KnowARC Reference Manual

Generated by Doxygen 1.3.5

Mon May 21 23:57:21 2007

Contents

Chapter 1

Service Definitions

1.1 Bindings

- Binding "echo"

Chapter 2

KnowARC Hierarchical Index

2.1 KnowARC Class Hierarchy

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

Chapter 3

KnowARC Class Index

3.1 KnowARC Class List

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

Chapter 4

KnowARC Page Index

4.1 KnowARC Related Pages

Here is a list of all related documentation pages:

Chapter 5

KnowARC Class Documentation

5.1 Arc::AttributeIterator Class Reference

An iterator class for accessing multiple values of an attribute.

`#include <MessageAttributes.h>`

Public Member Functions

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

Protected Member Functions

- AttributeIterator (AttrConstIter begin, AttrConstIter end)

Protected Attributes

- AttrConstIter current_
- AttrConstIter end_

Friends

- class MessageAttributes

5.1.1 Detailed Description

An iterator class for accessing multiple values of an attribute.

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

Typical usage is:

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

5.1.2 Constructor & Destructor Documentation

5.1.2.1 Arc::AttributeIterator::AttributeIterator ()

Default constructor.

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

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

Protected constructor used by the MessageAttributes class.

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

Parameters:

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

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

5.1.3 Member Function Documentation

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

Predicate method for iteration termination.

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

Returns:

Returns true if there are more values, otherwise false.

5.1.3.2 const std::string& Arc::AttributeIterator::operator * () const

The dereference operator.

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

Returns:

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

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

The postfix advance operator.

Advances the iterator to the next value. Works intuitively.

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

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

The prefix advance operator.

Advances the iterator to the next value. Works intuitively.

Returns:
A const reference to this iterator.

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

The arrow operator.

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

5.1.4 Friends And Related Function Documentation

5.1.4.1 friend class MessageAttributes [friend]

The MessageAttributes class is a friend.

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

5.1.5 Member Data Documentation

5.1.5.1 AttrConstIter Arc::AttributeIterator::current_ [protected]

A const_iterator pointing to the current key-value pair.

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

5.1.5.2 AttrConstIter Arc::AttributeIterator::end_ [protected]

A const_iterator pointing beyond the last key-value pair.

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

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

- MessageAttributes.h

5.2 authnhandler_descriptor Struct Reference

`#include <AuthNHandlerLoader.h>`

Public Attributes

- const char * **name**
- int **version**
- Arc::AuthNHandler *(* **get_instance**)(Arc::Config *cfg)
- const char * **name**
- Arc::AuthNHandler *(* **get_instance**)(Arc::Config *cfg)

5.2.1 Detailed Description

This structure describes set of authentication handlers stored in shared library. It contains name of plugin, version number and pointer to function which creates an instance of object inherited from AuthNHandler class.

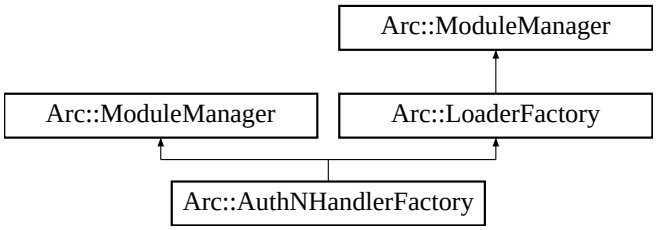
The documentation for this struct was generated from the following files:

- AuthNHandlerLoader.h
- BAK/AuthNHandlerLoader.h

5.3 Arc::AuthNHandlerFactory Class Reference

`#include <AuthNHandlerFactory.h>`

Inheritance diagram for Arc::AuthNHandlerFactory::



Public Member Functions

- AuthNHandlerFactory (Config *cfg)
- AuthNHandler * get_instance (const std::string &name, Arc::Config *cfg)
- AuthNHandler * **get_instance** (const std::string &name, int version, Arc::Config *cfg)
- AuthNHandler * **get_instance** (const std::string &name, int min_version, int max_version, Arc::Config *cfg)
- AuthNHandlerFactory (Config *cfg)
- AuthNHandler * get_instance (const std::string &name, Arc::Config *cfg)
- AuthNHandler * **get_instance** (const std::string &name, int version, Arc::Config *cfg)
- AuthNHandler * **get_instance** (const std::string &name, int min_version, int max_version, Arc::Config *cfg)
- void **load_all_instances** (const std::string &libname)

5.3.1 Detailed Description

This class handles shared libraries containing authentication handlers

5.3.2 Constructor & Destructor Documentation

5.3.2.1 Arc::AuthNHandlerFactory::AuthNHandlerFactory (Config * *cfg*)

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

5.3.2.2 Arc::AuthNHandlerFactory::AuthNHandlerFactory (Config * *cfg*)

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

5.3.3 Member Function Documentation

5.3.3.1 AuthNHandler* Arc::AuthNHandlerFactory::get_instance (const std::string & *name*, Arc::Config * *cfg*)

This method loads shared library named lib'name', locates symbol representing descriptor of AuthNHandler and calls it's constructor function. Supplied configuration tree is passed to constructor. Returns

created AuthNHandler instance.

Reimplemented from Arc::LoaderFactory.

5.3.3.2 AuthNHandler* Arc::AuthNHandlerFactory::get_instance (const std::string & *name*, Arc::Config * *cfg*)

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

Reimplemented from Arc::LoaderFactory.

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

- AuthNHandlerFactory.h
- BAK/AuthNHandlerFactory.h

5.4 authzhandler_descriptor Struct Reference

`#include <AuthZHandlerLoader.h>`

Public Attributes

- const char * **name**
- int **version**
- Arc::AuthZHandler *(* **get_instance**)(Arc::Config *cfg)
- const char * **name**
- Arc::AuthZHandler *(* **get_instance**)(Arc::Config *cfg)

5.4.1 Detailed Description

This structure describes set of authorization handlers stored in shared library. It contains name of plugin, version number and pointer to function which creates an instance of object inherited from AuthZHandler class.

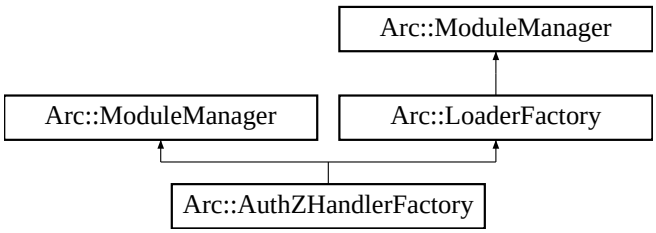
The documentation for this struct was generated from the following files:

- AuthZHandlerLoader.h
- AuthNZandlerLoader.h

5.5 Arc::AuthZHandlerFactory Class Reference

`#include <AuthZHandlerFactory.h>`

Inheritance diagram for Arc::AuthZHandlerFactory::



Public Member Functions

- AuthZHandlerFactory (Config *cfg)
- AuthZHandler * get_instance (const std::string &name, Arc::Config *cfg)
- AuthZHandler * **get_instance** (const std::string &name, int version, Arc::Config *cfg)
- AuthZHandler * **get_instance** (const std::string &name, int min_version, int max_version, Arc::Config *cfg)
- AuthZHandlerFactory (Config *cfg)
- AuthZHandler * get_instance (const std::string &name, Arc::Config *cfg)
- AuthZHandler * **get_instance** (const std::string &name, int version, Arc::Config *cfg)
- AuthZHandler * **get_instance** (const std::string &name, int min_version, int max_version, Arc::Config *cfg)
- void **load_all_instances** (const std::string &libname)

5.5.1 Detailed Description

This class handles shared libraries containing authentication handlers

5.5.2 Constructor & Destructor Documentation

5.5.2.1 Arc::AuthZHandlerFactory::AuthZHandlerFactory (Config * *cfg*)

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

5.5.2.2 Arc::AuthZHandlerFactory::AuthZHandlerFactory (Config * *cfg*)

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

5.5.3 Member Function Documentation

5.5.3.1 AuthZHandler* Arc::AuthZHandlerFactory::get_instance (const std::string & *name*, Arc::Config * *cfg*)

This method loads shared library named lib'name', locates symbol representing descriptor of AuthZHandler and calls it's constructor function. Supplied configuration tree is passed to constructor. Returns

created AuthZHandler instance.

Reimplemented from Arc::LoaderFactory.

5.5.3.2 AuthZHandler* Arc::AuthZHandlerFactory::get_instance (const std::string & *name*, Arc::Config * *cfg*)

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

Reimplemented from Arc::LoaderFactory.

