OIO Service Oriented Infrastructure

OIOSI RASP Library for .Net RC1 P2

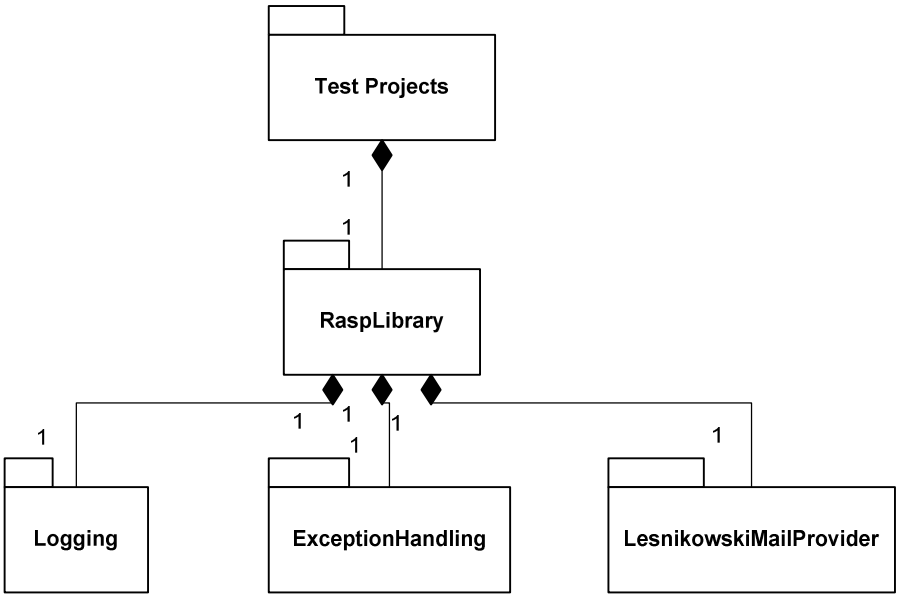
Overview

Contents

1 Introduction

The purpose of this document is to describe the main components of the RASP software library for .Net RC1 release. Only the most important classes are presented, to give an overview of the various elements.

2 Overview

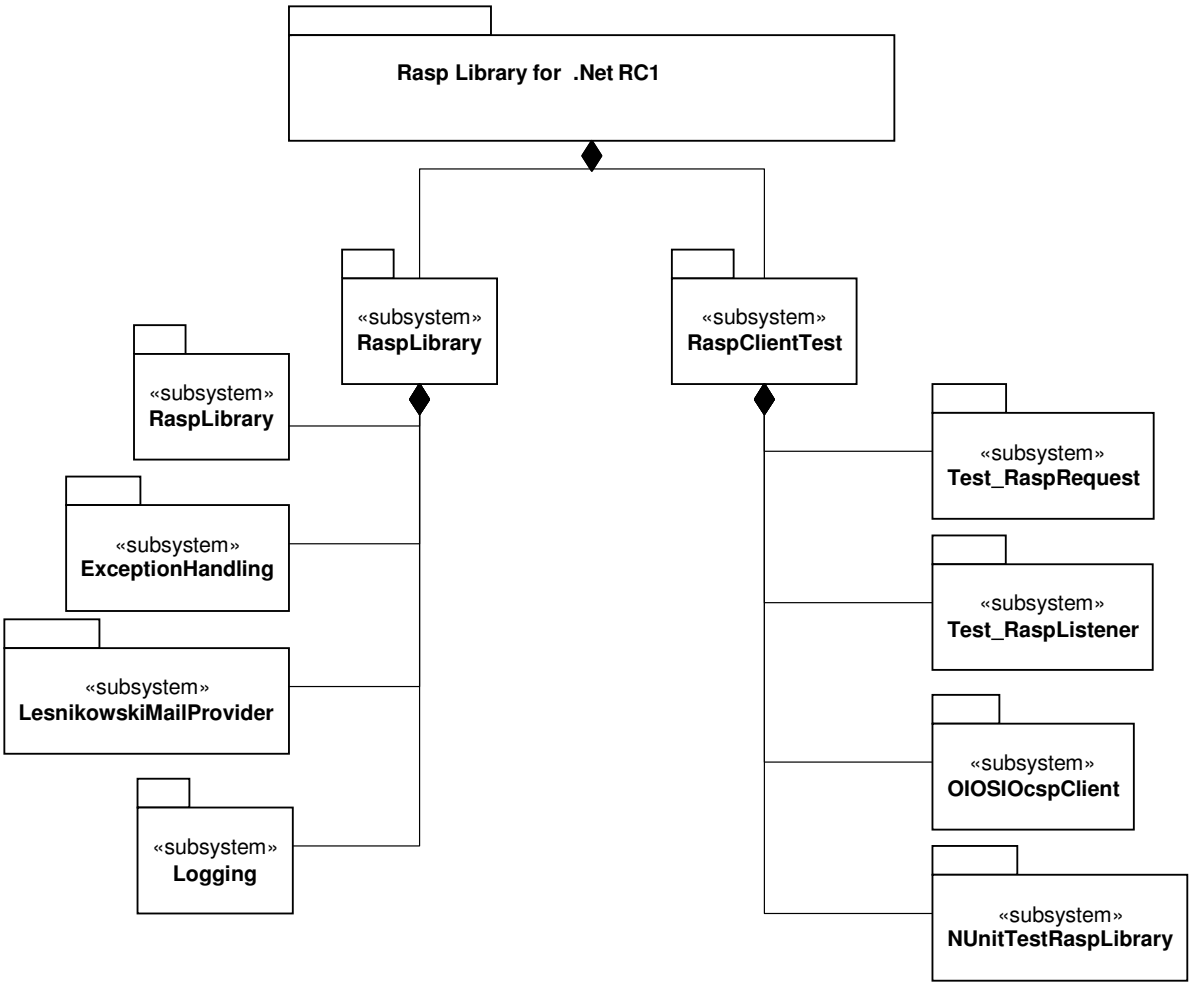


The RASP Library ships with a logging component, an exception component (which all libraries depend on) and LesnikowskiMailProvider, which provides mail functionality through the Lesnikowski mail library. The Lesnikowski mail provider is pluggable and can be exchanged for other mail libraries.

