Guideline

Acronym: PEPPOL
Grant Agreement number: 224974
Title: Pan-European Public Procurement Online

PEPPOL Transport Infrastructure .Net Core Library Developer Guide

Version: 1.0

Authors:
Nisha Appanah (NITA/Software Teams)
Jens Jakob Andersen (NITA)
Klaus Vilstrup Pedersen (DIFI)

Project co-funded by the European Commission within the ICT Policy Support Programme

<table>
<thead>
<tr>
<th>Dissemination Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>P</td>
</tr>
<tr>
<td>C</td>
</tr>
</tbody>
</table>

This work is licensed under the Creative Commons Attribution 4.0 International License (CC BY 4.0).
Revision History

<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
<th>Author</th>
<th>Organisation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0</td>
<td>20100215</td>
<td>Nisha Appanah</td>
<td>NITA/Software Teams</td>
<td>First version (pending EC approval)</td>
</tr>
</tbody>
</table>

Statement of originality

This deliverable contains original unpublished work except where clearly indicated otherwise. Acknowledgement of previously published material and of the work of others has been made through appropriate citation, quotation or both.

Statement of copyright

This deliverable is released under the terms of the Creative Commons Licence accessed through the following link: http://creativecommons.org/licenses/by/3.0/.

In short, it is free to

**Share** — to copy, distribute and transmit the work

**Remix** — to adapt the work

Under the following conditions

**Attribution** — You must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work).
Contributors

Organisations

DIFI (Direktoratet for forvaltning og IKT)\(^1\), Norway, [www.difi.no](http://www.difi.no)
NITA (IT- og Telestyrelsen)\(^2\), Denmark, [www.itst.dk](http://www.itst.dk)

Persons

Gert Sylvest, NITA/Avanada
Heveen Kejiou, NITA/Software Teams
Jens Jakob Andersen, NITA
Klaus Vilstrup Pedersen, DIFI
Nisha Appanah, NITA/Software Teams (editor)

---

\(^1\) English: Agency for Public Management and eGovernment

\(^2\) English: National IT- and Telecom Agency
# Table of Content

1 Introduction .................................................................................................................. 5  
1.1 Objective .................................................................................................................. 5  
1.2 Scope ........................................................................................................................ 5  
2 PEPPOL .Net Core Library Developer Guide .............................................................. 6 
  2.1 Prerequisites ........................................................................................................... 6 
  2.2 Tracing ..................................................................................................................... 6 
3 Lesson: Creating a LIME Client .................................................................................. 8  
4 Lesson: Creating a LIME Access Point ...................................................................... 12  
  4.1 Solution overview .................................................................................................... 13  
  4.2 The IOLayerLibrary project .................................................................................... 13  
  4.3 The LIMELibrary project ......................................................................................... 13  
  4.4 The LIMEService project ....................................................................................... 14  
  4.5 The LIMESTARTTest project and LIMEtestproject .................................................. 14  
  4.6 The ServiceMetadataLibrary .................................................................................. 14  
5 Lesson: Creating a START Access Point .................................................................... 17  
  5.1 Introduction ............................................................................................................. 17  
  5.2 Solution overview ................................................................................................... 17  
  5.3 The STARTLibrary project ....................................................................................... 17  
  5.4 The STARTAccessPoint project .............................................................................. 18  
  5.5 The SampleSTARTClient project .......................................................................... 19
1 Introduction

1.1 Objective
This document describes how to use and customize source code and libraries in the PEPPOL .Net Core Library.

The libraries are used for implementing the BUSDOX Lightweight Message Exchange Profile (LIME) and the Secure Trusted Asynchronous Reliable Transport (START).

You can find information about the profiles in the document “PEPPOL Implementation and Specs.doc”.