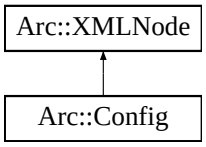
The documentation for this class was generated from the following files:

- AuthZHandlerFactory.h
- BAK/AuthZHandlerFactory.h

5.6 Arc::Config Class Reference

`#include <ArcConfig.h>`

Inheritance diagram for Arc::Config::



Public Member Functions

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

5.6.1 Detailed Description

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

5.6.2 Constructor & Destructor Documentation

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

Dummy constructor - produces empty structure

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

Loads configuration document from file 'filename'

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

Parse configuration document from memory

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

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

5.6.3 Member Function Documentation

5.6.3.1 void Arc::Config::print (void)

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

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

- ArcConfig.h

5.7 Arc::Loader Class Reference

`#include <Loader.h>`

Public Types

- typedef std::map< std::string, MCC * > **mcc_container_t**
- typedef std::map< std::string, Service * > **service_container_t**
- typedef std::map< std::string, Plexer * > **plexer_container_t**
- typedef std::map< std::string, MCC * > **mcc_container_t**
- typedef std::map< std::string, Service * > **service_container_t**
- typedef std::map< std::string, Plexer * > **plexer_container_t**
- typedef std::map< std::string, AuthNHandler * > **authn_container_t**
- typedef std::map< std::string, AuthZHandler * > **authz_container_t**

Public Member Functions

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

5.7.1 Detailed Description

This class processes XML configration and creates message chains. Accepted configuration is defined by XML schema mcc.xsd. Supported components are of types MCC, Service and Plexer. MCC and Service are loaded from dynamic libraries. For Plexer only internal implementation is supported. This object is also a container for loaded componets. All components are destroyed if this object is destroyed. Chains are created in 2 steps. First all components are loaded and corresponding objects are created. Constructor are supplied with corresponding configuration subtrees. During next step components are linked together by calling their Next() methods. Each creates labeled link to next component in a chain. 2 step method has an advantage over 1 step because it allows loops in chains and makes loading procedure simple. But that also means during short period of time components are only partly160configured. Components in such state must produce proper error response if Message arrives. Note: Current implementation requires all components and links to be labeled. All labels must be unique.

5.7.2 Constructor & Destructor Documentation

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

Constructor takes whole XML configuration and creates components' chains

5.7.2.2 Arc::Loader::~Loader (void)

Destructor destroys all components created by constructor

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

Constructor takes whole XML configuration and creates components' chains

5.7.2.4 Arc::Loader::~Loader (void)

Destructor destroys all components created by constructor

5.7.3 Member Function Documentation

5.7.3.1]

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

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

5.7.3.2]

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 files:

- BAK/Loader.h
- Loader.h

5.8 Arc::loader_descriptor Struct Reference

`#include <LoaderFactory.h>`

Public Attributes

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

5.8.1 Detailed Description

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

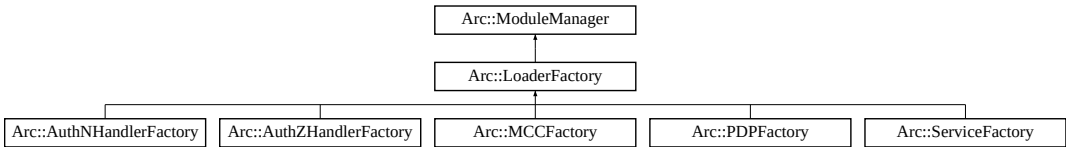
The documentation for this struct was generated from the following file:

- LoaderFactory.h

5.9 Arc::LoaderFactory Class Reference

`#include <LoaderFactory.h>`

Inheritance diagram for Arc::LoaderFactory::



Public Member Functions

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

Protected Member Functions

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

5.9.1 Detailed Description

This class handles shared libraries containing loadable classes

5.9.2 Constructor & Destructor Documentation

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

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

5.9.3 Member Function Documentation

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

These methods load shared library named lib'name', locates symbol 'id' representing descriptor of elements and calls it's constructor function. Supplied configuration tree is passed to constructor. Returns created instance. This classes must be rewritten in real implementation with proper type casting.

Reimplemented in Arc::AuthNHandlerFactory, Arc::AuthZHandlerFactory, Arc::AuthNHandlerFactory, Arc::AuthZHandlerFactory, Arc::MCCFactory, Arc::MCCFactory, Arc::ServiceFactory, Arc::MCCFactory, Arc::PDPFactory, and Arc::ServiceFactory.

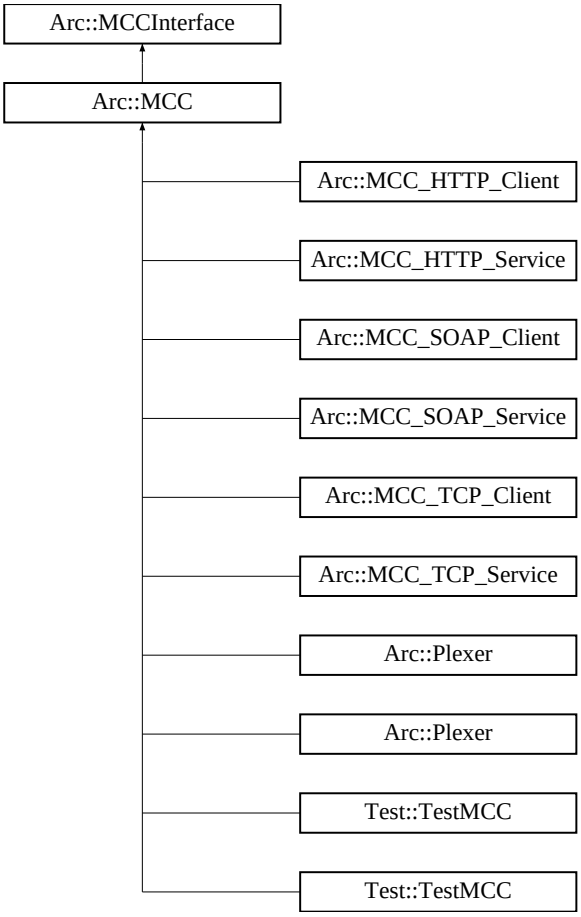
The documentation for this class was generated from the following file:

- LoaderFactory.h

5.10 Arc::MCC Class Reference

`#include <MCC.h>`

Inheritance diagram for Arc::MCC::



Public Member Functions

- MCC (Arc::Config *cfg)
- virtual void Next (MCCInterface *next, const std::string &label="")
- virtual void **AuthN** (AuthNHandler *authn, const std::string &label="")
- virtual void **AuthZ** (AuthZHandler *authz, const std::string &label="")
- virtual void Unlink (void)
- virtual MCC_Status process (Message &request, Message &response)

Protected Member Functions

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

Protected Attributes

- std::map< std::string, MCCInterface * > next_

- std::map< std::string, std::list< AuthNHandler * > > authn_
- std::map< std::string, std::list< AuthZHandler * > > **authz_**

5.10.1 Detailed Description

Message Chain Component - base class for every MCC plugin. This is partialy virtual class which defines interface and common functionality for every MCC plugin needed for managing of component in a chain.

5.10.2 Constructor & Destructor Documentation

5.10.2.1 Arc::MCC::MCC (Arc::Config * *cfg*) [inline]

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

5.10.3 Member Function Documentation

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

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

5.10.3.2 virtual MCC_Status Arc::MCC::process (Message & *request*, Message & *response*) [inline, virtual]

Dummy Message processing method. Just a placeholder.

Implements Arc::MCCInterface.

Reimplemented in Arc::MCC_HTTP_Service, Arc::MCC_HTTP_Client, Arc::MCC_SOAP_Service, Arc::MCC_TCP_Service, and Arc::MCC_TCP_Client.

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

Removing all links. Useful for destroying chains.

5.10.4 Member Data Documentation

5.10.4.1 std::map<std::string,std::list<AuthNHandler*> > Arc::MCC::authn_ [protected]

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

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

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

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

- MCC.h

5.11 mcc_descriptor Struct Reference

`#include <MCCLoader.h>`

Public Attributes

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

5.11.1 Detailed Description

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

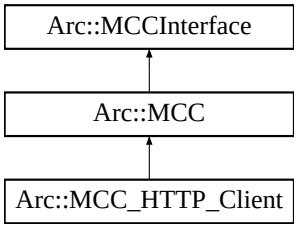
The documentation for this struct was generated from the following files:

- BAK/MCCLoader.h
- MCCLoader.h

5.12 Arc::MCC_HTTP_Client Class Reference

`#include <MCCHTTP.h>`

Inheritance diagram for Arc::MCC_HTTP_Client::



Public Member Functions

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

Protected Attributes

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

5.12.1 Detailed Description

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

5.12.2 Member Function Documentation

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

Dummy Message processing method. Just a placeholder.

Reimplemented from Arc::MCC.

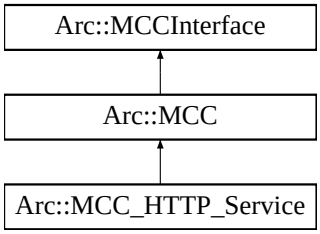
The documentation for this class was generated from the following file:

- MCCHTTP.h

5.13 Arc::MCC_HTTP_Service Class Reference

`#include <MCCHTTP.h>`

Inheritance diagram for Arc::MCC_HTTP_Service::



Public Member Functions

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

5.13.1 Detailed Description

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

5.13.2 Member Function Documentation

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

Dummy Message processing method. Just a placeholder.

Reimplemented from Arc::MCC.

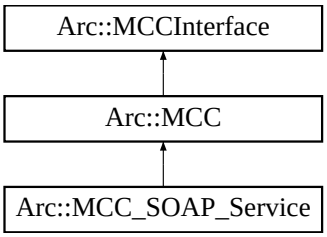
The documentation for this class was generated from the following file:

- MCCHTTP.h

5.14 Arc::MCC_SOAP_Service Class Reference

`#include <MCCSOAP.h>`

Inheritance diagram for Arc::MCC_SOAP_Service::



Public Member Functions

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

5.14.1 Detailed Description

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

5.14.2 Member Function Documentation

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

Dummy Message processing method. Just a placeholder.

Reimplemented from Arc::MCC.