The main component, **the RASP software library** contains the following elements:

- Communication (client)
- Communication extended (client)
- Certificate handling & LDAP
- OCSP
- UDDI
- Validation / schema
- Validation / Schematron
- Email
- WCF extensions / email transport
- WCF extensions / interceptor
- Application level resending

3 RC1 Release Pack



This release pack includes two visual studio solutions, one for RASP Library functionality, and one for tests. The library functionality will be discussed further in later sections of this document.

Test_RaspRequest demonstrates the sending functionality of the RASP Library, and how to use it in different ways.

Test_RaspListener provides a simple command line mail service.

OIOSIOcspClient demonstrates OCSP lookup functionality.

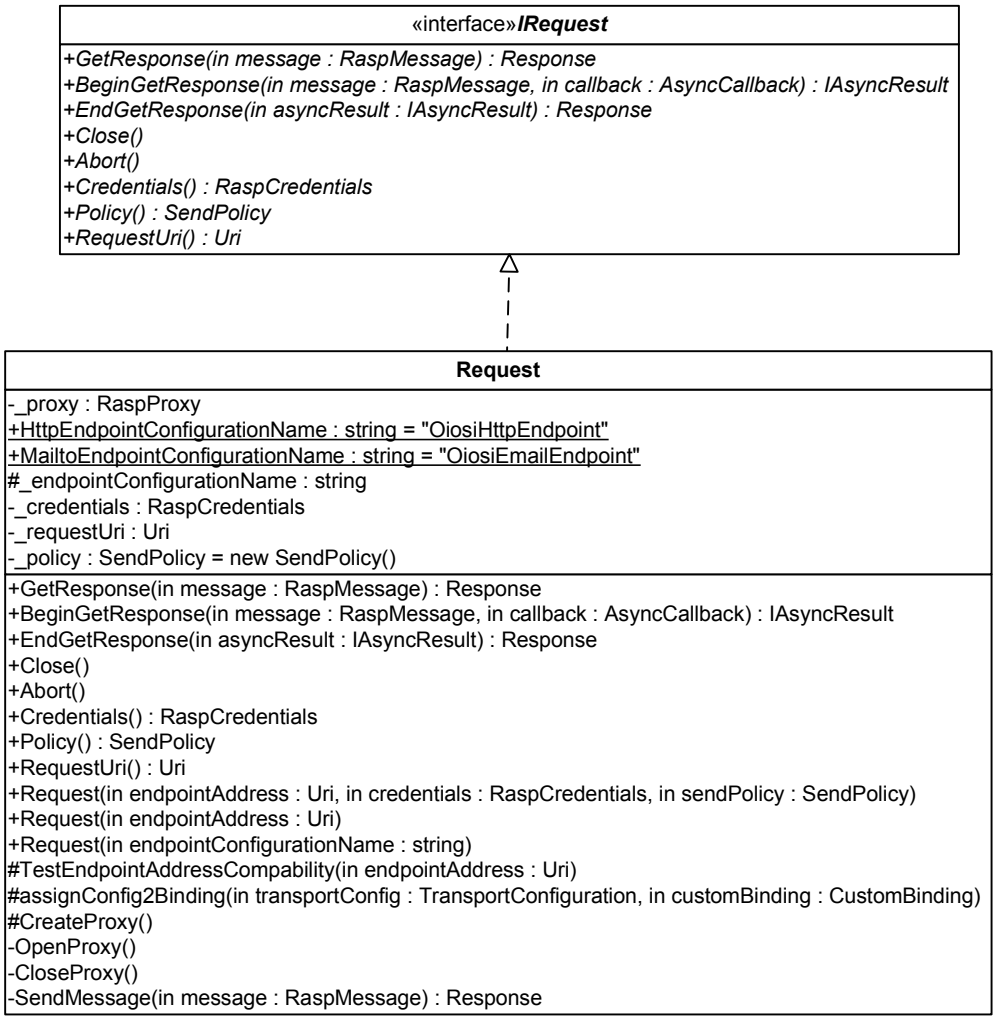
NUnitTestRaspLibrary provides NUnit tests for all major parts of the RASP Library, enabling simple error testing. A run through of this project in NUnit might show whether the RASP Library has been set up correctly.

4 Components

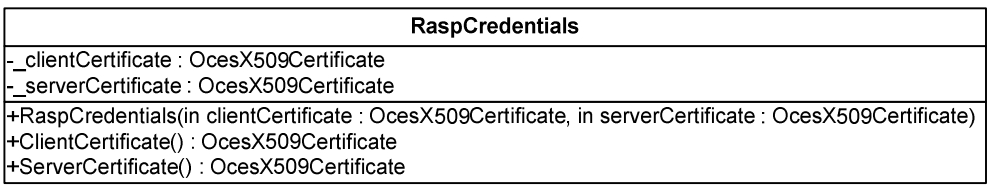
This section describes each of the high level interfaces to the RASP library.

4.1 Communication, client side

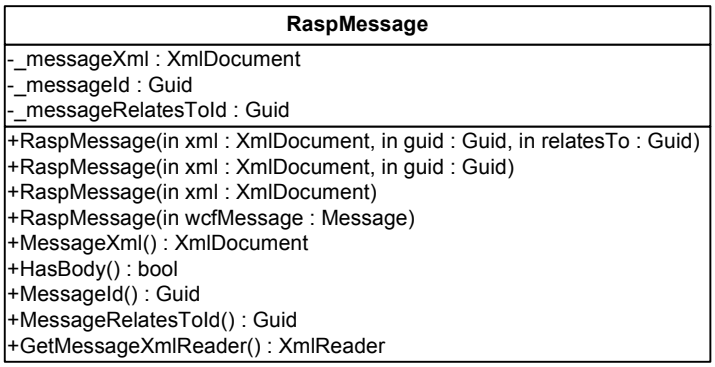
The main client side class is the Request, which enables sending functionality. Request can either be initialized with an URI and some credentials or with the reference to a client endpoint in the configuration file.