1.2 Scope
This guideline relates to the Technical Transport Layer i.e. BusDox specifications. The BusDox specifications can be used in many interoperability settings. In the PEPPOL context, it provides transport for procurement documents as specified in the PEPPOL Profiles.
2 PEPPOL .Net Core Library Developer Guide

2.1 Prerequisites
General prerequisites:
Installation requirements for developers:
- Microsoft .NET Framework 3.0/3.5
- Microsoft Visual Studio 2008 or Microsoft Visual Studio 2005 (with WCF extension)
- Internet Information Services (IIS) - if you plan to run the server-side samples
- If in addition you want to have a look at the START prototype SAML bindings, you need to install Windows Identity Foundation.
- System.IdentityModel.dll and System.ServiceModel.dll in min. version 3.0.4506.4446

2.2 Tracing
SOAP tracing can be enabled by adding the information below in the app.config file. There will be two types of traces.
1. Message Logging will trace the SOAP request and response, that is logs all messages that flow through the system. This will be stored in the app_messages.svclog.
2. Trace Listener will log all operation of the LIME client. The traces will be stored in app_tracelog.svclog.

Under the configuration tag, copy and paste the system.diagnostics information in your app.config file.

```xml
<system.diagnostics>
  <sources>
    <source name="System.ServiceModel.MessageLogging" switchValue="Warning, ActivityTracing">
      <listeners>
        <add type="System.Diagnostics.DefaultTraceListener" name="Default">
          <filter type="" />
        </add>
        <add name="ServiceModelMessageLoggingListener">
          <filter type="" />
        </add>
      </listeners>
    </source>
    <source name="System.ServiceModel" switchValue="Warning, ActivityTracing">
      propagateActivity="true">
      <listeners>
        <add type="System.Diagnostics.DefaultTraceListener" name="Default">
          <filter type="" />
        </add>
        <add name="ServiceModelTraceListener">
          <filter type="" />
        </add>
      </listeners>
    </source>
  </sources>
  <sharedListeners>
</system.diagnostics>
```
Under the **system.serviceModel** tag, copy and paste the diagnostics information in your **app.config** file.

```xml
<diagnostics>
    <messageLogging logEntireMessage="true" logMalformedMessages="true"
        logMessagesAtServiceLevel="true" logMessagesAtTransportLevel="true" />
</diagnostics>
```
3 Lesson: Creating a LIME Client

A LIME Client is capable of sending and receiving XML documents to and from LIME Access Points.

A LIME Client can be created by using the library “LIMElibrary.dll”. This dll contains classes for communicating with an Access Point that supports the PEPPOL LIME Profile.

The LIMElibrary has the following main features

- Contains classes for a document inbox and outbox
- Contains classes for business, document and process identifiers
- Contains a web service client for an LIME Access Point

LIME Access Points identifies senders of documents by a Business Identifier and Inboxes by a Channel Identifier (see LIME Profile Spec for more details).

It is very easy to create an LIME Client, and it can be done in few steps:

1) Create a method for sending messages
2) Create a method for listing messages in an Inbox
3) Create a method for retrieving a specific message in an Inbox
4) Create a method for deleting messages

The interfaces that must be used to create a LIME Client can be found in the package “eu.peppol.lime.api.interfaces”. A factory for instantiating implementations of the interfaces can be found in “eu.peppol.lime.api.impl.Factory”. Inboxes can be compared to a standard POP3-inbox, where you must poll the Inbox to retrieve messages. To get a specific message one must first retrieve a list of Message Identifiers in the Inbox and then fetch a single message afterwards.

LIME Clients are also responsible for deleting messages when they are read or no longer needed in the Inbox.