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

- MCCSOAP.h

5.15 Arc::MCC_Status Class Reference

`#include <MCC.h>`

Public Member Functions

- **MCC_Status** (int code=0)
- **operator int** (void)
- MCC_Status & **operator=** (int code)
- **operator bool** (void)
- bool **operator!** (void)

Protected Attributes

- int **code_**

5.15.1 Detailed Description

This class represents status of Message processing. Currently it's just a placeholder for int code with 0 meaning there were no errors during processing. It's methods allow it to be treated as ordinary int as well. The precise meaning of non-zero values and other extensions have to be decided. The purpose of such object is to indicate if message was processed at endpoint Service or it hasn't reached it due to error in it's path.

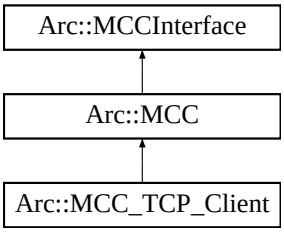
The documentation for this class was generated from the following file:

- MCC.h

5.16 Arc::MCC_TCP_Client Class Reference

`#include <MCCTCP.h>`

Inheritance diagram for Arc::MCC_TCP_Client::



Public Member Functions

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

5.16.1 Detailed Description

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

5.16.2 Member Function Documentation

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

Dummy Message processing method. Just a placeholder.

Reimplemented from Arc::MCC.

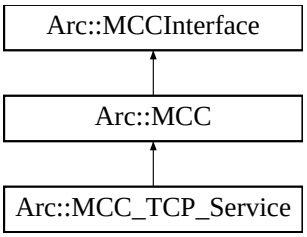
The documentation for this class was generated from the following file:

- MCCTCP.h

5.17 Arc::MCC_TCP_Service Class Reference

`#include <MCCTCP.h>`

Inheritance diagram for Arc::MCC_TCP_Service::



Public Member Functions

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

Friends

- class **mcc_tcp_exec_t**

5.17.1 Detailed Description

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

5.17.2 Constructor & Destructor Documentation

5.17.2.1 Arc::MCC_TCP_Service::MCC_TCP_Service (Arc::Config * *cfg*)

executing function for connection thread

5.17.3 Member Function Documentation

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

Dummy Message processing method. Just a placeholder.

Reimplemented from Arc::MCC.

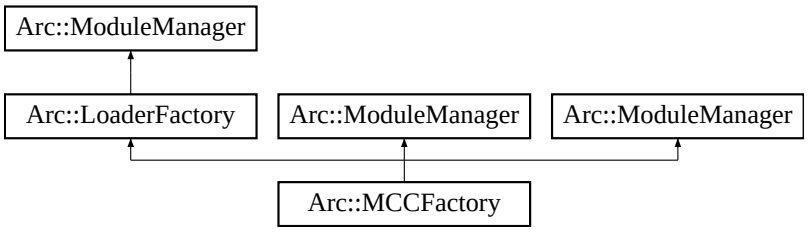
The documentation for this class was generated from the following file:

- MCCTCP.h

5.18 Arc::MCCFactory Class Reference

`#include <Factory.h>`

Inheritance diagram for Arc::MCCFactory::



Public Member Functions

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

- void **load_all_instances** (const std::string &libname)
- MCCFactory (Config *cfg)
- MCC * get_instance (const std::string &name, Arc::Config *cfg)
- MCC * **get_instance** (const std::string &name, int version, Arc::Config *cfg)
- MCC * **get_instance** (const std::string &name, int min_version, int max_version, Arc::Config *cfg)

- void **load_all_instances** (const std::string &libname)
- MCCFactory (Config *cfg)
- MCC * get_instance (const std::string &name, Arc::Config *cfg)
- MCC * **get_instance** (const std::string &name, int version, Arc::Config *cfg)
- MCC * **get_instance** (const std::string &name, int min_version, int max_version, Arc::Config *cfg)

5.18.1 Detailed Description

This class handles shared libraries containing MCCs

5.18.2 Constructor & Destructor Documentation

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

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

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

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

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

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

5.18.3 Member Function Documentation

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

This methods load shared library named lib'name', locates 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.

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

This method loads shared library named lib'name', locates 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.

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

This method loads shared library named lib'name', locates 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.

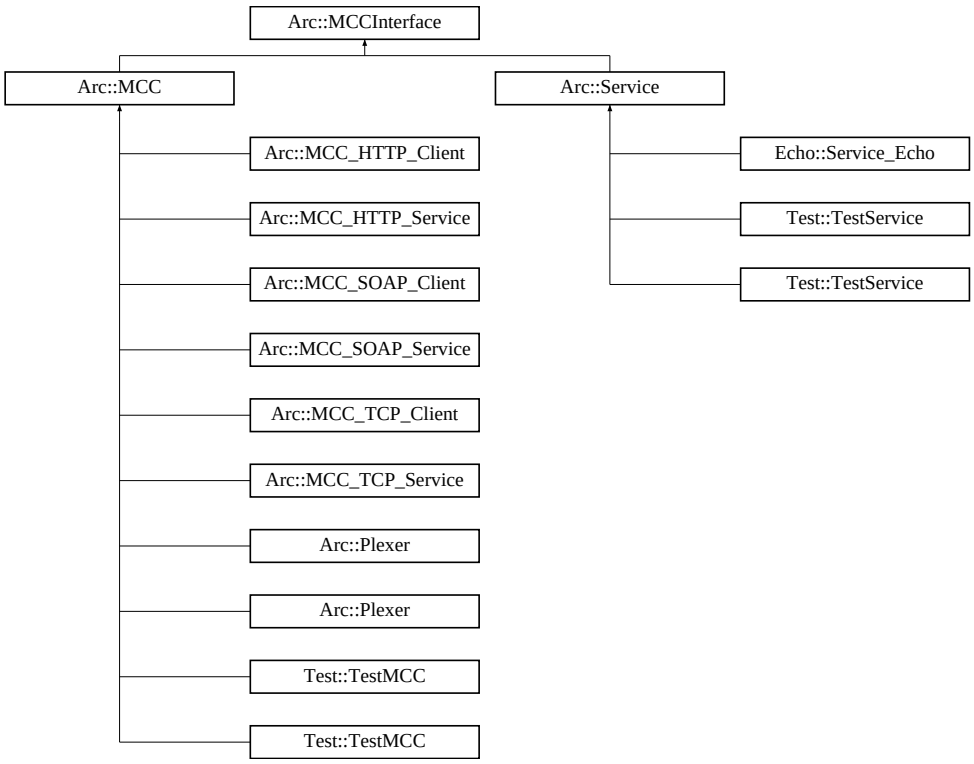
The documentation for this class was generated from the following files:

- Factory.h
- BAK/MCCFactory.h
- MCCFactory.h

5.19 Arc::MCCInterface Class Reference

`#include <MCC.h>`

Inheritance diagram for Arc::MCCInterface::



Public Member Functions

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

5.19.1 Detailed Description

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

5.19.2 Member Function Documentation

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

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

Parameters:

request The request that needs to be processed.

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

Returns:

An object representing the status of the call.

Implemented in Arc::MCC, Arc::MCC_HTTP_Service, Arc::MCC_HTTP_Client, Arc::MCC_SOAP_-Service, Arc::MCC_TCP_Service, Arc::MCC_TCP_Client, and Echo::Service_Echo.

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

- MCC.h

5.20 Arc::Message Class Reference

`#include <Message.h>`

Public Member Functions

- Message (void)
- Message (Message &msg)
- ~Message (void)
- Message & operator= (Message &msg)
- MessagePayload * Payload (void)
- MessagePayload * Payload (MessagePayload *new_payload)

5.20.1 Detailed Description

Message is passed through chain of MCCs. It refers to objects with main content (payload), authentication/authorization information and common purpose attributes. Message class does not manage pointers to objects and theur content. it only serves for grouping those objects. Message objects are supposed to be processed by objects' implementing MCCInterface method process(). All objects constituting content of Message object are subject to following policies:

1. All objects created inside call to process() method using new command must be explicitely destroyed within same call using delete command with following exceptions. a) Objects which are assigned to 'response' Message. b) Objects whose management is completely acquired by objects assigned to 'response' Message.

2. All objects not created inside call to process() method are not explicitely destroyed within that call with following exception. a) Objects which are part of 'response' Method returned from call to next's process() method. Unless those objects are passed further to calling process(), of course.

3. It is not allowed to make 'response' point to same objects as 'request' does on entry to process() method. That is needed to avoid double destruction of same object. (Note: if in a future such need arises it may be solved by storing additional flags in Message object).

4. It is allowed to change content of pointers of 'request' Message. Calling process() method mus tnot rely on that object to stay intact.

5. Called process() method should either fill 'response' Message with pointers to valid objects or to keep them intact. This makes it possible for calling process() to preload 'response' with valid error message.

5.20.2 Constructor & Destructor Documentation

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

Dummy constructor

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

Copy constructor. Ensures shallow copy.

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

Destructor does not affect refered objects

5.20.3 Member Function Documentation

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

Assignment. Ensures shallow copy.

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

Replace payload with new one

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

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

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

- Message.h

5.21 Arc::MessageAttributes Class Reference

A class for storage of attribute values.

`#include <MessageAttributes.h>`

Public Member Functions

- MessageAttributes ()
- void set (const std::string &key, const std::string &value)
- void add (const std::string &key, const std::string &value)
- void removeAll (const std::string &key)
- void remove (const std::string &key, const std::string &value)
- int count (const std::string &key) const
- const std::string & get (const std::string &key) const
- AttributeIterator getAll (const std::string &key) const

Protected Attributes

- AttrMap attributes_

5.21.1 Detailed Description

A class for storage of attribute values.