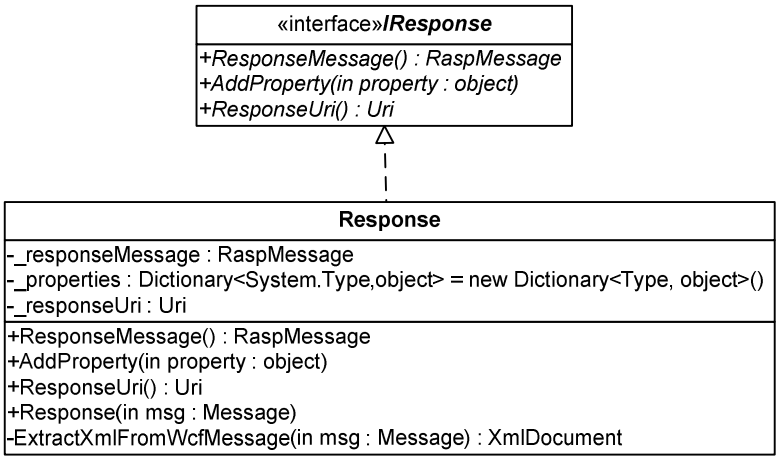
If initialized with RaspCredentials, two certificates should be given. One for client side encryption and one for validating server side responses



When sending Request takes a RaspMessage, which basically is an XML document with metadata attached to it.



In return a Response is given, which again is an XML document, the one returned from the called service, coupled with metadata.

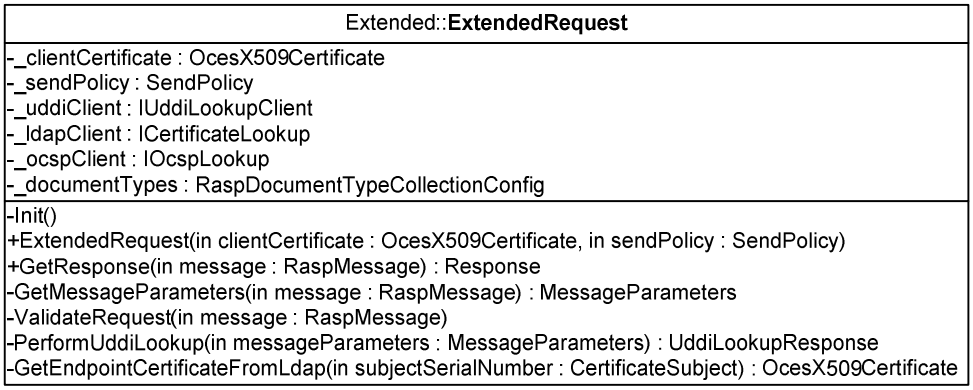


4.2 Extended communication, client side

The RequestExtended request demonstrates the following in addition to the plain Request:

- Gets identifiers from xml documents (e.g. EAN or OVT numbers) from documents using configurable xpath expressions
- Performs a UDDI lookup using these parameters
- Retrieves an endpoint certificate from LDAP based on information returned from UDDI
- Checks certificate revocation status against OCSP.

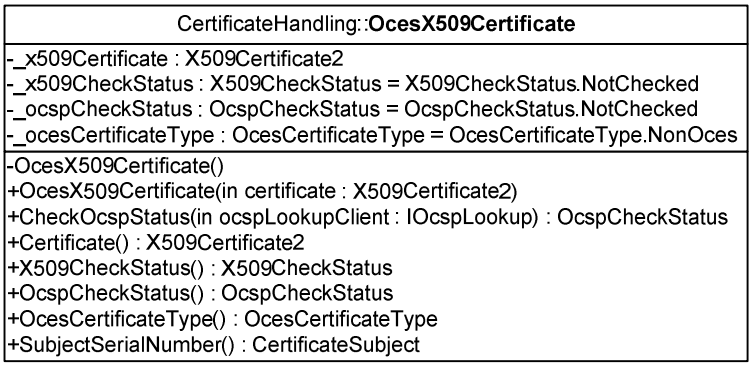
The main method is the GetResponse(RaspMessage). When calling this, the message is examined and configuration retrieved in order to make the relevant OCSP, LDAP and UDDI lookups.



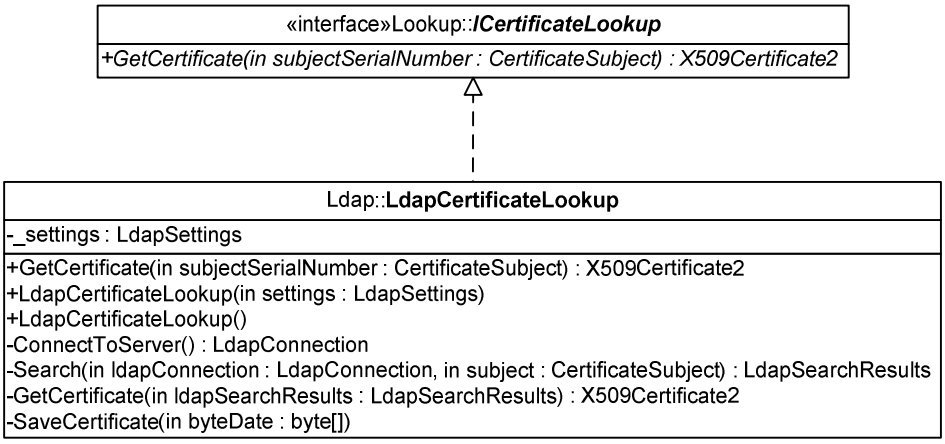
See the tutorial document for more information on how to configure and use this class.

4.3 Certificate handling & LDAP utilities

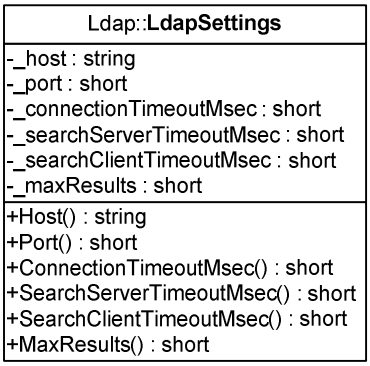
These components are used to validate the client certificates. These are represented by the OcesX509Certificate. The class is also responsible for checking the certificates expiration date, activation date, chain checks etc.



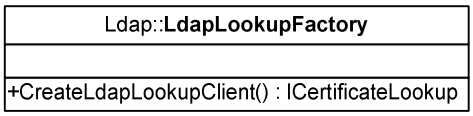
The Ldap lookup uses the ldap protocol to query for a given certificate. It is also responsible for opening and closing connections to the ldap server.



The LdapCertificateLookup constructor takes an ldap settings object as a parameter. The setting object is used every time a lookup is performed in the GetCertificate function.