Below is shown an example of how to retrieve a list of messages in the Inbox.

```csharp
Factory factory = new Factory();
IEndpointReference endpointReference = factory.CreateEndpointReference();
endpointReference.ChannelIdentifier = "0010:5798000000001";
endpointReference.PageIdentifier = string.Empty;

ICredentials credentials = factory.CreateCredentials();
credentials.Username = "peppol";
credentials.Password = "peppol";

List<IEndpointReference> endpointReferences = factory.CreateInbox("LIMEAP").GetMessageList(credentials, endpointReference);
```

Below is an example of how to get at specific message from an Inbox.

```csharp
Factory factory = new Factory();
IEndpointReference endPointReference = factory.CreateEndpointReference();
endPointReference.ChannelIdentifier = "0010:5798000000002";
endPointReference.MessageIdentifier = "uuid:b7c4f236-2e75-485a-b8be-ea08a294f433";

ICredentials credentials = factory.CreateCredentials();
credentials.Username = "peppol";
```
Sending messages is just as easy. Below is shown an example of how to send a message. You need to specify the business identifier of both the sender and recipient. When the message is received at the Access Point (AP), the AP performs a lookup in the SML (Service Metadata Locator) and SMP (Service Metadata Publisher) based on the recipient identifier and document type, and forwards the message to the correct destination AP. It is important to note that the channel identifier determine on which AP the message is to be sent. The SMP/SML is described in detail in the next section.

```csharp
private IMessage CreateSampleMessage(Factory factory, String xmlFilename, string senderId, string receiverId)
{
    BusinessIdentifierType senderIdentifier = new BusinessIdentifierType();
    senderIdentifier.Value = senderId;
    senderIdentifier.scheme = "http://busdox.org/profiles/serviceMetadata/1.0/UniversalBusinessIdentifier/1.0/";

    BusinessIdentifierType receiverIdentifier = new BusinessIdentifierType();
    receiverIdentifier.Value = receiverId;
    receiverIdentifier.scheme = "http://busdox.org/profiles/serviceMetadata/1.0/UniversalBusinessIdentifier/1.0/";

    DocumentIdentifierType documentType = new DocumentIdentifierType();
    documentType.scheme = "http://busdox.org/profiles/serviceMetadata/1.0/DocumentTypeIdentifierType/QNameSubtype/";

    ProcessIdentifierType processIdentifierType = new ProcessIdentifierType();
    processIdentifierType.Value = "BI1I19";
    processIdentifierType.scheme = "http://busdox.org/processIdentifiers/1.0/UBL/CENBII/1.0/";

    //set in message object
    Metadata metadata = new Metadata();
    metadata.DocumentIdentifierType = documentType;
    metadata.ProcessIdentifierType = processIdentifierType;
    metadata.Receiver = receiverIdentifier;
    metadata.Sender = senderIdentifier;

    Message message = new Message(metadata);
    if (xmlFilename != null)
    {
        XmlDocument xmlDoc = new XmlDocument();
        xmlDoc.Load(xmlFilename);
        message.Document = xmlDoc;
    }
}
```

This function creates an `IMessage` object with all the required values. You may set your own information in the variables. Note that in order to perform the lookup, you need to set the document type.
return (IMessage)message;

The following information must be obtained from an LIME AP administrator prior to sending and receiving documents.

- Channel Identifier for each Business Identifier sending and receiving documents
- LIME Access Point URL for each Channel Identifier
- Username/password information needed to authenticate the calls

A sample project which contains unit tests for the LIME Client can be found in the SVN Repository at: https://svn.forge.osor.eu/svn/peppol/dotnet/transportLibrary/tags/v0.9.5.0

Configuring an Endpoint Reference for the LIME Access Point.

To be able to communicate with a LIME Access point, we first need to perform some configuration steps.

These are the steps to follow. These steps can be written in the configuration file or simply use the WCF Configuration Editor Tool as described in online MSDN http://msdn.microsoft.com/en-us/library/ms732009.aspx

1. Create a binding under system.serviceModel in the configuration file (preferable in the Test project) as shown below. Note that all the LIME AP use a custom Binding configuration. Set the binding properties as shown below.

```xml
<system.serviceModel>
  <bindings>
    <customBinding>
      <binding name="LIMEAPUnevenSecureCustomBinding" closeTimeout="00:20:00" openTimeout="00:20:00" receiveTimeout="00:20:00" sendTimeout="00:20:00">
        <textMessageEncoding messageVersion="Soap11WSAddressing10">
          <readerQuotas maxDepth="104857600" maxStringContentLength="104857600" maxArrayLength="104857600" maxBytesPerRead="104857600" maxNameTableCharCount="104857600" />
        </textMessageEncoding>
        <httpsTransport maxBufferPoolSize="104857600" maxReceivedMessageSize="104857600" authenticationScheme="Basic" maxBufferSize="104857600" />
      </binding>
    </customBinding>
  </bindings>
</system.serviceModel>
```