This class is used to store attributes of messages. All attribute keys and their corresponding values are stored as strings. Any key or value that is not a string must thus be represented as a string during storage. Furthermore, an attribute is usually a key-value pair with a unique key, but there may also be multiple such pairs with equal keys.

The key of an attribute is composed by the name of the Message Chain Component (MCC) which produce it and the name of the attribute itself with a colon (:) in between, i.e. MCC_Name:Attribute_Name. For example, the key of the "Content-Length" attribute of the HTTP MCC is thus "HTTP:Content-Length".

There are also "global attributes", which may be produced by different MCCs depending on the configuration. The keys of such attributes are NOT prefixed by the name of the producing MCC. Before any new global attribute is introduced, it must be agreed upon by the core development team and added below. The global attributes decided so far are:

- `Request-URI` Identifies the service to which the message shall be sent. This attribute is produced by e.g. the HTTP MCC and used by the plexer for routing the message to the appropriate service.

5.21.2 Constructor & Destructor Documentation

5.21.2.1 Arc::MessageAttributes::MessageAttributes ()

The default constructor.

This is the default constructor of the MessageAttributes class. It constructs an empty object that initially contains no attributes.

5.21.3 Member Function Documentation

5.21.3.1 void Arc::MessageAttributes::add (const std::string & *key*, const std::string & *value*)

Adds a value to an attribute.

This method adds a new value to an attribute. Any previous value will be preserved, i.e. the attribute may become multiple valued.

Parameters:

key The key of the attribute.

value The (new) value of the attribute.

5.21.3.2 int Arc::MessageAttributes::count (const std::string & *key*) const

Returns the number of values of an attribute.

Returns the number of values of an attribute that matches a certain key.

Parameters:

key The key of the attribute for which to count values.

Returns:

The number of values that corresponds to the key.

5.21.3.3 const std::string& Arc::MessageAttributes::get (const std::string & *key*) const

Returns the value of a single-valued attribute.

This method returns the value of a single-valued attribute. If the attribute is not single valued (i.e. there is no such attribute or it is a multiple-valued attribute) an empty string is returned.

Parameters:

key The key of the attribute for which to return the value.

Returns:

The value of the attribute.

5.21.3.4 AttributeIterator Arc::MessageAttributes::getAll (const std::string & *key*) const

Access the value(s) of an attribute.

This method returns an AttributeIterator that can be used to access the values of an attribute.

Parameters:

key The key of the attribute for which to return the values.

Returns:

An AttributeIterator for access of the values of the attribute.

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

Removes one value of an attribute.

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

Parameters:

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

value The value to remove.

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

Removes all attributes with a certain key.

This method removes all attributes that match a certain key.

Parameters:

key The key of the attributes to remove.

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

Sets a unique value of an attribute.

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

Parameters:

key The key of the attribute.

value The (new) value of the attribute.

5.21.4 Member Data Documentation

5.21.4.1 AttrMap Arc::MessageAttributes::attributes_ [protected]

Internal storage of attributes.

An AttrMap (multimap) in which all attributes (key-value pairs) are stored.

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

- MessageAttributes.h

5.22 Arc::MessageAuth Class Reference

`#include <MessageAuth.h>`

Public Member Functions

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

5.22.1 Detailed Description

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

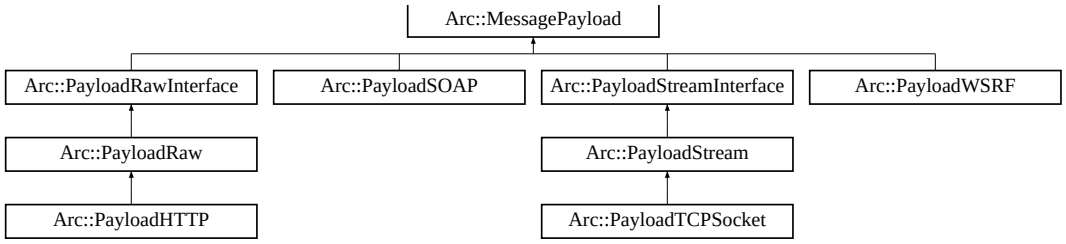
The documentation for this class was generated from the following file:

- MessageAuth.h

5.23 Arc::MessagePayload Class Reference

`#include <Message.h>`

Inheritance diagram for Arc::MessagePayload::



5.23.1 Detailed Description

Base class for content of message passed through chain. It's not intended to be used directly. Instead functional classes must be derived from it.

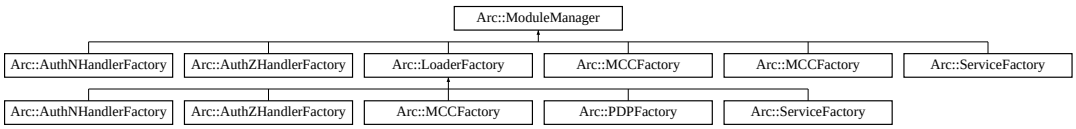
The documentation for this class was generated from the following file:

- Message.h

5.24 Arc::ModuleManager Class Reference

`#include <ModuleManager.h>`

Inheritance diagram for Arc::ModuleManager::



Public Member Functions

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

5.24.1 Detailed Description

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

5.24.2 Constructor & Destructor Documentation

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

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

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

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

5.24.3 Member Function Documentation

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

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

5.24.3.2 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 files:

- BAK/ModuleManager.h
- ModuleManager.h

5.25 ns1__echoRequest Class Reference

"urn:echo":echoRequest is a complexType.

`#include <echo.h>`

Public Member Functions

- virtual int **soap_type** () const
- virtual void **soap_default** (struct soap *)
- virtual void **soap_serialize** (struct soap *) const
- virtual int **soap_put** (struct soap *, const char *, const char *) const
- virtual int **soap_out** (struct soap *, const char *, int, const char *) const
- virtual void * **soap_get** (struct soap *, const char *, const char *)
- virtual void * **soap_in** (struct soap *, const char *, const char *)

Public Attributes

- ns1__say say
- soap * soap
- std::string **say**
- soap * **soap**

5.25.1 Detailed Description

"urn:echo":echoRequest is a complexType.

5.25.2 Member Data Documentation

5.25.2.1 ns1__say ns1__echoRequest::say

Element say of type "urn:echo":say.

Required element.

5.25.2.2 struct soap* ns1__echoRequest::soap

A handle to the soap struct that manages this instance (automatically set).

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

- wsdl/echo.h
- soapStub.h

5.26 ns1__echoResponse Class Reference

"urn:echo":echoResponse is a complexType.

`#include <echo.h>`

Public Member Functions

- virtual int **soap_type** () const
- virtual void **soap_default** (struct soap *)
- virtual void **soap_serialize** (struct soap *) const
- virtual int **soap_put** (struct soap *, const char *, const char *) const
- virtual int **soap_out** (struct soap *, const char *, int, const char *) const
- virtual void * **soap_get** (struct soap *, const char *, const char *)
- virtual void * **soap_in** (struct soap *, const char *, const char *)

Public Attributes

- ns1__hear hear
- soap * soap
- std::string **hear**
- soap * **soap**

5.26.1 Detailed Description

"urn:echo":echoResponse is a complexType.

5.26.2 Member Data Documentation

5.26.2.1 ns1__hear ns1__echoResponse::hear

Element hear of type "urn:echo":hear.

Required element.

5.26.2.2 struct soap* ns1__echoResponse::soap

A handle to the soap struct that manages this instance (automatically set).

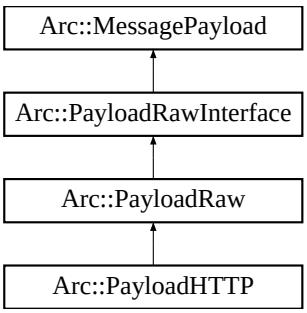
The documentation for this class was generated from the following files:

- wsdl/echo.h
- soapStub.h

5.27 Arc::PayloadHTTP Class Reference

`#include <PayloadHTTP.h>`

Inheritance diagram for Arc::PayloadHTTP::



Public Member Functions

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

Protected Member Functions

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

Protected Attributes

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

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

5.27.1 Detailed Description

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

5.27.2 Constructor & Destructor Documentation

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

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

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

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

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

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

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

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

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

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

5.27.3 Member Function Documentation

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

Sets HTTP header attribute 'name' to 'value'

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

Returns HTTP header attribute with specified name. Empty string if nosuch attribute.

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

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

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

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

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

Read HTTP header and fill internal variables

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

Read up to 'size' bytes from stream_

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

Read from stream till

5.27.4 Member Data Documentation

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

true if content is chunked

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

Content-length of HTTP message

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

HTTP method being used or requested

5.27.4.4 int Arc::PayloadHTTP::length_ [protected]

HTTP reason being sent or supplied

5.27.4.5 std::string Arc::PayloadHTTP::method_ [protected]

minor number of HTTP version - must be 0 or 1

5.27.4.6 std::string Arc::PayloadHTTP::reason_ [protected]

HTTP code being sent or supplied

5.27.4.7 std::string Arc::PayloadHTTP::uri_ [protected]

stream used to comminicate to outside

5.27.4.8 int Arc::PayloadHTTP::version_major_ [protected]

URI being contacted

5.27.4.9 int Arc::PayloadHTTP::version_minor_ [protected]

major number of HTTP version - must be 1

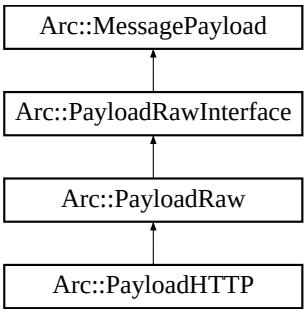
The documentation for this class was generated from the following file:

- PayloadHTTP.h

5.28 Arc::PayloadRaw Class Reference

`#include <PayloadRaw.h>`

Inheritance diagram for Arc::PayloadRaw::



Public Member Functions

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

Protected Attributes

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

5.28.1 Detailed Description

Implementation of PayloadRawInterface - raw byte multi-buffer.