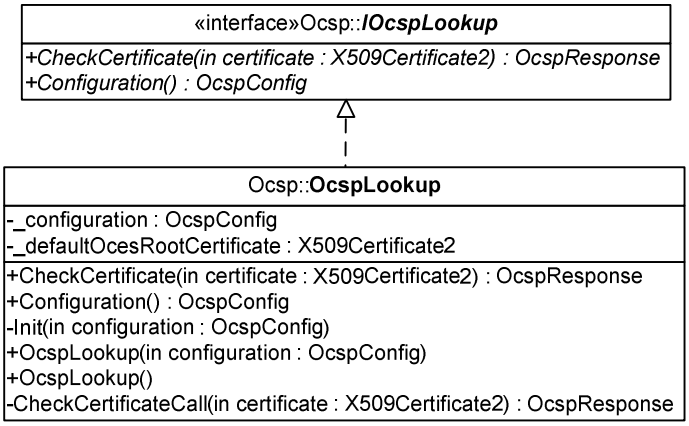
The LdapLookupFactory instantiates classes with the ICertificateLookup interface based on the configuration.



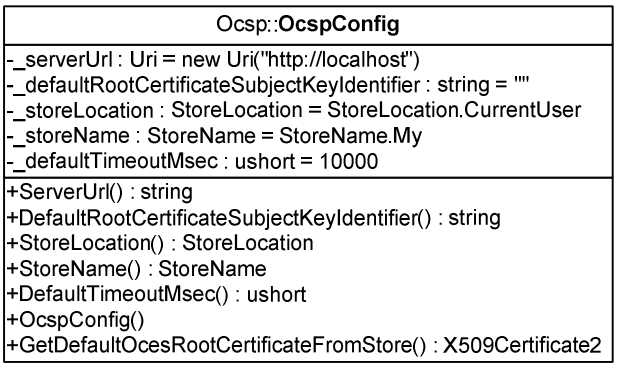
The factory will instantiate either *LdapCertificateLookup* or *LdapCertificateLookupTest*. You can change this behavior in the *RaspConfiguration.xml* file, in the section **<LdapLookupFactoryConfig>** element by setting the namespace and assembly name of the ICertificateLookup implementation to instantiate. Developers may add their own implementation libraries and instantiate these just by changing the configuration.

4.4 OCSP

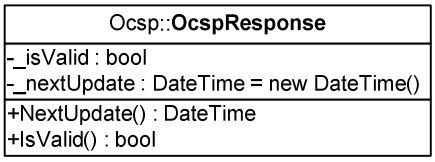
OCSP lookup component is used to check certificate revocation status against an OCSP server.



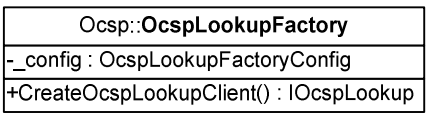
The OcspConfig is used to dynamically set the URL to the ocsp server, timeout, various default settings etc.



The response from the lookup contains an isValid boolean, which indicates if the certificate has been revoked, either temporarily or permanently. The response also contains a nextUpdate value, which is used to store the time when newer information will be available, but this has not been implemented for this preview.



OcspLookupFactory creates an instance of a class with the IOscspLookup interface.

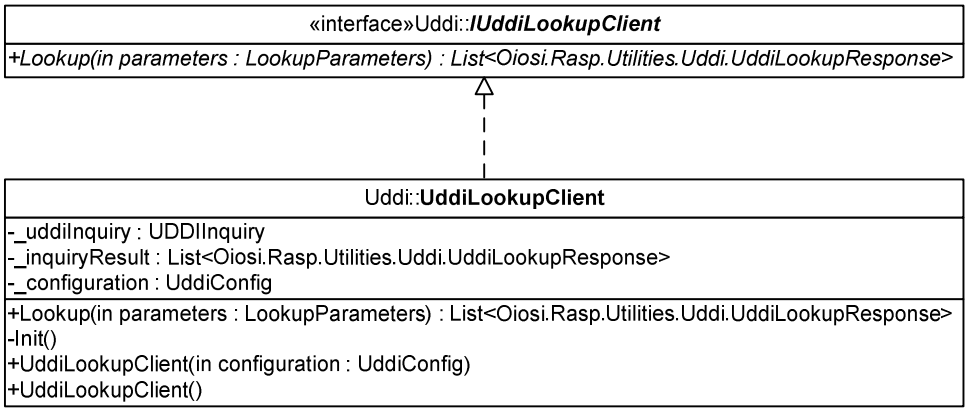


The factory will instantiate either *OcspLookup* or *OcspLookupTest*. You can change this behavior in the *RaspConfiguration.xml* file, in the section **<OcspLookupFactoryConfig>**

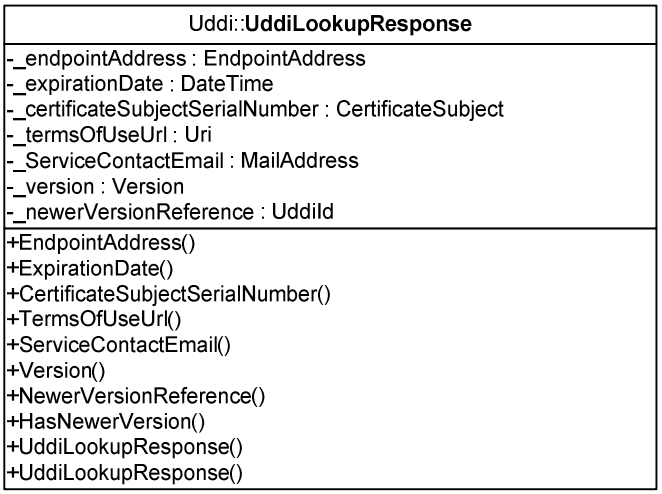
element by setting the namespace and assembly name of the IOcspLookup implementation to instantiate. Developers may add their own implementation libraries and instantiate these just by changing the configuration.