2. Create a client endpoint in the configuration file under system.serviceModel

```xml
<client>
  <!--iisEndPoint--> 
  <endpoint binding="customBinding" bindingConfiguration="LIMEAPUnsecureCustomBinding" contract="eu.peppol.lime.api.interfaces.Resource" name="iisEndPoint" address="http://localhost:8081/limeLibrary/ResourceService.svc">
    <identity>
      <dns value="localhost" />
    </identity>
  </endpoint>
</client>
</system.serviceModel>
```
Set the end point properties as shown above.

The properties:

i. **Binding** – is sent to the type of binding to be used – here it is customBinding

ii. **bindingConfiguration** – is set to the name of the binding that have just created.

iii. **Contract** – is set to "eu.peppol.lime.api.interfaces.Resource"

iv. **Name** – a name that you set for this end point

v. **Address** – is the address where you have deployed the LIMEService.

3. Use this endpoint as a reference in a LIME client like shown below

```csharp
[TestMethod]
public void GetMessage()
{
    Factory factory = new Factory();
    IEndpointReference endPointReference = factory.CreateEndpointReference();
    endPointReference.ChannelIdentifier = "0010:5798000000001";
    endPointReference.MessageIdentifier = "uuid:2651e277-3e51-4c6e-99c8-3375b5dcdea3";

    ICredentials credentials = factory.CreateCredentials();
    credentials.Username = "peppol";
    credentials.Password = "peppol";

    Message message = (Message)factory.CreateInbox("iisEndPoint").GetMessage(credentials, endPointReference);
}  
```

The endpoint – iisEndpoint- is used here as a configuration before communicating with the LIME Access point.
4 Lesson: Creating a LIME Access Point

A Lightweight Message Exchange Profile (LIME-AP) is an Access Point that implements the interface specified in the PEPPOL LIME Profile.

Companies identified by “Business Identifiers” can send and receive documents by using a LIME-AP.

The LIME-AP can be compared to a mailbox, where each mailbox is identified by a Channel Identifier and owned by a company (Business Identifier).

The main features of the LIME-AP are

- Send documents to Secure Trusted Asynchronous Reliable Transport (START - AP)
- Request documents from START AP.
- Has a WS-Transfer web service for sending and receiving documents
  - This interface may be used for internal communication between a START client and START service. – i.e. in this implementation, the LIME interface is implemented by a separate service. The main objectives of LIME Service are to send and request documents from START AP.
- Can perform a lookup in the SMP/SML to find the destination AP of a message

An easy way to build a LIME-AP is to download the sample LIME-AP from the SVN-repository and customize it.

https://svn.forge.osor.eu/svn/peppol/dotnet/transportLibrary/tags/v0.9.5.0/LIME

The next section shows a brief description of the LIME solution.

Some requirements before deploying the LIME-AP on your server are:

- Install Microsoft Frontpage Extensions 2002
- Create and add server certificates under IIS
- Tick the checkbox “require SSL”
- Create windows account, set NTFS permissions and select basic authentication

You may now deploy the LIME-AP on your server by right-clicking on the project and selecting “publish”.