5.28.2 Constructor & Destructor Documentation

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

Constructor. Created object contains no buffers.

5.28.2.2 virtual Arc::PayloadRaw::~PayloadRaw (void) `[virtual]`

Destructor. Frees allocated buffers.

5.28.3 Member Function Documentation

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

Returns pointer to num'th buffer

Implements Arc::PayloadRawInterface.

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

Returns length of num'th buffer

Implements Arc::PayloadRawInterface.

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

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

Implements Arc::PayloadRawInterface.

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

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

Implements Arc::PayloadRawInterface.

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

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

Implements Arc::PayloadRawInterface.

5.28.3.6]

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

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

Implements Arc::PayloadRawInterface.

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

Returns cumulative length of all buffers

Implements Arc::PayloadRawInterface.

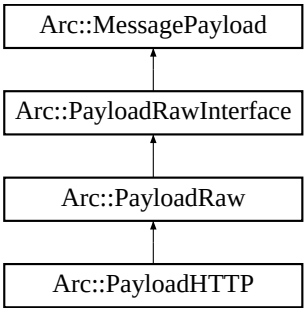
The documentation for this class was generated from the following file:

- PayloadRaw.h

5.29 Arc::PayloadRawInterface Class Reference

`#include <PayloadRaw.h>`

Inheritance diagram for Arc::PayloadRawInterface::



Public Member Functions

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

5.29.1 Detailed Description

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

5.29.2 Member Function Documentation

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

Returns pointer to num'th buffer

Implemented in Arc::PayloadRaw.

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

Returns length of num'th buffer

Implemented in Arc::PayloadRaw.

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

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

Implemented in Arc::PayloadRaw.

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

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

Implemented in Arc::PayloadRaw.

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

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

Implemented in Arc::PayloadRaw.

5.29.2.6]

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

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

Implemented in Arc::PayloadRaw.

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

Returns cumulative length of all buffers

Implemented in Arc::PayloadRaw.

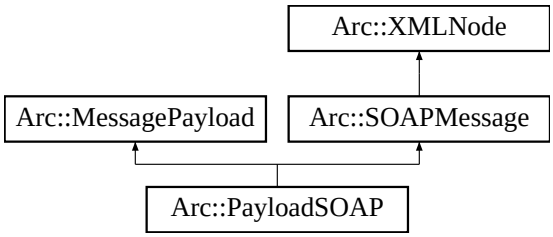
The documentation for this class was generated from the following file:

- PayloadRaw.h

5.30 Arc::PayloadSOAP Class Reference

`#include <PayloadSOAP.h>`

Inheritance diagram for Arc::PayloadSOAP::



Public Member Functions

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

5.30.1 Detailed Description

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

5.30.2 Constructor & Destructor Documentation

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

Constructor - creates new Message payload

5.30.2.2 Arc::PayloadSOAP::PayloadSOAP (const SOAPMessage & *soap*)

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

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

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

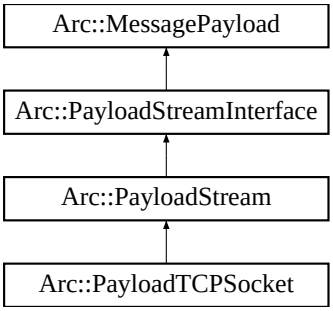
The documentation for this class was generated from the following file:

- PayloadSOAP.h

5.31 Arc::PayloadStream Class Reference

`#include <PayloadStream.h>`

Inheritance diagram for Arc::PayloadStream::



Public Member Functions

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

Protected Attributes

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

5.31.1 Detailed Description

Implemetation of PayloadStreamInterface for generic POSIX handle.

5.31.2 Constructor & Destructor Documentation

5.31.2.1 Arc::PayloadStream::PayloadStream (int *h* = -1)

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

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

Destructor.

5.31.3 Member Function Documentation

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

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

Implements Arc::PayloadStreamInterface.

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

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

Implements Arc::PayloadStreamInterface.

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

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

Implements Arc::PayloadStreamInterface.

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

Returns true if stream is valid.

Implements Arc::PayloadStreamInterface.

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

Returns true if stream is invalid.

Implements Arc::PayloadStreamInterface.

5.31.3.6 virtual bool Arc::PayloadStream::Put (const char * *buf*) [inline, virtual]

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

Implements Arc::PayloadStreamInterface.

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

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

Implements Arc::PayloadStreamInterface.

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

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

Implements Arc::PayloadStreamInterface.

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

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

Implements Arc::PayloadStreamInterface.

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

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

Implements Arc::PayloadStreamInterface.

5.31.4 Member Data Documentation

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

Timeout for read/write operations

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

Handle for operations

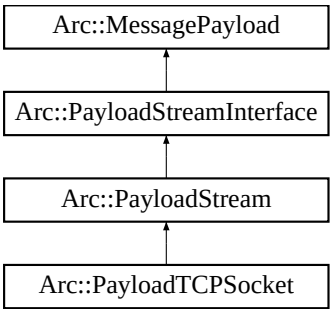
The documentation for this class was generated from the following file:

- PayloadStream.h

5.32 Arc::PayloadStreamInterface Class Reference

`#include <PayloadStream.h>`

Inheritance diagram for Arc::PayloadStreamInterface::



Public Member Functions

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

5.32.1 Detailed Description

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

5.32.2 Member Function Documentation

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

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

Implemented in Arc::PayloadStream.

5.32.2.2 virtual bool Arc::PayloadStreamInterface::Get (std::string & *buf*) [pure virtual]

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

Implemented in Arc::PayloadStream.

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

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

Implemented in Arc::PayloadStream.

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

Returns true if stream is valid.

Implemented in Arc::PayloadStream.

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

Returns true if stream is invalid.

Implemented in Arc::PayloadStream.

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

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

Implemented in Arc::PayloadStream.

5.32.2.7 virtual bool Arc::PayloadStreamInterface::Put (const std::string & *buf*) [pure virtual]

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

Implemented in Arc::PayloadStream.

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

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

Implemented in Arc::PayloadStream.

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

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

Implemented in Arc::PayloadStream.

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

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

Implemented in Arc::PayloadStream.

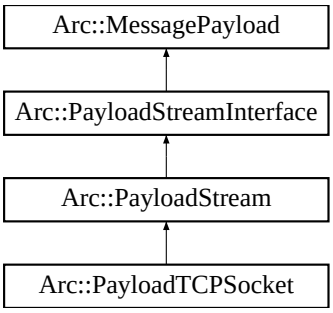
The documentation for this class was generated from the following file:

- PayloadStream.h

5.33 Arc::PayloadTCPSocket Class Reference

`#include <PayloadTCPSocket.h>`

Inheritance diagram for Arc::PayloadTCPSocket::



Public Member Functions

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

5.33.1 Detailed Description

This class extends PayloadStream with TCP socket specific features

5.33.2 Constructor & Destructor Documentation

5.33.2.1 Arc::PayloadTCPSocket::PayloadTCPSocket (const char * *hostname*, int *port*)

Constructor - connects to TCP server at specified hostname:port

5.33.2.2 Arc::PayloadTCPSocket::PayloadTCPSocket (const std::string *endpoint*)

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

5.33.2.3 Arc::PayloadTCPSocket::PayloadTCPSocket (int *s*) [inline]

Constructor - creates object of already connected socket

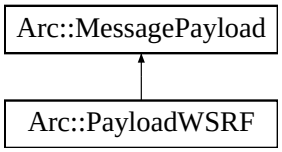
The documentation for this class was generated from the following file:

- PayloadTCPSocket.h

5.34 Arc::PayloadWSRF Class Reference

`#include <PayloadWSRF.h>`

Inheritance diagram for Arc::PayloadWSRF::



Public Member Functions

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

Protected Attributes

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

5.34.1 Detailed Description

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

5.34.2 Constructor & Destructor Documentation

5.34.2.1 Arc::PayloadWSRF::PayloadWSRF (const SOAPMessage & *soap*)

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

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

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

5.34.2.3 Arc::PayloadWSRF::PayloadWSRF (const MessagePayload & *source*)

Constructor - creates WSRF message from payload. All classes derived from SOAPMessage are supported.

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

- PayloadWSRF.h

5.35 pdp_descriptor Struct Reference

`#include <PDPLoader.h>`

Public Attributes

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

5.35.1 Detailed Description

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

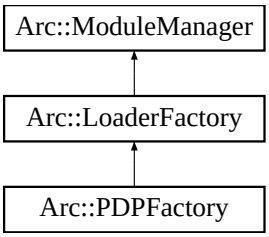
The documentation for this struct was generated from the following file:

- PDPLoader.h

5.36 Arc::PDPFactory Class Reference

`#include <PDPFactory.h>`

Inheritance diagram for Arc::PDPFactory::



Public Member Functions

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

5.36.1 Detailed Description

This class handles shared libraries containing authentication handlers

5.36.2 Constructor & Destructor Documentation

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

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

5.36.3 Member Function Documentation

5.36.3.1 PDP* Arc::PDPFactory::get_instance (const std::string & *name*, Arc::Config * *cfg*)

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

Reimplemented from Arc::LoaderFactory.