4.5 UDDI

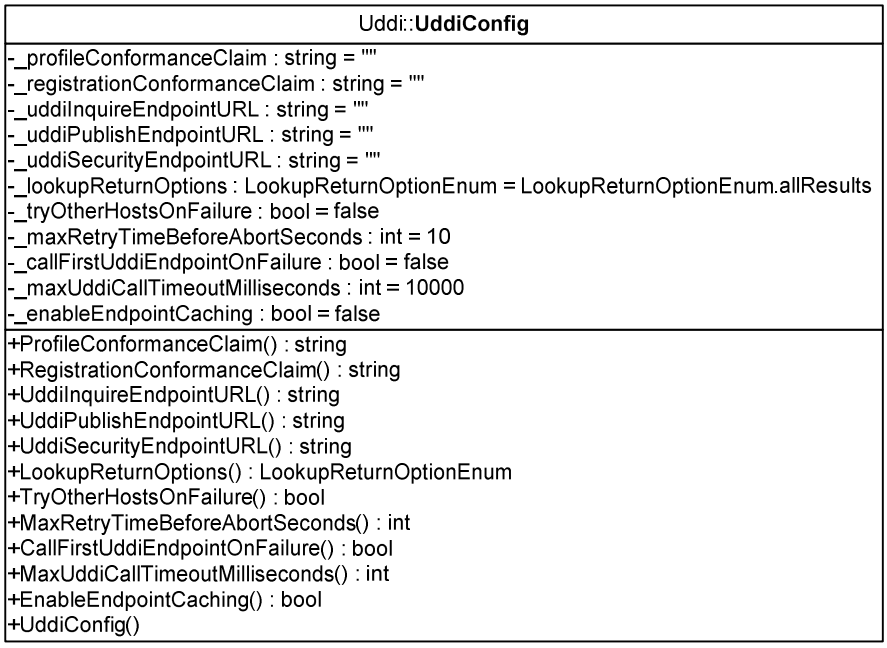
The UDDI is used in the ARS (Address Resolving Service), which is used to resolve business level endpoints to service endpoints. Below you can see the main UDDI classes for lookup.



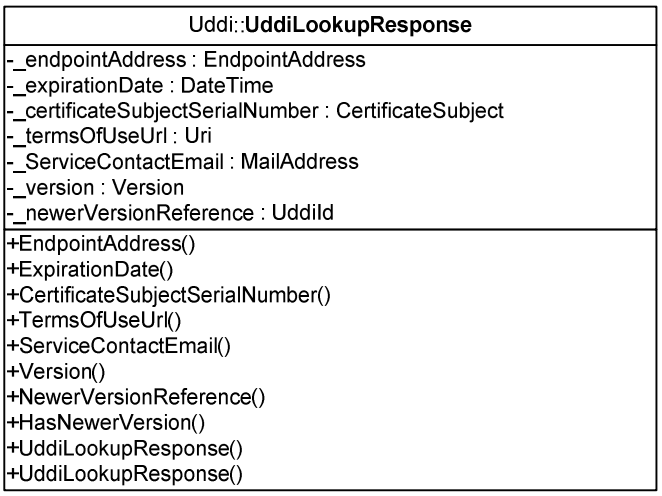
The lookup is based on a endpoint key, like EAN number, and other lookup options like address type etc. The component can also handle automatic caching of endpoints of resolving attempts and supports caching policies based on the call result feedback.



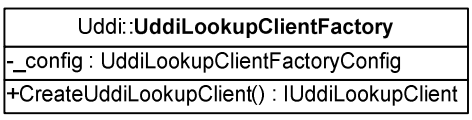
The client takes a configuration object, that sets attributes like UDDI endpoint url, return options, timeouts etc.



The response from the lookup contains the endpoint address and other parameters.



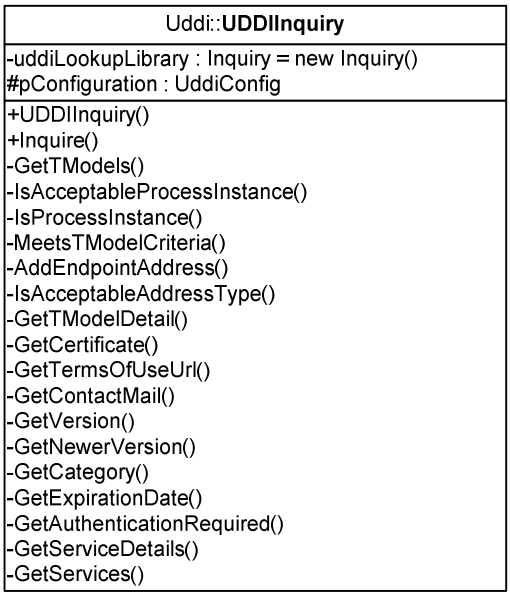
One of the UDDI support classes is the UddiLookupClientFactory, which creates an IUddiLookup implementation, as set in config.



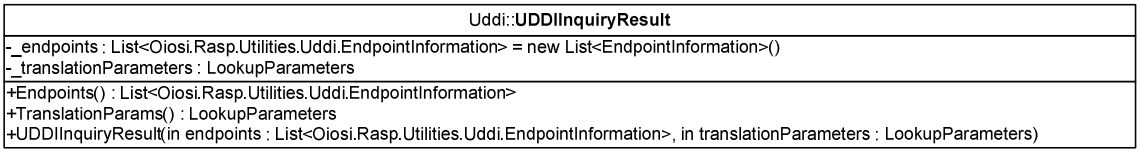
The factory will instantiate either *UddiLookupClient* or *UddiLookupClientTest*. You can change this behavior in the *RaspConfiguration.xml* file, in the section **<UddiLookupFactoryConfig>** element by setting the namespace and assembly name of the IUddiLookupClient

implementation to instantiate. Developers may add their own implementation libraries and instantiate these just by changing the configuration.