A LIME-AP implementation can be tested by using the sample LIME-Client as described in the previous section.
4.1 Solution overview

The LIMELibrary.suo contains the following projects:

<table>
<thead>
<tr>
<th>Project</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IOLayerLibrary</td>
<td>A library that deals directly with the IO layer. It contains the implementation that manipulates the messages in the file system.</td>
</tr>
<tr>
<td>LIMELibrary</td>
<td>Contains the core types for building LIME Access Points and Clients, as well as the WSDL and Schemas used to create the client and server stubs.</td>
</tr>
<tr>
<td>LIMEService</td>
<td>Contains the implementation of the LIME Access Points (AP) that have been deployed in the PEPPOL infrastructure</td>
</tr>
<tr>
<td>LIMESTARTTestProject</td>
<td>Contain unit test that deals with the START clients.</td>
</tr>
<tr>
<td>LIMEtestProject</td>
<td>Contains all unit tests for the above projects. These units test deals with Java-LIME AP and Java-START AP. Also contains tests of prototypes of the LIME binding.</td>
</tr>
</tbody>
</table>

4.2 The IOLayerLibrary project

The IOLayerLibrary contain a class – IOLayer.cs. This class manages documents in the file system. It is capable of creating, updating, and deleting documents. It can also get the details of a particular document as well as get a list of messages.

The IOLayer creates two document folders – Inbox and Temporary. The Inbox folder contains the documents that are managed by the IOLayer.cs. Whereas the Temporary folder, messages are stored temporarily by the LIME AP. Before sending the messages details and content to START AP, the LIME stores the messages temporarily in the Temporary folder in the CREATE request. Then the LIME extracts the messages details and content of a documents and then send the details to the START client and at this moment, the messages in the Temporary folder is deleted.

4.3 The LIMELibrary project

The Schemas and WSDL under “LIMELibrary\xml-resources” has been used to generate the interface and proxy class “Resources.cs” under eu\peppol\start\interfaces. The main WSDL is a draft copy of the WS-Transfer specification. WS-Transfer defines a set of operations, of which the LIME specification uses ‘Create’, ‘PUT’, ‘GET’ and ‘DELETE’.

1. The ‘Create’ operation transfers a single resource (any XML document).
2. The ‘PUT’ operation transfers a single resource to the START AP.
3. The ‘GET’ operation can get the details of a single document or even a list of documents.
4. The ‘DELETE’ operations delete a particular document.

In order to create a client or service from this proxy, it must be instantiated and bound to a transport. The specific transport bindings are not present in the LIMELibrary solution, but rather in the test, Access Point and sample client solutions.

The LIMELibrary includes two sections: “impl” and “interfaces”. The “impl” section contains a class that deals with the Endpoint reference of a LIME client; a factory class that deals with the messaging among others.

The interfaces contain the generated proxy class and also some interface that define the credentials, end point reference, inbox as well as the outbox.
4.4 The LIMEService project

This project contains the implementation of the LIME Access Points that are deployed in the PEPPOL infrastructure.

The file “ResourceService.svc.cs” contains the service implementation, which implements the ‘Resources’ interface from the LIMELibrary.

    public class ResourceService : Resource

It implements all the operation define in the LIMELibrary. As it have been said before, when the message is received at the Access Point (AP), the AP performs a lookup in the SML (Service Metadata Locator) and SMP (Service Metadata Publisher) based on the recipient identifier and document type, and forwards the message to the correct destination AP. There are two LIME access points that can be deployed. One of them deals with even business identifiers and the other one deal with the uneven access point.

The web.config contains an appSetting section as shown below.

```xml
<appSettings>
    <!--
    For Even AP RepositoryResult = 0
    And for UnEven RepositoryResult = 1
    -->
    <add key="RepositoryResult" value="0" />
</appSettings>
```

When publish to an access point that deals with the even business identifiers, set the value of the “RepositoryResult” key to 0 and 1 when deployed to an Uneven access point.

The SML/SMP is described in the next section.

Deploying the AP

The AP is deployed to IIS in the usual way. For example, on localhost:

1. Choose a folder to publish the site to, using the VS2008 publishing wizard (right-click the project and choose ‘publish..’. You may choose a folder under c:\inetpub\www\)

2. From the IIS manager, create a virtual application for the folder.

3. Under security settings, add an SSL certificate to the virtual application. You may use “test-0000000003.pfx” under “STARTAccessPoint\src\SSL\Cert”, with password ‘peppol’.

