



**Guideline**



|  |  |
| --- | --- |
| **Project Acronym:** | **PEPPOL** |
| **Grant Agreement number:** | **224974** |
| **Project Title:** | **Pan-European Public Procurement Online** |
| **Transport Infrastructure ICT**  **Services-Components**  **PEPPOL-Silicone**  **SMP Developer Guide**  **Version: 2.2.1**  **Status: In Use**  **Editors:**  **Philip Helger, PEPPOL.AT** | |

|  |  |  |
| --- | --- | --- |
| **Project co-funded by the European Commission within the ICT Policy Support Programme** | | |
| **Dissemination Level** | | |
| **P** | **Public** | **X** |
| **C** | **Confidential, only for members of the consortium and the Commission Services** |  |

**Revision History**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Version** | **Date** | **Editor** | **Org** | **Description** |
| 2.2.1 | 2012-03-29 | PH | BRZ | Initial version for peppol-silicone-2.2.1 |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

**Contributors**

**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).

**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This deliverable contains original unpublished work except where clearly indicated otherwise.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.

**Organisations**

PEPPOL.AT/BRZ (Bundesrechenzentrum), AT, <http://www.brz.gv.at>

**Persons**

PH: Philip Helger (PEPPOL.AT/BRZ)

**Table of Contents**

[1 Introduction 5](#_Toc320784494)

[1.1 Objective and Scope 5](#_Toc320784495)

[1.2 Audience 5](#_Toc320784496)

[2 Prerequisites 5](#_Toc320784497)

[2.1 Environment 5](#_Toc320784498)

[2.2 Resources and libraries 5](#_Toc320784499)

[3 Getting and Compiling the Source Code 6](#_Toc320784500)

[3.1 General 6](#_Toc320784501)

[3.2 transport-start-server 6](#_Toc320784502)

[3.3 transport-start-client 6](#_Toc320784503)

[4 START AP service 7](#_Toc320784504)

[4.1 Building and deploying the service 7](#_Toc320784505)

[4.2 Configuring the service 7](#_Toc320784506)

[4.3 Implementation overview 9](#_Toc320784507)

[5 START AP client library 10](#_Toc320784508)

[5.1 Building and packing 10](#_Toc320784509)

[5.2 Using the library 10](#_Toc320784510)

# Introduction

## Objective and Scope

This document is a developer guide to the components of the PEPPOL Java START AccessPoint (AP) service. The components described in this document are:

* A START AP service (project name: transport-start-server)
* A START client library (project name: transport-starkl-client)

This version of the document is for peppol-silicone v2.2.1 to be found at <http://code.google.com/p/peppol-silicone/>

Before reading this document it is highly recommended to go through the "Setting up the development environment" document for the machine setup.

## Audience

The audience for this document is organizations in need for a PEPPOL SMP. These may include the following PEPPOL Stakeholders:

* PEPPOL Community Governance
* Contracting Authorities
* Economic Operators
* ICT Providers
* Service Providers

More specific it is the following roles:

* ICT Architects
* ICT Developers

# Prerequisites

## Environment

Software components required:

* JDK 6 or later
* Tomcat 6 or 7 server with Metro framework
* Apache Maven 3.x

Recommendations for development:

* Eclipse 3.6 or higher - <http://www.eclipse.org/>

Recommended tools for contributors to the online community

* Subclipse – Subversion plugin for Eclipse - <http://subclipse.tigris.org>
* TortoiseSVN –Subversion client for Windows - <http://tortoisesvn.net/>

The library has been tested on the following operating systems:

* Windows Server 2003, XP and 7
* Ubuntu 8.10

## Resources and libraries

All prerequisite libraries are already included in the Maven dependencies but are listed for completeness.

