extend
 - Arc::Counter, 19
 - Arc::CounterTicket, 23

- Arc::IntraProcessCounter, 73
- failure_reason
 - Arc::DataPointDirect, 45
- failure_reason_t
 - Arc::DataPointDirect, 42
- Fault
 - Arc::SOAPEnvelope, 136
- FaultTo
 - Arc::WSAHeader, 156
- for_read
 - Arc::DataBufferPar, 27
- for_write
 - Arc::DataBufferPar, 27, 28
- From
 - Arc::WSAHeader, 156
- fullstr
 - Arc::URL, 148
 - Arc::URLLocation, 152
- Get
 - Arc::InformationContainer, 63
 - Arc::InformationInterface, 65
 - Arc::PayloadStream, 114
 - Arc::PayloadStreamInterface, 116
 - Arc::XMLNode, 169
- get
 - Arc::MessageAttributes, 101
- get_info
 - Arc::DataPoint, 35
 - Arc::DataPointIndex, 48
- get_instance
 - Arc::DMCFactory, 59
 - Arc::LoaderFactory, 77
 - Arc::MCCFactory, 93
 - Arc::PDPFactory, 121
 - Arc::SecHandlerFactory, 128
 - Arc::ServiceFactory, 132
- getAll
 - Arc::MessageAttributes, 101
- getCounterTicket
 - Arc::Counter, 20
- getCurrentTime
 - Arc::Counter, 20
- getExcess
 - Arc::Counter, 18
 - Arc::IntraProcessCounter, 71
- getExpirationReminder
 - Arc::Counter, 21
- getExpiryTime
 - Arc::Counter, 20
 - Arc::ExpirationReminder, 60
- getExplanation
 - Arc::MCC_Status, 91
- GetFormat
 - Arc::Time, 143
- GetHandle
 - Arc::PayloadStream, 115
- getKind
 - Arc::MCC_Status, 90
- getLevel
 - Arc::LogMessage, 84
- getLimit
 - Arc::Counter, 17
 - Arc::IntraProcessCounter, 71
- getOrigin
 - Arc::MCC_Status, 91
- getPattern
 - Arc::RegularExpression, 126
- getReservationID
 - Arc::ExpirationReminder, 60
- GetRoot
 - Arc::XMLNode, 171
- getThreshold
 - Arc::Logger, 81
- GetTime
 - Arc::Time, 143
- getValue
 - Arc::Counter, 19
 - Arc::IntraProcessCounter, 72
- GetXML
 - Arc::SOAPEnvelope, 136
 - Arc::XMLNode, 169
- handle_
 - Arc::PayloadStream, 115
- hasMore
 - Arc::AttributeIterator, 7
- hasPattern
 - Arc::RegularExpression, 125
- have_location
 - Arc::DataPoint, 39
 - Arc::DataPointIndex, 48
- have_locations
 - Arc::DataPoint, 39
 - Arc::DataPointIndex, 48
- Header
 - Arc::SOAPEnvelope, 136
- header_allocated_
 - Arc::WSAHeader, 157
- hold
 - Arc::DataSpeed, 55
- Host
 - Arc::URL, 147
- host
 - Arc::URL, 149
- HTTPOption
 - Arc::URL, 147

- HTTPOptions
 - Arc::URL, [147](#)
- httpoptions
 - Arc::URL, [149](#)
- IDType
 - Arc::Counter, [17](#)
- InformationInterface
 - Arc::InformationInterface, [65](#)
- InformationRequest
 - Arc::InformationRequest, [67](#)
- InformationResponse
 - Arc::InformationResponse, [69](#)
- Insert
 - Arc::PayloadRaw, [108](#)
 - Arc::PayloadRawInterface, [111](#)
- IntraProcessCounter
 - Arc::IntraProcessCounter, [70](#)
- is_notwritten
 - Arc::DataBufferPar, [28](#)
- is_owner_
 - Arc::XMLNode, [171](#)
- is_read
 - Arc::DataBufferPar, [27](#)
- is_temporary_
 - Arc::XMLNode, [171](#)
- is_written
 - Arc::DataBufferPar, [28](#)
- IsFault
 - Arc::SOAPEnvelope, [136](#)
- isOk
 - Arc::MCC_Status, [90](#)
 - Arc::RegularExpression, [125](#)
- isValid
 - Arc::CounterTicket, [22](#)
- list_files
 - Arc::DataPoint, [35](#)
 - Arc::DataPointDirect, [44](#)
- load
 - Arc::ModuleManager, [106](#)
- load_all_instances
 - Arc::LoaderFactory, [77](#)
- Loader
 - Arc::Loader, [74](#)
- LoaderFactory
 - Arc::LoaderFactory, [77](#)
- local
 - Arc::DataPoint, [38](#)