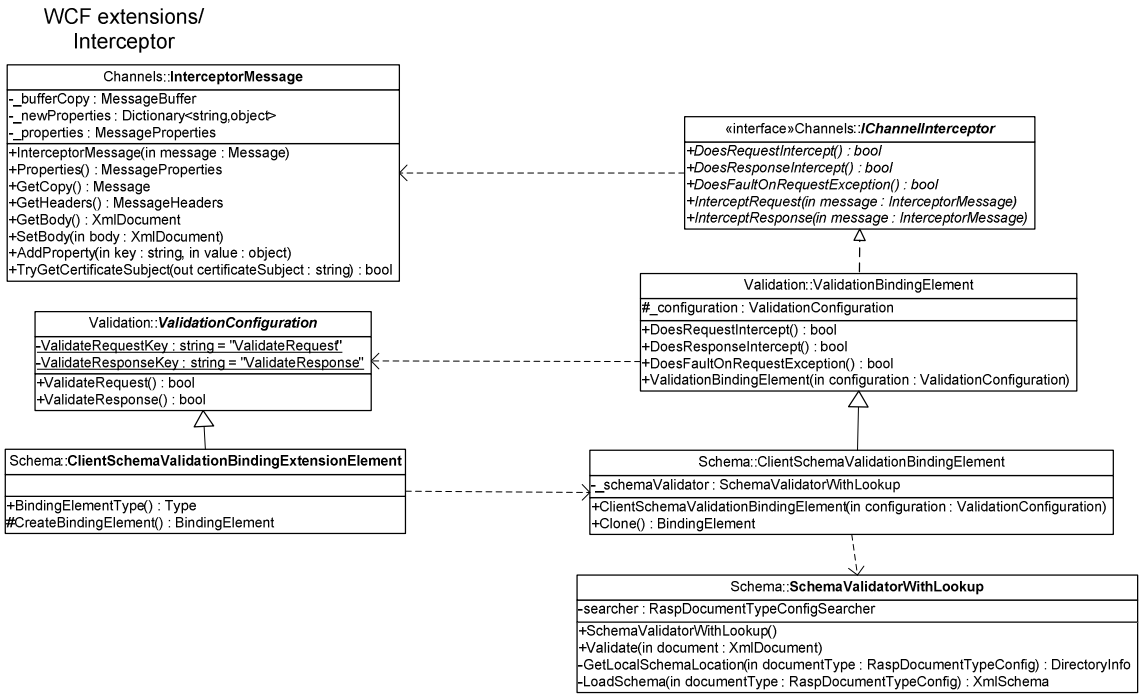
The UDDI inquiry represents an inquiry to the UDDI inquiry API. The inquiry takes parameters like endpoint key, lookup policies, the UDDI endpoint etc.



The result from an inquiry is represented in the UDDIInquiryResult class. It consists of the found endpoints, which includes metadata on the endpoints, such as UDDI tModel identifiers.



4.6 Validation, schema, client



InterceptorMessage is an extension on the Message from the service model. It has extended functionality and is used by the interceptor when calling the IChannelInterceptor.

IChannelInterceptor is the interface any interceptor must implement. It is called when the request or response is intercepted.

ValidationConfiguration implements basic validation configuration.

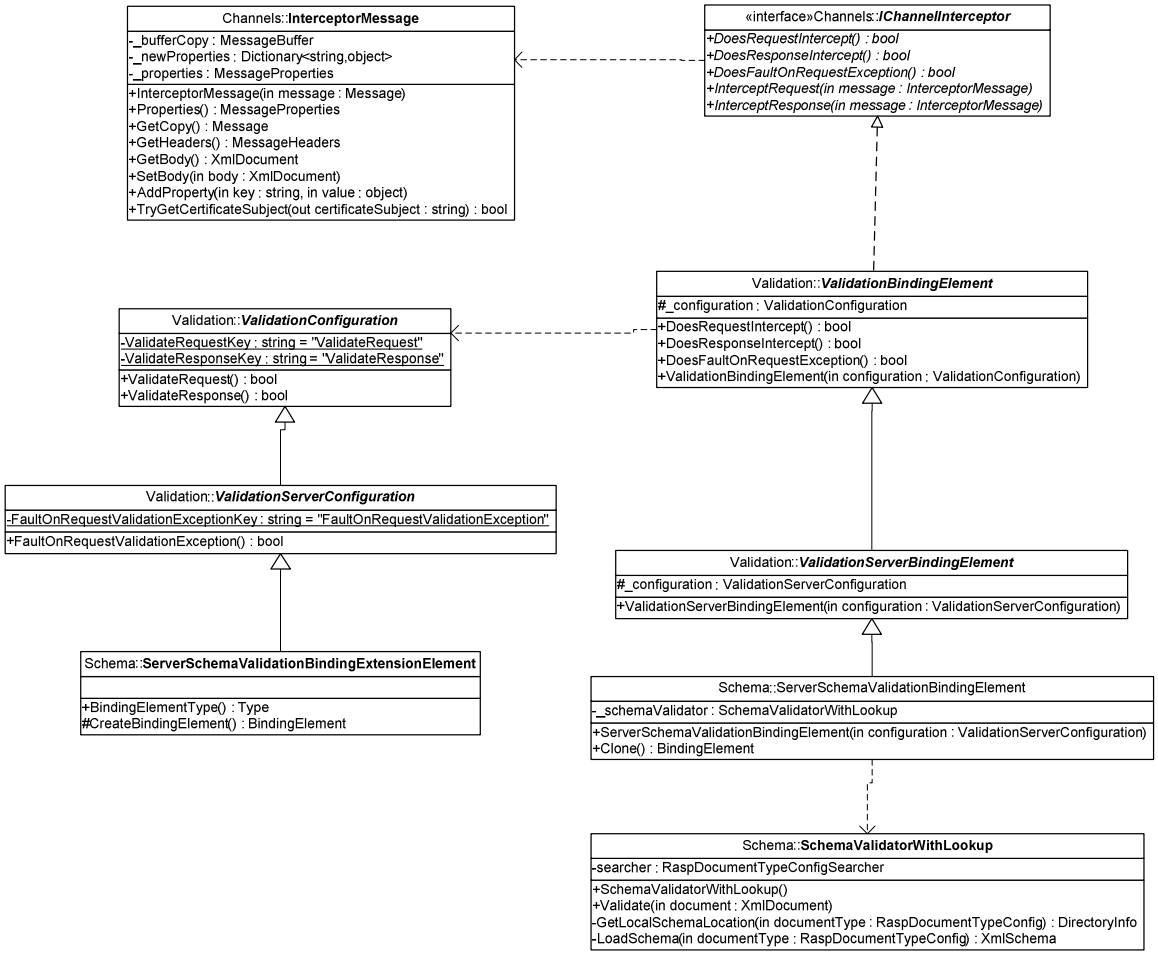
ClientSchemaValidationBindingExtensionElement inherits the ValidationConfiguration and creates the ClientSchemaValidationBindingElement when wcf is building the stack.

ValidationBindingElement partly implements the IChannelInterceptor interface. It does not implement the interception methods, but it implements whether the inceptor does intercept request or response. This is determined by the state of ValidationConfiguration.

ClientSchemaValidationBindingElement inherits the ValidationBindingElement and implements the interception methods. Here the actual interception takes place and it calls the SchemaValidationWithLookup to validate whether the intercepted message body is schema valid.