### Prerequisites for transport-start-server

Resources:

* A keystore with a PEPPOL AP key

PEPPOL projects used:

* SMP client library (smp-client-library)
* Commons PEPPOL (commons-peppol)
* Commons BusDox (commons-busdox)
* Transport API (transport-api)

Third-party libraries used:

* Jersey Client 1.12
* Metro WS runtime 2.1.1
* log4j 1.2.16 via SLF4J 1.6.2
* phloc commons 3.4.8

### Prerequisites for transport-start-client

PEPPOL projects used:

* Commons PEPPOL (commons-peppol)
* Commons BusDox (commons-busdox)
* Transport API (transport-api)

Third-party libraries used:

* Metro WS runtime 2.1.1
* phloc commons 3.4.8
* SLF4J 1.6.2

### Change history

|  |  |
| --- | --- |
| Version | Changes |
| 2.2.1 | Fixed error in transport-start-filereceiver SPI configuration file |
| 2.2.0 | The endpoint address of the START AP is now read from the configServer.properties configuration file to be more flexible |

# Getting and Compiling the Source Code

## General

All of the components are added to the repository as Eclipse projects. The easiest way to build the projects is therefore to import the projects into Eclipse including all related PEPPOL projects.

## transport-start-server

The trunk of the START AP server project source code is located at:

[https://peppol-silicone.googlecode.com/svn/trunk/java/transport-start-server](https://peppol-silicone.googlecode.com/svn/trunk/java/smp-webapp)

The tags of the component are located at:

[https://peppol-silicone.googlecode.com/svn/tags/peppol-transport-start-server](https://peppol-silicone.googlecode.com/svn/tags/peppol-smp-webapp)

As this is a web application (WAR) an easy-to-use Jetty setup is added within the test-part of the application. It can be used to test the START AP locally from within the IDE. To run the START AP web application from within your IDE run the class at.peppol.transport.start.standalone.RunInJettySTARTServer. It will spawn the START AP on your local machine on port 8090. Direct your browser to the URL http://localhost:8090/ and you should see the text "Welcome". Direct your browser to the URL http://localhost:8090/accessPointService and you should see the an auto-generated page with the available Web Services.

If you get errors starting the START AP server, you may need to do some configuration. See the sections below for details.

## transport-start-client

The trunk of the START AP client project source code is located at:

[https://peppol-silicone.googlecode.com/svn/trunk/java/transport-start-client](https://peppol-silicone.googlecode.com/svn/trunk/java/smp-client-library)

The tags of the component are located at:

<https://peppol-silicone.googlecode.com/svn/tags/peppol-transport-start-client>

The START client is only a library, that is not runnable by itself. You need to wrap it in an application to get it working!

# START AP service

The START AP service has been implemented as a WebService using the Metro Framework.

## Building and deploying the service

The service can be deployed in two ways:

* Compiling the project on the command line using mvn clean install. The result is a WAR file in the target folder and additionally an exploded version of the WAR file in the target/peppol-transport-start-server-x.y.z directory.
* Start the application src/test/java/at/peppol/transport/start/standalone/RunInJettySTARTServer from within Eclipse. Than the application will be running on port 8090 (see above).

Metro must be installed on the Tomcat server for the service to work, since it makes use of the SML management client.

Note that the START AP service can be deployed with any context name and must not be the ROOT application. Since the production START AP must run using HTTPS it is recommended to deploy it on port 443 (standard https port) as other ports may not be reachable from all senders!

Hint when using Eclipse: it is best to close the project in Eclipse, because otherwise Eclipse might want to refresh while the console build is in progress. After the build finished you may re-open the project Eclipse and clean it there again, because the Eclipse compiler and the Sun console compiler produce incompatible byte code for enum classes!

## Configuring the service

The service is configured using three configuration files, all residing in src/main/resources:

* configOCSP.properties containing the OCSP configuration (extended certificate validation)
* configSAML.properties containing the SAML configuration
* configServer.properties containing the START AP configuration

Please note that these files currently contain redundant information. Trust store information is needed in configOCSP.properties and configServer.properties. Key store information is needed in configSAML.properties and configServer.properties. Please make sure that you pass the same values in all files.

### configOCSP.properties

As OCSP configuration is only about the trust store, and peppol-silicone ships with a default trust store, usually there is no need to modify the existing file at all.