- Locations
 - Arc::URL, [148](#)
- locations
 - Arc::DataPointIndex, [50](#)
 - Arc::URL, [150](#)
- lock
 - Arc::SimpleCondition, [133](#)
- lock_
 - Arc::InformationInterface, [66](#)
- log
 - Arc::LogDestination, [79](#)
 - Arc::LogStream, [85](#)
- LogDestination
 - Arc::LogDestination, [79](#)
- Logger
 - Arc::Logger, [80](#)
 - Arc::LogMessage, [84](#)
- logger
 - Arc::MCC, [88](#)
- LogMessage
 - Arc::LogMessage, [83](#)
- LogStream
 - Arc::LogStream, [85](#)
- match
 - Arc::RegularExpression, [126](#)
- MatchXMLName
 - Arc::XMLNode, [171](#)
- max_inactivity_time_failure
 - Arc::DataSpeed, [55](#)
- MCC
 - Arc::MCC, [87](#)
- mcc_descriptor, [89](#)
- MCC_Status
 - Arc::MCC_Status, [90](#)
- MCCFactory
 - Arc::MCCFactory, [93](#)
- Message
 - Arc::Message, [97](#)
- MessageAttributes
 - Arc::AttributeIterator, [7](#)
 - Arc::MessageAttributes, [99](#)
- MessageID
 - Arc::WSAHeader, [156](#)
- meta
 - Arc::DataPoint, [37](#), [38](#)
 - Arc::DataPointDirect, [43](#)
 - Arc::DataPointIndex, [49](#)
- meta_checksum
 - Arc::DataPoint, [36](#)
- meta_checksum_available
 - Arc::DataPoint, [36](#)
- meta_checksum_force
 - Arc::DataPoint, [36](#)
- meta_compare
 - Arc::DataPoint, [38](#)
- meta_created
 - Arc::DataPoint, [37](#)
- meta_created_available

- Arc::DataPoint, 36
- meta_created_force
 - Arc::DataPoint, 37
- meta_postregister
 - Arc::DataPoint, 35
- meta_preregister
 - Arc::DataPoint, 34
- meta_preunregister
 - Arc::DataPoint, 35
- meta_resolve
 - Arc::DataPoint, 34
- meta_size
 - Arc::DataPoint, 36
- meta_size_available
 - Arc::DataPoint, 36
- meta_size_force
 - Arc::DataPoint, 36
- meta_stored
 - Arc::DataPoint, 38
 - Arc::DataPointIndex, 49
- meta_unregister
 - Arc::DataPoint, 35
- meta_validtill
 - Arc::DataPoint, 37
- meta_validtill_available
 - Arc::DataPoint, 37
- meta_validtill_force
 - Arc::DataPoint, 37
- MetaData
 - Arc::WSAEndpointReference, 154
- min_average_speed_failure
 - Arc::DataSpeed, 55
- min_speed_failure
 - Arc::DataSpeed, 55
- ModuleManager
 - Arc::ModuleManager, 106
- msg
 - Arc::Logger, 81
- Name
 - Arc::URLLocation, 152
 - Arc::XMLNode, 169
- name
 - Arc::URLLocation, 152
- NamespacePrefix
 - Arc::XMLNode, 170
- Namespaces
 - Arc::SOAPEnvelope, 136
 - Arc::XMLNode, 170
- New
 - Arc::SOAPEnvelope, 136
 - Arc::XMLNode, 168
- NewAttribute
 - Arc::XMLNode, 170
- NewChild
 - Arc::XMLNode, 170, 171
- NewReferenceParameter
 - Arc::WSAHeader, 157
- Next
 - Arc::MCC, 88
 - Arc::Plexer, 122
- next_
 - Arc::MCC, 88
- next_location
 - Arc::DataPoint, 38
 - Arc::DataPointIndex, 48
- Node
 - Arc::SOAPFault, 138
- operator *
 - Arc::AttributeIterator, 6
- operator bool
 - Arc::DataBufferPar, 26
 - Arc::MCC_Status, 91
 - Arc::PayloadStream, 114
 - Arc::PayloadStreamInterface, 117
 - Arc::SOAPFault, 138
 - Arc::URL, 148
 - Arc::WSRF, 159
 - Arc::XMLNode, 168
- operator MCCFactory *
 - Arc::ChainContext, 9
- operator PDPFactory *
 - Arc::ChainContext, 9
- operator SecHandlerFactory *
 - Arc::ChainContext, 9
- operator ServiceFactory *
 - Arc::ChainContext, 9
- operator std::string
 - Arc::MCC_Status, 91
 - Arc::Time, 143
 - Arc::XMLNode, 169
- operator XMLNode
 - Arc::WSAEndpointReference, 154
 - Arc::WSAHeader, 157
- operator!