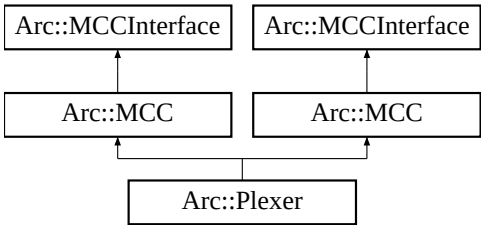
The documentation for this class was generated from the following file:

- PDPFactory.h

5.37 Arc::Plexer Class Reference

`#include <Plexer.h>`

Inheritance diagram for Arc::Plexer::



Public Member Functions

- Plexer (Arc::Config *cfg)
- void process (void)
- Plexer (Arc::Config *cfg)
- void process (void)

5.37.1 Detailed Description

This class routes Messages depending on their destination.

5.37.2 Constructor & Destructor Documentation

5.37.2.1 Arc::Plexer::Plexer (Arc::Config * *cfg*)

Constructor - it will take configuration tree and generate routing table of it

5.37.2.2 Arc::Plexer::Plexer (Arc::Config * *cfg*)

Constructor - it will take configuration tree and generate routing table of it

5.37.3 Member Function Documentation

5.37.3.1 void Arc::Plexer::process (void)

Entry point to Plexer. This method is called by last MCC in chain ending at Plexer.

5.37.3.2 void Arc::Plexer::process (void)

Entry point to Plexer. This method is called by last MCC in chain ending at Plexer.

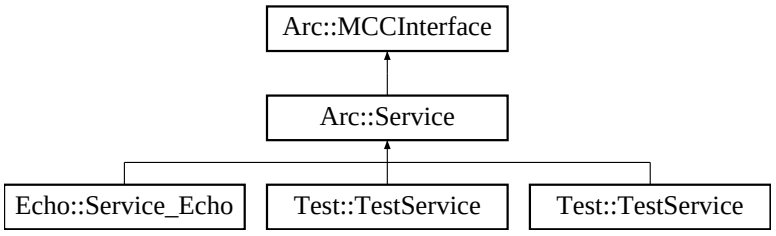
The documentation for this class was generated from the following files:

- BAK/Plexer.h
- Plexer.h

5.38 Arc::Service Class Reference

`#include <Service.h>`

Inheritance diagram for Arc::Service::



Public Member Functions

- Service (Arc::Config *cfg)
- virtual void **AuthN** (AuthNHandler *authn, const std::string &label="")
- virtual void **AuthZ** (AuthZHandler *authz, const std::string &label="")

Protected Attributes

- std::map< std::string, std::list< AuthNHandler * > > **authn_**
- std::map< std::string, std::list< AuthZHandler * > > **authz_**

5.38.1 Detailed Description

Service - last plugin in a Message Chain. This is virtual class which defines interface (in a future also common functionality) for every Service plugin. Interface is made of method process() which is called by Plexer or MCC class. There is one Service object created for every service description processed by Loader class objects. Classes derived from Service class must implement process() method of MCCInterface. It is up to developer how internal state of service is stored and communicated to other services and external utilites. Service is free to expect any type of payload passed to it and generate any payload as well. Useful types depend on MCCs in chain which leads to that service. For example if service is expected to by linked to SOAP MCC it must accespt and generate messages with PayloadSOAP payload. Method process() of class derived from Service class may be called concurently in multiple threads. Developers must take that into account and write thread-safe implementation. Simple example of service is provided in /src/tests/echo/echo.cpp . The way to write client couterpart of corresponding service is undefined. For example see /src/tests/echo/test.cpp .

5.38.2 Constructor & Destructor Documentation

5.38.2.1 Arc::Service::Service (Arc::Config * *cfg*) [inline]

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

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

- Service.h

5.39 service_descriptor Struct Reference

`#include <ServiceLoader.h>`

Public Attributes

- const char * **name**
- int **version**
- Arc::Service *(* **get_instance**)(Arc::Config *cfg)
- const char * **name**
- Arc::Service *(* **get_instance**)(Arc::Config *cfg)

5.39.1 Detailed Description

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

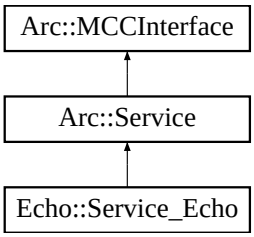
The documentation for this struct was generated from the following files:

- BAK/ServiceLoader.h
- ServiceLoader.h

5.40 Echo::Service_Echo Class Reference

`#include <echo.h>`

Inheritance diagram for Echo::Service_Echo::



Public Member Functions

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

Protected Member Functions

- Arc::MCC_Status **make_fault** (Arc::Message &outmsg)

Protected Attributes

- std::string **prefix_**
- std::string **suffix_**
- Arc::XMLNode::NS **ns_**

5.40.1 Detailed Description

This is a test service which accepts SOAP requests and produces response as described in echo.wsdl. Response contains string passed in request with prefix_ and suffix_ added.

5.40.2 Constructor & Destructor Documentation

5.40.2.1 Echo::Service_Echo::Service_Echo (Arc::Config * *cfg*)

Constructor accepts configuration describing content of prefix and suffix

5.40.3 Member Function Documentation

5.40.3.1 virtual Arc::MCC_Status Echo::Service_Echo::process (Arc::Message &, Arc::Message &) `[virtual]`

Service request processing routine

Implements Arc::MCCInterface.

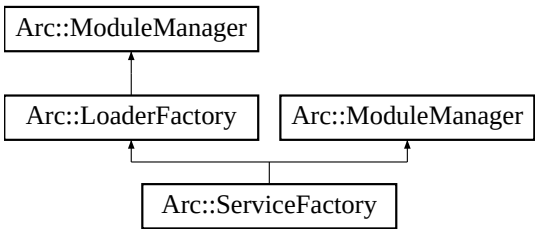
The documentation for this class was generated from the following file:

- echo.h

5.41 Arc::ServiceFactory Class Reference

`#include <ServiceFactory.h>`

Inheritance diagram for Arc::ServiceFactory::



Public Member Functions

- ServiceFactory (Arc::Config *cfg)
- Service * get_instance (const std::string &name, Arc::Config *cfg)
- Service * **get_instance** (const std::string &name, int version, Arc::Config *cfg)
- Service * **get_instance** (const std::string &name, int min_version, int max_version, Arc::Config *cfg)
- void **load_all_instances** (const std::string &libname)
- ServiceFactory (Arc::Config *cfg)
- Service * get_instance (const std::string &name, Arc::Config *cfg)
- Service * **get_instance** (const std::string &name, int version, Arc::Config *cfg)
- Service * **get_instance** (const std::string &name, int min_version, int max_version, Arc::Config *cfg)

5.41.1 Detailed Description

This class handles shared libraries containing Services

5.41.2 Constructor & Destructor Documentation

5.41.2.1 Arc::ServiceFactory::ServiceFactory (Arc::Config * *cfg*)

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

5.41.2.2 Arc::ServiceFactory::ServiceFactory (Arc::Config * *cfg*)

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

5.41.3 Member Function Documentation

5.41.3.1 Service* Arc::ServiceFactory::get_instance (const std::string & *name*, Arc::Config * *cfg*)

This 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.

5.41.3.2 Service* Arc::ServiceFactory::get_instance (const std::string & *name*, Arc::Config * *cfg*)

This method loads shared library named lib'name', locates 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 files:

- BAK/ServiceFactory.h
- ServiceFactory.h

5.42 Arc::SOAPFault Class Reference

`#include <SOAPMessage.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 **SOAPMessage**

5.42.1 Detailed Description

SOAPFault provides an interface to conveinet access to elements of SOAP faults. It also tries to expose single interface for both version 1.0 and 1.2 faults. This class is not intended to 'own' any information stored. It's purpose is to manipulate information which under control of XMLNode or SOAPMessage classes. If instance does not refer to valid SOAP Fault structure all manipulation methods will have no effect.

5.42.2 Member Enumeration Documentation

5.42.2.1 enum Arc::SOAPFault::SOAPFaultCode

Fault codes of SOAP specs

5.42.3 Constructor & Destructor Documentation

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

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

5.42.4 Member Function Documentation

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

Set Fault Code element

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

Returns Fault Code element

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

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

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

Set content of Fault Node element to 'node'

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

Returns content of Fault Node element

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

Returns true if instance refers to SOAP Fault

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

Set Fault Reason element at top level

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

Set Fault Reason content at various levels to 'reason'

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

Returns content of Fault Reason element at various levels

5.42.4.10 void Arc::SOAPFault::Role (const char * *role*)

Set content of Fault Role element to 'role'

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

Returns content of Fault Role element

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

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

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

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

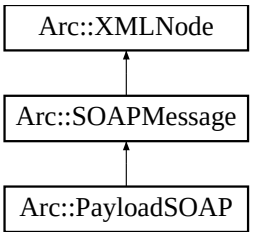
The documentation for this class was generated from the following file:

- SOAPMessage.h

5.43 Arc::SOAPMessage Class Reference

`#include <SOAPMessage.h>`

Inheritance diagram for Arc::SOAPMessage::



Public Member Functions

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

5.43.1 Detailed Description

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

5.43.2 Constructor & Destructor Documentation

5.43.2.1 Arc::SOAPMessage::SOAPMessage (const std::string & *xml*)

Create new SOAP message from textual representation of XML document. Created XML structure is owned by this instance. This constructor also sets default namespaces to default prefixes as specified below.