The following list describes all the possible configuration items:

* **ocsp.truststore.path**: Contains the classpath relative path to the JKS file containing the trust store. As the trust store is provided globally, you should not need to modify this value. The default value is truststore/global-truststore.jks. The file itself resides inside the commons-peppol project.
* **ocsp.truststore.password**: The password required to access the global trust store. When using the default trust store provided than the password is peppol.
* **ocsp.truststore.alias**: The alias inside the trust store that contains the AP certificate. When using the default trust store provided than the alias is peppol access point test ca (peppol root test ca).

The default configOCSP.properties file looks like this:

# Path to the trust store JKS key store

ocsp.truststore.path=truststore/global-truststore.jks

# Password used to access the trust store

ocsp.truststore.password=peppol

# The alias of the root certificate to use for trusting

ocsp.truststore.alias=peppol access point test ca (peppol root test ca)

Note: since the default Java properties file handling is used to read the configuration file recall that trailing whitespaces of a property name and leading white spaces of a property value are automatically skipped. Trailing whitespaces of a property value are not skipped!

### configSAML.properties

This file contains the configuration for SAML.

The following list describes all the possible configuration items:

* **peppol.senderid**: Contains the sender ID. This ID is used as the name of the SAML subject. If unsure what value to use you can use busdox:sender as the value.
* **peppol.servicename**: Contains the name of the access point. This name is used as the issuer of the SAML assertion. If unsure what value to use you can use PEPPOL-START-AP as the value.
* **peppol.keystore**: The classpath relative path to the JKS key store containing your private key.  
  Example when using the source distribution: create the folder src/main/resources/keystore, put your JKS file in this folder (e.g. called peppolkeys.jks), than the respective configuration value would be keystore/peppolkeys.jks.
* **peppol.password**: The password required to access the provided key store. This is the password you defined when creating the key store.
* **peppol.key-alias**: The alias inside your key store that points to the key pair for your access point.
* **peppol.key-password**: The password used to access the key pair identified by the peppol.key-alias within the key store. Usually it is the same password as the password used to access the whole key store (see property peppol.passsword)

This is an example configSAML.properties file:

peppol.senderid = busdox:sender

peppol.servicename = PEPPOL-START-AP

peppol.keystore = keystore/peppolkeys.jks

peppol.password = peppol

peppol.key-alias = ap\_key

peppol.key-password = peppol

Note: since the default Java properties file handling is used to read the configuration file recall that trailing whitespaces of a property name and leading white spaces of a property value are automatically skipped. Trailing whitespaces of a property value are not skipped!

### configServer.properties

This file contains the configuration for the START AP service itself.

The following list describes all the possible configuration items:

* **server.endpoint.url**: Contains the full URL under which the AP will be available. This URL must include the name of the Web Service URL (e.g. /accessPointService).
* **server.keystore.path**: The classpath relative path to the JKS key store containing your private key.  
  Example when using the source distribution: create the folder src/main/resources/keystore, put your JKS file in this folder (e.g. called peppolkeys.jks), than the respective configuration value would be keystore/peppolkeys.jks.
* **server.keystore.password**: The password required to access the provided key store. This is the password you defined when creating the key store.
* **server.keystore.alias**: The alias inside your key store that points to the key pair for your access point.
* **server.keystore.aliaspassword**: The password used to access the key pair identified by the peppol.key-alias within the key store. Usually it is the same password as the password used to access the whole key store (see property peppol.passsword)
* **server.truststore.path**: Contains the classpath relative path to the JKS file containing the trust store. As the trust store is provided globally, you should not need to modify this value. The default value is truststore/global-truststore.jks. The file itself resides inside the commons-peppol project.
* **server.truststore.password**: The password required to access the global trust store. When using the default trust store provided than the password is peppol.
* **server.truststore.alias**: The alias inside the trust store that contains the AP certificate. When using the default trust store provided than the alias is peppol access point test ca (peppol root test ca).
* **server.truststore.aliaspassword**: The password used to access the certificate identified by the server.truststore.alias within the trust store. When using the default trust store provided than the password is peppol