4. Enable one-way SSL

4.5 The LIMESTARTTest project and LIMEtestproject

These projects contain numerous test projects that simulate LIME clients. The implementations have been described above in lesson 4.

4.6 The ServiceMetadataLibrary

The ServiceMetadata library is a lookup library. This library has the capability to return an access point based on certain parameters as shown below.

The implementation is found in “LIMEService\EndPointConfigurations.cs”, in the GetStartEndPoint method:
```csharp
public class EndPointConfigurations
{
    public static EndpointAddress GetStartEndPoint(string businessIdentifier, string documentType, string processIdentifier)
    {
        // constructing the URLs
        string registryUrl = "http://5798000000001.0010.ubis.smloc.org/smp/";
        string businessIdentifierUrl = string.Concat("http://busdox.org/profiles/serviceMetadata/1.0/UniversalBusinessIdentifier/1.0/::", businessIdentifier); // 0010:5798000000001"
        ServiceMetadataPublishing SMP = new ServiceMetadataPublishing();
        string endpoint = SMP.GetEndPointReference(registryUrl, businessIdentifierUrl, documentType, processIdentifier);
        if (String.IsNullOrEmpty(endpoint))
        {
            throw new Exception("The START endpoint is not available for the requestde Business Identifier..");
        }
        Uri uri = new Uri(endpoint);
        EndpointAddress endPointAddress = new EndpointAddress(uri);
        return endPointAddress;
    }
}
```

The method instantiates an object from `ServiceMetadataLibrary.dll` which was referenced by the LIMEService.

```csharp
ServiceMetadataPublishing SMP = new ServiceMetadataPublishing();
```

The method takes four parameters. The values of these parameters are filled during the testing process as describe in lesson 4 when creating an IMessage object.

```csharp
BusinessIdentifierType senderIdentifier = new BusinessIdentifierType();
senderIdentifier.Value = senderId;
senderIdentifier.scheme = "http://busdox.org/profiles/serviceMetadata/1.0/UniversalBusinessIdentifier/1.0/";
```

```csharp
BusinessIdentifierType receiverIdentifier = new BusinessIdentifierType();
receiverIdentifier.Value = receiverId;
receiverIdentifier.scheme = "http://busdox.org/profiles/serviceMetadata/1.0/UniversalBusinessIdentifier/1.0/";
```

```csharp
DocumentIdentifierType documentType = new DocumentIdentifierType();
documentType.scheme = "http://busdox.org/profiles/serviceMetadata/1.0/DocumentTypeIdentifierType/QNameSubtype/";
```

```csharp
ProcessIdentifierType processIdentifierType = new ProcessIdentifierType();
processIdentifierType.Value = "BII19";
processIdentifierType.scheme = "http://busdox.org/processIdentifiers/1.0/UBL/CENBII/1.0/";
```

The ServiceMetadataLibrary uses these parameters to construct a URL that returns a list of START Access points. Now, based on the process identifier, here it is "BII19", a START Access point is returned. This in turn is used to connect to the START client.

The solution of this library is found under
https://svn.forge.osor.eu/svn/peppol/dotnet/servicemetadata/tags/v0.9.5.0

The ServiceMetadata.suo contains these list projects:

<table>
<thead>
<tr>
<th>Project</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ServiceMetaDataLibrary</td>
<td>Contain the core implementation of this solution. Able to</td>
</tr>
<tr>
<td>ServiceMetaDataLocatorTestProject</td>
<td>Contains Unit Test that deals with the SML Manage business Identifier.</td>
</tr>
<tr>
<td>SMLTestProject</td>
<td>Contains numerous unit tests that deal with the core implementation of the library.</td>
</tr>
</tbody>
</table>
5 Lesson: Creating a START Access Point

5.1 Introduction
A START Access Point (START AP) is an Access Point that implements the BUSDOX START profile (Secure Trusted Asynchronous Reliable Transport). The START code provided by PEPPOL covers the following:

- Sample Access Point implementation
- Sample client implementation
- Library

Prerequisites
Apart from .NET 3.5 and Visual Studio 2008, the code relies on Microsoft “Windows Identity Foundation” (formerly ‘Geneva’) framework libraries. Two extra libraries is needed, System.IdentityModel.dll and System.ServiceModel.dll. Both libraries have to be registered in the GAC before use.