SchemaValidationWithLookup validates an xml document by first finding the corresponding type of xml and then validate the document with the fitting schema.

4.7 Validation, Schema, Server



InterceptorMessage is an extension on the Message from the service model. It has extended functionality and is used by the interceptor when calling the IChannelInterceptor.

IChannelInterceptor is the interface any interceptor must implement. It is called when the request or response is intercepted.

ValidationConfiguration implements basic validation configuration.

ValidationServerConfiguration implement the server specific validation configuration.

ServerSchemaValidationBindingExtensionElement inherits the ValidationServerConfiguration and creates the ServerSchemaValidationBindingElement when wcf is building the stack.

ValidationBindingElement partly implements the IChannelInterceptor interface. It does not implement the interception methods, but it implements whether the inceptor does intercept request or response. This is determined by the state of ValidationConfiguration.

ServerValidationBindingElement inherits the ValidationBindingElement and overrides whether the inceptor should fault or not. This is determined by the ValidationServerConfiguration.

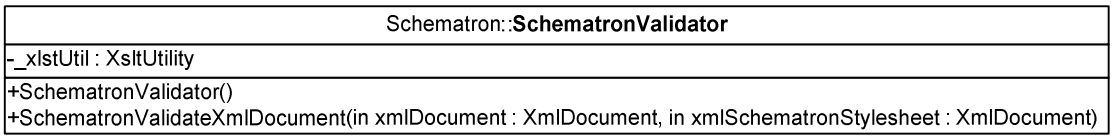
ServerSchemaValidationBindingElement inherits the ServerValidationBindingElement and implements the interception methods. Here the actual interception takes place and it calls the SchemaValidationWithLookup to validate whether the intercepted message body is schema valid.

SchemaValidationWithLookup validates an xml document by first finding the corresponding type of xml and then validate the document with the fitting schema.

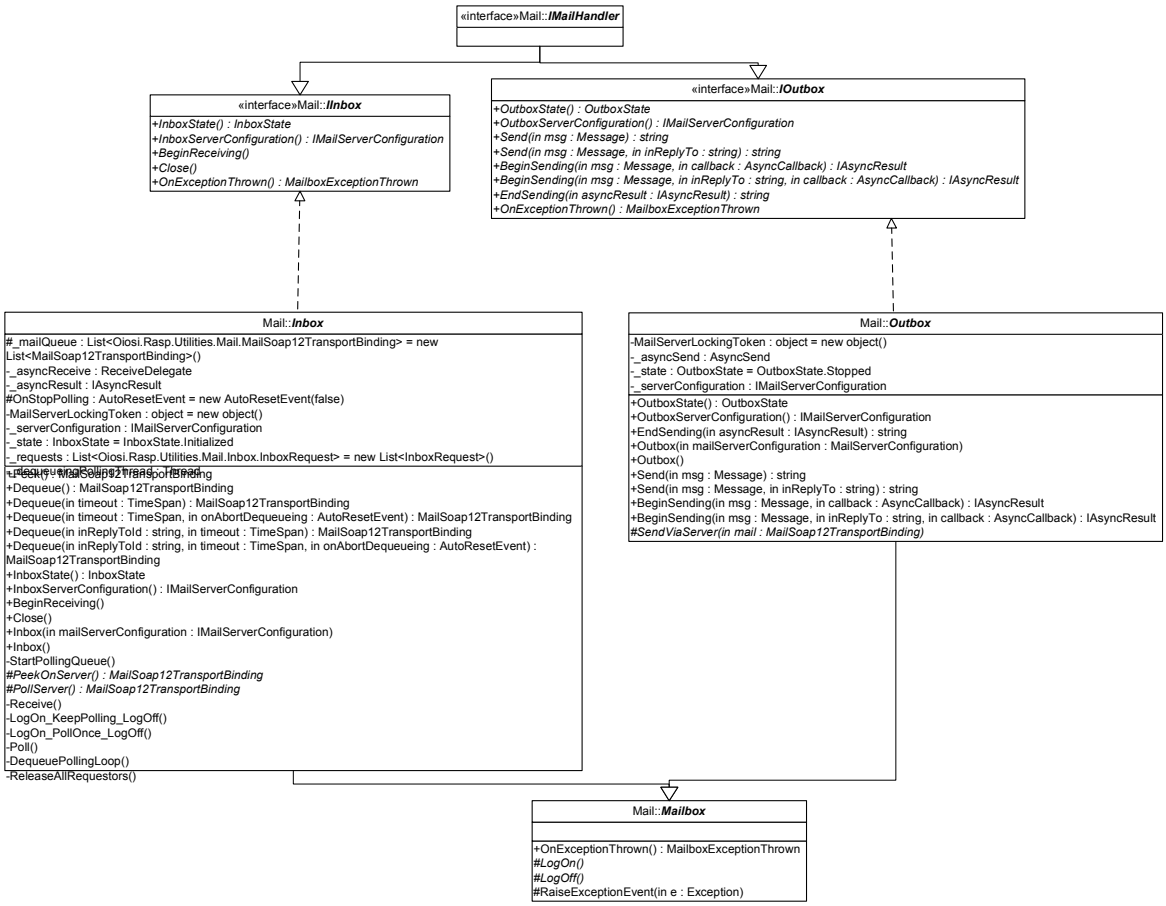
The server side interception functions partly as the client side interception. It has the same configuration and same functionality. The only difference is whether it should send a soap fault or propagate the information when the validation fails.

4.8 Validation, schematron

The schematron validation interceptor works as the schema validation interceptor, except that it uses a schematron validation instead of a schema validation.