5.43.2.2 Arc::SOAPMessage::SOAPMessage (const char * *xml*, int *len* = -1)

Same as previous

5.43.2.3 Arc::SOAPMessage::SOAPMessage (const NS & *ns*, bool *fault* = false)

Create new SOAP message with specified namespaces. Created XML structure is owned by this instance. If argument fault is set to true created message is fault.

5.43.3 Member Function Documentation

5.43.3.1 void Arc::SOAPMessage::GetXML (std::string & *xml*) const

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

Reimplemented from Arc::XMLNode.

5.43.3.2 void Arc::SOAPMessage::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.

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

- SOAPMessage.h

5.44 Arc::WSAEndpointReference Class Reference

`#include <WSA.h>`

Public Member Functions

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

Protected Attributes

- XMLNode **epr_**

5.44.1 Detailed Description

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

5.44.2 Constructor & Destructor Documentation

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

Linking to existing EPR in XML tree

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

Creating independent EPR - not implemented

5.44.2.3 Arc::WSAEndpointReference::WSAEndpointReference (void)

Dummy constructor - creates invalid instance

5.44.2.4 Arc::WSAEndpointReference::~WSAEndpointReference (void)

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

5.44.3 Member Function Documentation

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

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

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

Returns Address (URL) encoded in EPR

5.44.3.3 XMLNode Arc::WSAEndpointReference::MetaData (void)

Access to MetaData element of EPR. Obtained XML element should be manipulated directly in application-dependent way. If EPR had no MetaData element it is created.

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

Returns reference to EPR top XML node

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

Same as Address(uri)

5.44.3.6 XMLNode Arc::WSAEndpointReference::ReferenceParameters (void)

Access to ReferenceParameters element of EPR. Obtained XML element should be manipulated directly in application-dependent way. If EPR had no ReferenceParameters element it is created.

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

- WSA.h

5.45 Arc::WSAHeader Class Reference

`#include <WSA.h>`

Public Member Functions

- WSAHeader (SOAPMessage &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)

Protected Attributes

- XMLNode **header_**
- bool header_allocated_

5.45.1 Detailed Description

Interface to manipulate WS-Addressing related information in SOAP header

5.45.2 Constructor & Destructor Documentation

5.45.2.1 Arc::WSAHeader::WSAHeader (SOAPMessage & *soap*)

Linking to a header of existing SOAP message

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

Creating independent SOAP header - not implemented

5.45.3 Member Function Documentation

5.45.3.1 void Arc::WSAHeader::Action (const std::string & *uri*)

Set content of Action element of SOAP Header. If such element does not exist it's created.

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

Returns content of Action element of SOAP Header.

5.45.3.3 WSAEndpointReference Arc::WSAHeader::FaultTo (void)

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

5.45.3.4 WSAEndpointReference Arc::WSAHeader::From (void)

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

5.45.3.5 void Arc::WSAHeader::MessageID (const std::string & *uri*)

Set content of MessageID element of SOAP Header. If such element does not exist it's created.

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

Returns content of MessageID element of SOAP Header.

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

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

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

Returns reference to SOAP Header - not implemented

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

Returns first ReferenceParameter element with specified name

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

Return n-th ReferenceParameter element

5.45.3.11 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.

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

Returns content of RelatesTo element of SOAP Header.

5.45.3.13 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.

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

Returns content of RelationshipType element of SOAP Header.

5.45.3.15 WSAEndpointReference Arc::WSAHeader::ReplyTo (void)

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

5.45.3.16 void Arc::WSAHeader::To (const std::string & *uri*)

Set content of To element of SOAP Header. If such element does not exist it's created.

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

Returns content of To element of SOAP Header.

5.45.4 Member Data Documentation

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

SOAP header element

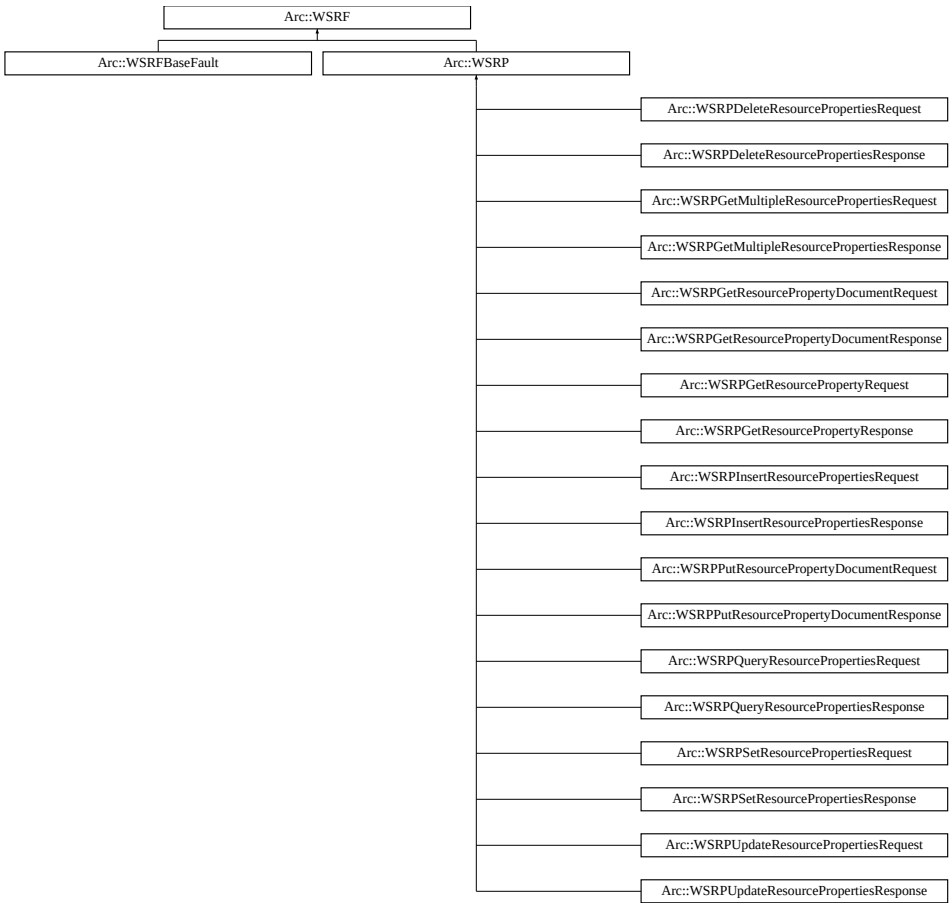
The documentation for this class was generated from the following file:

- WSA.h

5.46 Arc::WSRF Class Reference

`#include <WSRF.h>`

Inheritance diagram for Arc::WSRF::



Public Member Functions

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

Protected Member Functions

- void set_namespaces (void)

Protected Attributes

- SOAPMessage & **soap_**

- bool allocated_
- bool valid_

5.46.1 Detailed Description

Base class for every WSRF message to be derived from

5.46.2 Constructor & Destructor Documentation

5.46.2.1 Arc::WSRF::WSRF (SOAPMessage & *soap*, const std::string & *action* = "")

Constructor - creates object out of supplied SOAP tree.

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

Constructor - creates new WSRF object

5.46.3 Member Function Documentation

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

Returns true if instance is valid

5.46.3.2 void Arc::WSRF::set_namespaces (void) [protected]

set WS Resource namespaces and default prefixes in SOAP message

Reimplemented in Arc::WSRP.

5.46.3.3 virtual SOAPMessage& Arc::WSRF::SOAP (void) [inline, virtual]

Direct access to underlying SOAP element

5.46.4 Member Data Documentation

5.46.4.1 bool Arc::WSRF::allocated_ [protected]

Associated SOAP message - it's SOAP message after all

5.46.4.2 bool Arc::WSRF::valid_ [protected]

true if soap_ needs to be deleted in destructor

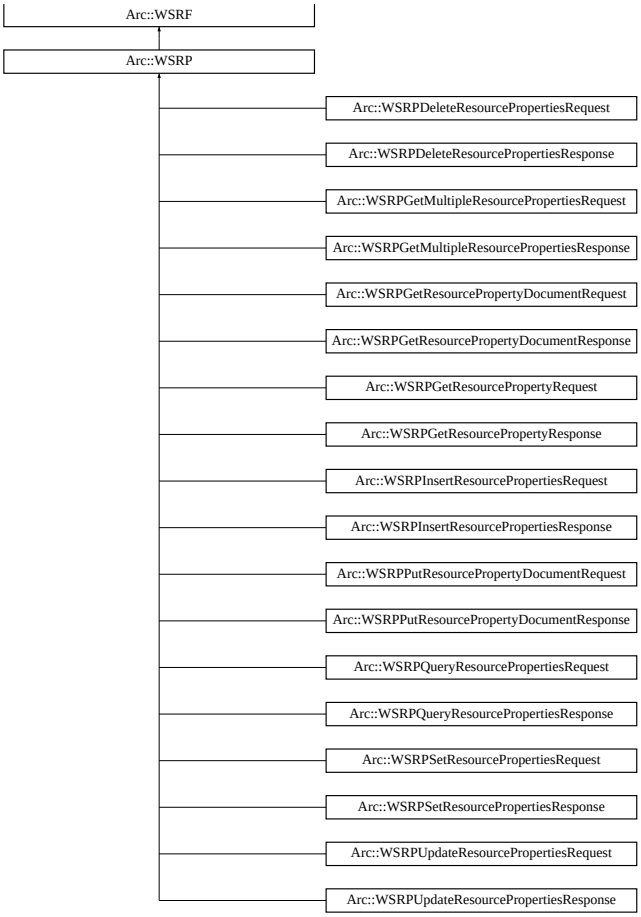
The documentation for this class was generated from the following file:

- WSRF.h

5.47 Arc::WSRP Class Reference

`#include <WSResourceProperties.h>`

Inheritance diagram for Arc::WSRP::



Public Member Functions

- WSRP (bool fault=false, const std::string &action="")
- WSRP (SOAPMessage &soap, const std::string &action="")

Protected Member Functions

- void set_namespaces (void)

5.47.1 Detailed Description

Base class for all WS-ResourceProperties structures. Inheriting classes implement specific WS-ResourceProperties messages and their properties/elements. Refer to WS-ResourceProperties specifications for things specific to every message.