5.2 Solution overview
The START.suo solution contains the following projects:

<table>
<thead>
<tr>
<th>Project</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>STARTLibrary</td>
<td>Contains the core types for building START Access Points and Clients, as well as the WSDL and Schemas used to create the client and server stubs.</td>
</tr>
<tr>
<td>STARTAccessPoint</td>
<td>Contains the implementation of the START Access Points (AP) that have been deployed in the PEPPOL infrastructure</td>
</tr>
<tr>
<td>SampleSTARTClient</td>
<td>A sample START client implementation</td>
</tr>
</tbody>
</table>

5.3 The STARTLibrary project
The Schemas and WSDL under “STARTLibrary\xml-resources\0.9a” (version 1.0) has been used to generate the interface and proxy class “Resource.cs” under eu\peppol\start\interfaces. The main WSDL is a draft copy of the WS-Transfer specification. WS-Transfer defines a set of operations, of which the START specification uses ‘Create’. The ‘Create’ operation transfers a single resource (any XML document).

In order to create a client or service from this proxy, it must be instantiated and bound to a transport. The specific transport bindings are not present in the STARTLibrary solution, but rather in the test, Access Point and sample client solutions.

The STARTLibrary also includes a simple client (“SimpleAccessPointClient”, which implements “IAccessPointClient.cs”). This client uses the autogenerated proxy classes (based on “ws-tra.wsdl”) in the ‘Resource.cs’ file. See the SampleSTARTClient and STARTService projects for examples of how to use these classes.

The “IMessageMetadata” interface represents the message metadata exchanged between access points – such the sender and recipient business identifiers, the channel identifier of a recipient, a document type identifier, and a profile identifier for indicating the profile or process a message is exchanged under.

These values will be set as SOAP headers in the on-the-wire-format.
5.4 The STARTAccessPoint project

This project contains the implementation of the sample Access Points that are deployed in the PEPPOL infrastructure.

The file “STARTAccessPoint.svc.cs” contains the service implementation, which implements the ‘Resource’ interface from the STARTLibrary.

```csharp
public class STARTAccessPoint : Resource
```

Of all the operations of WS-Transfer, it only implements the ‘Create’ operation. When a message is sent to an AP through the ‘Create’ operation, an empty response is returned if it goes well, and a SOAP fault otherwise.

There are 2 Sample Access Points deployed, which are both instances of this service, and running in the same process.

Storing of received messages

Received messages are stored on the filesystem, based on the value of the recipient business identifier. Using the “IOLayer.cs” class in “STARTAccessPoint\src\IO”, a folder is created for each recipient identifier, and for each message XML 2 files are created:

- One holding the message sent to the AP (the payload of ‘Create’)
- One holding the headers of the message, i.e. recipient and sender identifiers, document and profile/process type information.

On an XP machine, a ‘peppol’ folder will be placed under “C:\Documents and Settings\All Users\Application Data\peppol\MessageStore\Inbox\”.

In the PEPPOL infrastructure, after a message has been received by a sample START AP, it may be retrieved by a LIME client through a LIME AP. The PEPPOL sample LIME AP implementation can then retrieve messages from the same file structure as the START AP writes message to.

Note that for the sake of simplicity, the sample START AP implementation is extremely ‘lax’. It will accept any incoming message if the recipient business identifier is even/odd according to policies.