 - Arc::MCC_Status, 91
 - Arc::PayloadStream, 114
 - Arc::PayloadStreamInterface, 117
 - Arc::XMLNode, 168
- operator!=
 - Arc::Time, 144
- operator+
 - Arc::Time, 144
- operator++
 - Arc::AttributeIterator, 6, 7
- operator-
 - Arc::Time, 144

- operator->
 - Arc::AttributeIterator, 6
- operator<
 - Arc::ExpirationReminder, 60
 - Arc::Time, 143
 - Arc::URL, 148
- operator<<
 - Arc::LogMessage, 84
 - Arc::URL, 149
- operator<=
 - Arc::Time, 143
- operator=
 - Arc::Message, 97
 - Arc::RegularExpression, 125
 - Arc::SOAPMessage, 141
 - Arc::Time, 143
 - Arc::WSAEndpointReference, 154
 - Arc::XMLNode, 169, 170
- operator==
 - Arc::Time, 144
 - Arc::URL, 148
- operator>
 - Arc::Time, 143
- operator>=
 - Arc::Time, 143
- operator[]
 - Arc::DataBufferPar, 27
 - Arc::Loader, 75
 - Arc::PayloadRaw, 108
 - Arc::PayloadRawInterface, 110
 - Arc::XMLNode, 168, 169
- Option
 - Arc::URL, 147
- Options
 - Arc::URL, 147
- out_of_order
 - Arc::DataPointDirect, 44
- parse
 - Arc::Config, 14
- passive
 - Arc::DataPointDirect, 45
- Passwd
 - Arc::URL, 146
- passwd
 - Arc::URL, 149
- Path
 - Arc::URL, 147
- path
 - Arc::URL, 149
- Path2BaseDN
 - Arc::URL, 149
- Payload
 - Arc::Message, 97
 - Arc::SOAPMessage, 141
- PayloadRaw
 - Arc::PayloadRaw, 107
- PayloadSOAP
 - Arc::PayloadSOAP, 112
- PayloadStream
 - Arc::PayloadStream, 113
- PayloadWSRF
 - Arc::PayloadWSRF, 118
- pdp_descriptor, 120
- PDPFactory
 - Arc::PDPFactory, 121
- Plexer
 - Arc::Plexer, 122
- Port
 - Arc::URL, 147
- port
 - Arc::URL, 149
- print
 - Arc::Config, 14
- process
 - Arc::MCC, 88
 - Arc::MCCInterface, 94
 - Arc::Plexer, 123
- Protocol
 - Arc::URL, 146
- protocol
 - Arc::URL, 149
- provides_meta
 - Arc::DataPoint, 37
 - Arc::DataPointIndex, 49
- Put
 - Arc::PayloadStream, 114
 - Arc::PayloadStreamInterface, 117
- range
 - Arc::DataPointDirect, 45
- Reason
 - Arc::SOAPFault, 138
- ReferenceParameter
 - Arc::WSAHeader, 157
- ReferenceParameters
 - Arc::WSAEndpointReference, 154
- RegularExpression
 - Arc::RegularExpression, 125
- RelatesTo
 - Arc::WSAHeader, 156, 157
- RelationshipType
 - Arc::WSAHeader, 157
- remove
 - Arc::DataPointDirect, 43
 - Arc::MessageAttributes, 100
- remove_location
 - Arc::DataPoint, 39

- Arc::DataPointIndex, 48
- remove_locations
 - Arc::DataPoint, 39
 - Arc::DataPointIndex, 49
- removeAll
 - Arc::MessageAttributes, 100
- Replace
 - Arc::XMLNode, 171
- ReplyTo
 - Arc::WSAHeader, 156
- reserve
 - Arc::Counter, 19
 - Arc::IntraProcessCounter, 72
- reset
 - Arc::DataSpeed, 54
 - Arc::SimpleCondition, 134
- Result
 - Arc::InformationResponse, 69
- Role
 - Arc::SOAPFault, 138, 139
- rootLogger
 - Arc::Logger, 82
- sechandler_descriptor, 127
- SecHandlerFactory
 - Arc::SecHandlerFactory, 128
- sechandler_
 - Arc::MCC, 88
 - Arc::Service, 130
- secure
 - Arc::DataPointDirect, 44
- seekable_
 - Arc::PayloadStream, 115
- Service
 - Arc::Service, 129
- service_descriptor, 131
- ServiceFactory