4.9 Email handling

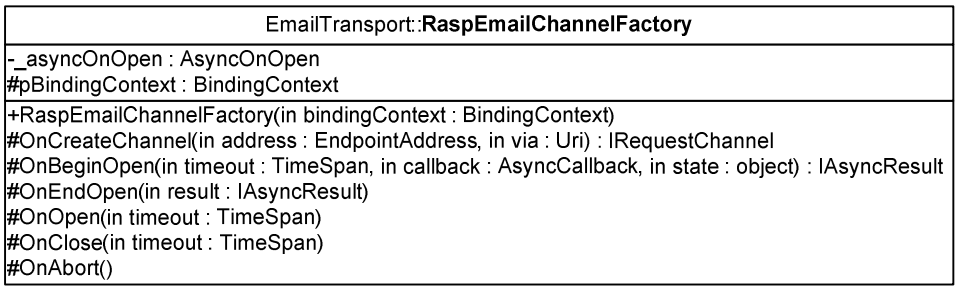


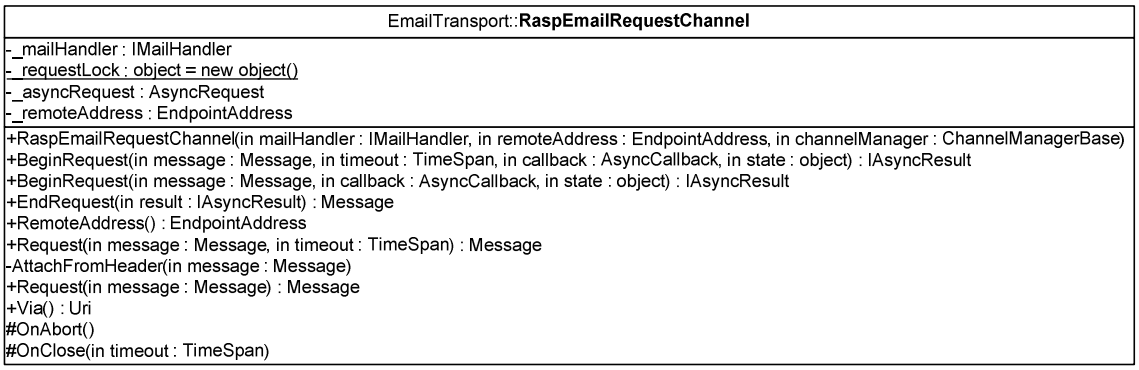
The email handling component contains two mailboxes, one inbox – used for receiving mails, and one outbox – for sending mails. The implementation of the methods doing the actual calls to the mail server is pluggable, and even though the Rasp library ships with an implementation using the Lesnikowski mail library, one could choose to use their own implementation.

While running the inbox will eat every mail there is on the mail server, and keeps them in memory, for de-queuing.

4.10 WCF extensions, Email transport - client

The email transport enables Windows Communication Foundation (WCF) to communicate via Smtp/Pop3. As with standard WCF channel pattern, a channel factory provides channels, in this case Request channels. The email request channel will be at the bottom of the stack, and handle only the actual transport, letting WCF handle reliable messaging and security.





4.11 WCF extensions, Email transport - server

The server side pattern reminds a lot of the client side, only with a Listener acting as factory, and producing ReplyChannels.

EmailTransport::**RaspEmailChannelListener**
#pContext : BindingContext #pMailHandler : RaspMailHandler -_onEndDequeueing : AutoResetEvent = new AutoResetEvent(false) -_asyncOnOpen : AsyncOnOpen -_asyncOnClose : AsyncOnClose -_asyncOnAcceptChannel : AsyncOnAcceptChannel
+Uri() : Uri +RaspEmailChannelListener(in context : BindingContext) #OnAcceptChannel(in timeout : TimeSpan) : IReplyChannel #OnBeginAcceptChannel(in timeout : TimeSpan, in callback : AsyncCallback, in state : object) : IAsyncResult #OnEndAcceptChannel(in result : IAsyncResult) : IReplyChannel #OnWaitForChannel(in timeout : TimeSpan) : bool #OnBeginWaitForChannel(in timeout : TimeSpan, in callback : AsyncCallback, in state : object) : IAsyncResult #OnEndWaitForChannel(in result : IAsyncResult) : bool #OnAbort() #OnBeginOpen(in timeout : TimeSpan, in callback : AsyncCallback, in state : object) : IAsyncResult #OnEndOpen(in result : IAsyncResult) #OnOpen(in timeout : TimeSpan) #OnBeginClose(in timeout : TimeSpan, in callback : AsyncCallback, in state : object) : IAsyncResult #OnEndClose(in result : IAsyncResult) #OnClose(in timeout : TimeSpan) -CallbackMailExceptionThrown(in e : Exception, in caller : object)

EmailTransport::**RaspEmailReplyChannel**
-_mailHandler : IMailHandler -_asyncTryReceiveRequest : AsyncTryReceiveRequest -_asyncTryReceiveRequestOutContext : RequestContext
+RaspEmailReplyChannel(in channelManager : ChannelManagerBase, in mailQueue : IMailHandler) +BeginReceiveRequest(in timeout : TimeSpan, in callback : AsyncCallback, in state : object) : IAsyncResult +BeginReceiveRequest(in callback : AsyncCallback, in state : object) : IAsyncResult +EndReceiveRequest(in result : IAsyncResult) : RequestContext +ReceiveRequest(in timeout : TimeSpan) : RequestContext +ReceiveRequest() : RequestContext +BeginTryReceiveRequest(in timeout : TimeSpan, in callback : AsyncCallback, in state : object) : IAsyncResult +EndTryReceiveRequest(in result : IAsyncResult, out context : RequestContext) : bool +TryReceiveRequest(in timeout : TimeSpan, out context : RequestContext) : bool +BeginWaitForRequest(in timeout : TimeSpan, in callback : AsyncCallback, in state : object) : IAsyncResult +EndWaitForRequest(in result : IAsyncResult) : bool +WaitForRequest(in timeout : TimeSpan) : bool +LocalAddress() : EndpointAddress

The similarities end there though, since the incoming message caught by the Reply channel is propagated up the stack (through the security and reliable messaging layers to the application layer) in a RequestContext. This request context makes it possible for the other stack layers to actually reply to the incoming request.

EmailTransport::**RaspEmailRequestContext**
#_mailHandler : IMailHandler -_asyncReply : AsyncReply -_requestMessage : MailSoap12TransportBinding
+RequestMessage() : Message +RaspEmailRequestContext(in msg : MailSoap12TransportBinding, in mailHandler : IMailHandler) +Reply(in message : Message, in timeout : TimeSpan) +Reply(in message : Message) +BeginReply(in message : Message, in timeout : TimeSpan, in callback : AsyncCallback, in state : object) : IAsyncResult +BeginReply(in message : Message, in callback : AsyncCallback, in state : object) : IAsyncResult +EndReply(in result : IAsyncResult) -AttachFromAndToHeaders(in message : Message) +Close(in timeout : TimeSpan) +Close() +Abort()