**Configuration**

The endpoints and the bindings of the endpoints are configured in `web.config`. Example of the first service endpoint definition:

```xml
<endpoint address="" binding="customBinding" bindingConfiguration="START.SecurePeppolBinding" contract="eu.peppol.start.common.Resource"/>
```

Binding definition for the simplified START binding:

```xml
<binding name="SecurePeppolBinding">
    <reliableSession acknowledgementInterval="00:00:10" ordered="true"
        reliableMessagingVersion="WSReliableMessaging11"
        flowControlEnabled="false" maxRetryCount="1"
        inactivityTimeout="00:00:10" />
    <security messageSecurityVersion="WSSecurity10WSTrust13WSSecureConversation13WSSecurityPolicy12BasicSecurityProfile10"
        allowSerializedSigningTokenOnReply="true"
        messageProtectionOrder="SignBeforeEncrypt"
        authenticationMode="MutualCertificate"
        requireDerivedKeys="false"
        requireSignatureConfirmation="false"
        defaultAlgorithmSuite="Basic128"
        securityHeaderLayout="Lax" />
</binding>
```

Note that the sample implementation is extremely ‘lax’.
Deploying the AP

The AP is deployed to IIS in the usual way. For example, on localhost:

5. Choose a folder to publish the site to, using the VS2008 publishing wizard (right-click the project and choose ‘publish.’). You may choose a folder under c:\inetpub\www\.

6. From the IIS manager, create a virtual application for the folder.

7. Under security settings, add an SSL certificate to the virtual application. You may use “test-0000000003.pfx” under “STARTAccessPoint\Certificates”, with password ‘peppol’.

8. Enable one-way SSL

5.5 The SampleSTARTClient project

This project implements a sample START client, based on the STARTLibrary core, and the simplified START binding. The class “SampleSTARTClient” (under “src\eu\peppol\start\samples\client”) shows a stand-alone START client.

It works by

1. Instantiating the SimpleAccessPointClient class from the STARTLibrary project

2. Construct a sample message, and sample metadata

3. Send using the SimpleAccessPointClient, close after use (since the binding supports WS-ReliableMessaging, as long as the client is open, and the session has not timed out, additional messages are sent using the established WS-Security session)

```csharp
// 1. Instantiate the client
SimpleAccessPointClient startClient = new SimpleAccessPointClient(
    endpointConfigName, 
    3, 
    new Uri("http://SomeAccessPoint.org/MyIssuingAP"), 
    "sm11.smloc.org", 
    Certificates.ClientCertificate, 
    Certificates.ServiceCertificate);

// 2. Create sample data:
IMessageMetadata metadata = new MessageMetadata();
metadata.RecipientIdentifier = new ParticipantIdentifierType();
metadata.RecipientIdentifier.Value = "0010:5798000000002";
metadata.RecipientIdentifier.scheme = "busdox-actorid-upis";

metadata.SenderIdentifier = new ParticipantIdentifierType();
metadata.SenderIdentifier.Value = "0010:5798000000001";
metadata.SenderIdentifier.scheme = "busdox-actorid-upis";

metadata.DocumentIdentifier = new DocumentIdentifierType();
metadata.DocumentIdentifier.Value = "AcceptCatalogue##UBL-2.0";

metadata.ProcessIdentifier = new ProcessIdentifierType();
metadata.ProcessIdentifier.Value = "BII01";
metadata.ProcessIdentifier.scheme = "cenbii-procid-ubl";
```
Random rnd = new Random((int)DateTime.Now.Ticks);

XmlDocument document = new XmlDocument();
document.LoadXml("<?xml version='1.0' encoding='UTF-8'?><s:Test xmlns:s="http://test.dk">" + "MessageID: " + messageText + "</s:Test>" fit);

// 3. Send the XML:
startClient.Send(metadata, document);

The Send method calculates the address of the next endpoint from MessageMetadata and Domain. If the Domain e.g. ("sml1.smloc.org") is empty then Send method gets the endpoint from the configuration file referenced by “endpointConfigName”.

SSL trust
When using the binding that supports WS-Reliable Messaging with SSL, and running against services where the SSL CN does not match the domain of the service, you can require the code to ignore thus:

System.Net.ServicePointManager.ServerCertificateValidationCallback =
((sender, certificate, chain, sslPolicyErrors) => true);