 - Arc::ServiceFactory, 132
- Set
 - Arc::XMLNode, 169
- set
 - Arc::DataBufferPar, 26
 - Arc::MessageAttributes, 100
- set_base
 - Arc::DataSpeed, 54
- set_max_data
 - Arc::DataSpeed, 54
- set_max_inactivity_time
 - Arc::DataSpeed, 54
- set_min_average_speed
 - Arc::DataSpeed, 53
- set_min_speed
 - Arc::DataSpeed, 53
- set_namespaces
 - Arc::WSRF, 159
 - Arc::WSRFBBaseFault, 161
 - Arc::WSRP, 162
- set_progress_indicator
 - Arc::DataSpeed, 54
- setExcess
 - Arc::Counter, 18
 - Arc::IntraProcessCounter, 72
- SetFormat
 - Arc::Time, 143
- setIdentifier
 - Arc::LogMessage, 84
- setLimit
 - Arc::Counter, 17
 - Arc::IntraProcessCounter, 71
- setThreshold
 - Arc::Logger, 81
- SetTime
 - Arc::Time, 143
- signal
 - Arc::SimpleCondition, 133
- signal_nonblock
 - Arc::SimpleCondition, 133
- Size
 - Arc::PayloadRaw, 108
 - Arc::PayloadRawInterface, 111
 - Arc::XMLNode, 169
- SOAP
 - Arc::InformationRequest, 67
 - Arc::WSRF, 159
- SOAPEnvelope
 - Arc::SOAPEnvelope, 135, 136
- SOAPFault
 - Arc::SOAPFault, 138
- SOAPFaultCode
 - Arc::SOAPFault, 137
- SOAPMessage
 - Arc::SOAPMessage, 140
- speed
 - Arc::DataBufferPar, 31
- start_reading
 - Arc::DataPointDirect, 43
- start_writing
 - Arc::DataPointDirect, 43
- stop_reading
 - Arc::DataPointDirect, 43
- stop_writing
 - Arc::DataPointDirect, 43
- str
 - Arc::Time, 143
 - Arc::URL, 148
 - Arc::URLLocation, 152
- Subcode
 - Arc::SOAPFault, 138

- Time
 - Arc::Time, [142](#)
- Timeout
 - Arc::PayloadStream, [115](#)
 - Arc::PayloadStreamInterface, [117](#)
- To
 - Arc::WSAHeader, [156](#)
- transfer
 - Arc::DataSpeed, [54](#)
- transferred_size
 - Arc::DataSpeed, [55](#)
- tries
 - Arc::DataPoint, [39](#)
 - Arc::DataPointIndex, [49](#)
- Truncate
 - Arc::PayloadRaw, [109](#)
 - Arc::PayloadRawInterface, [111](#)
- Unlink
 - Arc::MCC, [88](#)
- unlock
 - Arc::SimpleCondition, [133](#)
- URL
 - Arc::URL, [146](#)
- URLLocation
 - Arc::URLLocation, [151](#)
- urloptions
 - Arc::URL, [150](#)
- Username
 - Arc::URL, [146](#)
- username
 - Arc::URL, [149](#)
- valid_
 - Arc::WSRF, [159](#)
- verbose
 - Arc::DataSpeed, [53](#)
- wait
 - Arc::DataBufferPar, [29](#)
 - Arc::SimpleCondition, [133](#), [134](#)
- wait_eof
 - Arc::DataBufferPar, [30](#)
- wait_eof_read
 - Arc::DataBufferPar, [30](#)
- wait_eof_write
 - Arc::DataBufferPar, [30](#)
- wait_nonblock
 - Arc::SimpleCondition, [134](#)
- wait_read
 - Arc::DataBufferPar, [30](#)
- wait_used
 - Arc::DataBufferPar, [30](#)
- wait_write
 - Arc::DataBufferPar, [30](#)
- WSAEndpointReference
 - Arc::WSAEndpointReference, [153](#)
- WSAHeader
 - Arc::WSAHeader, [155](#)
- WSRF
 - Arc::WSRF, [158](#)
- WSRFBBaseFault
 - Arc::WSRFBBaseFault, [160](#)
- WSRP
 - Arc::WSRP, [162](#)
- WSRPFault
 - Arc::WSRPFault, [164](#)
- WSRPResourcePropertyChangeFailure
 - Arc::WSRPResourcePropertyChangeFailure, [165](#)
- XMLNode
 - Arc::XMLNode, [167](#), [168](#)
- XPathLookup
 - Arc::XMLNode, [171](#)