5.47.2 Constructor & Destructor Documentation

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

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

5.47.2.2 Arc::WSRP::WSRP (SOAPMessage & *soap*, const std::string & *action* = "")

Constructor - creates object out of supplied SOAP tree. It does not check if 'soap' represents valid WS-ResourceProperties structure. Actual check for validity of structure has to be done by derived class.

5.47.3 Member Function Documentation

5.47.3.1 void Arc::WSRP::set_namespaces (void) [protected]

set WS-ResourceProperties namespaces and default prefixes in SOAP message

Reimplemented from Arc::WSRF.

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

- WSResourceProperties.h

5.48 Arc::WSRPFault Class Reference

`#include <WSResourceProperties.h>`

Inheritance diagram for Arc::WSRPFault::

Public Member Functions

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

5.48.1 Detailed Description

Base class for all WS-ResourceProperties faults

5.48.2 Constructor & Destructor Documentation

5.48.2.1 Arc::WSRPFault::WSRPFault (SOAPMessage & *soap*)

Constructor - creates object out of supplied SOAP tree.

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

Constructor - creates new WSRP fault

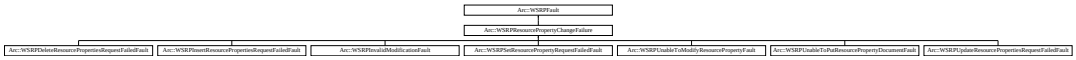
The documentation for this class was generated from the following file:

- WSResourceProperties.h

5.49 Arc::WSRPResourcePropertyChangeFailure Class Reference

`#include <WSResourceProperties.h>`

Inheritance diagram for Arc::WSRPResourcePropertyChangeFailure::



Public Member Functions

- WSRPResourcePropertyChangeFailure (SOAPMessage &soap)
- WSRPResourcePropertyChangeFailure (const std::string &type)
- XMLNode **CurrentProperties** (bool create=false)
- XMLNode **RequestedProperties** (bool create=false)

5.49.1 Detailed Description

Base class for WS-ResourceProperties faults which contain ResourcePropertyChangeFailure

5.49.2 Constructor & Destructor Documentation

5.49.2.1 Arc::WSRPResourcePropertyChangeFailure::WSRPResourcePropertyChangeFailure (SOAPMessage & *soap*) [inline]

Constructor - creates object out of supplied SOAP tree.

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

Constructor - creates new WSRP fault

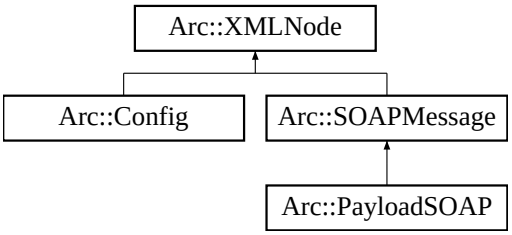
The documentation for this class was generated from the following file:

- WSResourceProperties.h

5.50 Arc::XMLNode Class Reference

`#include <XMLNode.h>`

Inheritance diagram for Arc::XMLNode::



Public Types

- typedef std::map< std::string, std::string > NS

Public Member Functions

- XMLNode (void)
- XMLNode (const XMLNode &node)
- XMLNode (const std::string &xml)
- XMLNode (const char *xml, int len=-1)
- XMLNode (const NS &ns)
- ~XMLNode (void)
- operator bool (void) const
- bool operator! (void) const
- XMLNode Child (int n=0) const
- XMLNode operator[ ] (const char *name) const
- XMLNode operator[ ] (const std::string &name) const
- XMLNode operator[ ] (int n) const
- int Size (void) const
- std::string Name (void) const
- void Name (std::string name)
- void GetXML (std::string &xml) const
- operator std::string (void) const
- XMLNode & operator= (const std::string &content)
- XMLNode & operator= (const char *content)
- XMLNode & operator= (const XMLNode &node)
- XMLNode Attribute (int n=0)
- XMLNode NewAttribute (const std::string &name)
- XMLNode NewAttribute (const char *name)
- XMLNode Attribute (const std::string &name)
- int AttributesSize (void)
- void Namespaces (const 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 Destroy (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)

5.50.1 Detailed Description

Wrapper for LibXML library Tree interface. This class wraps XML Node, Document and Property/Attribute structures. Each instance serves as pointer to actual LibXML element and provides convenient (for chosen purpose) methods for manipulating it. This class has no special ties to LibXML library and may be easily rewritten for any XML parser which provides interface similar to LibXML Tree. It implements only small subset of XML capabilities, which is probably enough for performing most of useful actions. This class also filters out (usually) useless textual nodes which are often used to make XML documents human-readable.

5.50.2 Member Typedef Documentation

5.50.2.1 typedef std::map<std::string,std::string> Arc::XMLNode::NS

convenience typedef representing mapping between namespace URIs and their prefixes

5.50.3 Constructor & Destructor Documentation

5.50.3.1 Arc::XMLNode::XMLNode (xmlNodePtr *node*) [inline, protected]

Private constructor for inherited classes Creates instance and links to existing LibXML structure. Acquired structure is not owned by class instance. If there is need to completely pass control of LibXML document to then instance's is_owner_ variable has to be set to true.

5.50.3.2 Arc::XMLNode::XMLNode (void) [inline]

Constructor of invalid node Created instance does not point to XML element. All methods are still allowed for such instance but produce no results.

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

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

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

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

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

Same as previous

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

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

5.50.3.7 Arc::XMLNode::~XMLNode (void) [inline]

Detructor Also destroys underlying XML document if owned by this instance

5.50.4 Member Function Documentation

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

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

5.50.4.2 XMLNode Arc::XMLNode::Attribute (int *n* = 0)

Returns XMLNode instance reresenting n-th attribute of node.

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

Returns number of attributes of node

5.50.4.4 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

5.50.4.5 void Arc::XMLNode::Destroy (void)

Destroys underlying XML element. XML element is unlinked from XML tree and destroyed. After this operation XMLNode instance becomes invalid

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

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

Reimplemented in Arc::SOAPMessage.

5.50.4.7 void Arc::XMLNode::Name (std::string *name*) [inline]

Assign new name to XML node

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

Returns name of XML node

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

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

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

Assign namespaces of XML document at point specified by this instance. If namespace already exists it gets new prefix. New namespaces are added. It is usefull to apply this method to XML being processed in order to refer to it's elements by known prefix.

Reimplemented in Arc::SOAPMessage.

5.50.4.11 XMLNode Arc::XMLNode::NewAttribute (const char * *name*)

Same as previous method

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

Creates new attribute with specified name.

5.50.4.13 XMLNode Arc::XMLNode::NewChild (const XMLNode & *node*, int *n* = -1, bool *global_order* = false)

Link a copy of supplied XML node as child. Returns instance refering to new child. XML element is a copy on supplied one but not ovned by returned instance

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

Same as previous method

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

Creates new child XML element at specified position with specified name. Default is to put it at end of list. If global order is true position applies to whole set of children, otherwise only to children of same name

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

Returns true if instance points to XML element - valid instance

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

Returns textual content of node excluding content of children nodes

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

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

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

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

5.50.4.20 XMLNode& Arc::XMLNode::operator= (const char * *content*) [inline]

Same as previous method

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

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

5.50.4.22]

XMLNode Arc::XMLNode::operator[ ] (int *n*) const

Returns XMLNode instance representing n-th node in sequence of siblings of same name. It's main purpose is to be used to retrieve element in array of children of same name like node["name"][5]

5.50.4.23]

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

Similar to previous method

5.50.4.24]

XMLNode Arc::XMLNode::operator[ ] (const char * *name*) const

Returns XMLNode instance representing first child element with specified name. Name may be "namespace_prefix:name" or simply "name". In last case namespace is ignored. If such node does not exist invalid XMLNode instance is returned

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

Returns number of children nodes

5.50.5 Friends And Related Function Documentation

5.50.5.1 bool MatchXMLName (const XMLNode & *node*, const char * *name*) [friend]

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

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

Returns true if underlying XML elements have same names

5.50.6 Member Data Documentation

5.50.6.1 bool Arc::XMLNode::is_owner_ [protected]

If true node is owned by this instance - hence released in destructor. Normally that may be true only for top level node of XML document.

5.50.6.2 bool Arc::XMLNode::is_temporary_ [protected]

This variable is for future

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

- XMLNode.h

Chapter 6

KnowARC Page Documentation

6.1 Binding "echo"

6.1.1 Operations of Binding "echo"

- __ns1__echo

6.1.2 Endpoints of Binding "echo"

- http://localhost:80

Index