This is an example configServer.properties file:

# What is the public URL of our AP?

server.endpoint.url = https://www.example.org/accessPointService

# Keystore configuration - fill your classpath relative values here

server.keystore.path = keystore/peppolkeys.jks

server.keystore.password = peppol

server.keystore.alias = ap\_key

server.keystore.aliaspassword = peppol

# Truststore configuration - no need to change anything

server.truststore.path=truststore/global-truststore.jks

server.truststore.password=peppol

server.truststore.alias=peppol access point test ca (peppol root test ca)

server.truststore.aliaspassword=peppol

Note: since the default Java properties file handling is used to read the configuration file recall that trailing whitespaces of a property name and leading white spaces of a property value are automatically skipped. Trailing whitespaces of a property value are not skipped!

## Implementation overview

The START AP makes heavy use of the Metro framework. That's why the main implementation is pretty small and only consists of one class at.peppol.transport.start.server.AccessPointService.

This class is invoked as a WebService server for new incoming documents. Sending documents is only done via the START AP client library.

The common WSDL file used for all implementations resides in the commons-busdox projects in the file src/main/resources/WEB-INF/wsdl/ peppol-start-2.0.wsdl. It is mirrored into this project to avoid having an out-of-date per-project copy of this file. The is **NO** need to modify this WSDL and for compatibility reasons it is **highly discouraged** to modify it! The project local copy of the file resides inside the src/main/webapp/WEB-INF/wsdl directory.

The only implemented method is CreateResponse create (Create aBody) which does all the hard work. After some checks whether the incoming document is correctly targeted for us, the receiving business logic is triggered via SPI invocation. SPI is a standard Java way to loosely couple components at runtime[[1]](#footnote-1). To add SPI implementations to your START AP service there are several possibilities to do it:

* For the source distribution add a Maven dependency in the pom.xml that references your SPI implementation project
* For the source distribution add the SPI implementation directly into this project
* For the binary distribution add the pre-compiled JAR file into the WEB-INF/lib directory and restart the application inside the application server

As there is no generally correct handling of received documents, the START AP service ships with a demo implementation, that stores incoming files into the file system. The implementation of this can be found inside the project transport-start-filereceiver which is a part of the peppol-silicone. It is currently directly referenced from the pom.xml file of the transport-start-server project but can be easily replaced or removed. It is also recommended to use the transport-start-filereceiver project as the basis for your own implementation. As an alternative to handle incoming documents in the filesystem you may consider using a database or a JMS system to store the documents, or you may simply invoke another WebService or REST client to forward the message to your handling backend system.

When creating your own SPI implementation, you must implement the SPI interface at.peppol.transport.start.server.IAccessPointServiceReceiverSPI that is located inside the transport-api project (which transitively depends on the PEPPOL projects commons-peppol and commons-busdox as well as some thirdparty libraries).

# START AP client library

## Building and packing

The easiest way to build the library is using Eclipse. A JAR file can be created by calling mvn clean install on the command line. This produces a library that is not runable by itself but needs to be included in another application.

Hint when using Eclipse: it is best to close the project in Eclipse, because otherwise Eclipse might want to refresh while the console build is in progress. After the build finished you may re-open the project Eclipse and clean it there again, because the Eclipse compiler and the Sun console compiler produce incompatible byte code for enum classes!

## Using the library

The AP client requires a configServer.properties file to be present for key store configuration as well as a configSAML.properties file for SAML configuration. As the transport-start-server project depends on the transport-start-client library, the client library does not ship with its own configuration files.

The WSDL file for the START AP client resides in the file src\main\resources\META-INF\AccessPointClient.xml and makes use of some callbacks for key store and trust store retrieval. There is no need to modify this WSDL file.

The at.peppol.transport.start.client.AccessPointClient is the main class when using the library. The class contains static methods for sending documents to a receiving AP. The class is documented using JavaDoc. The main method for sending document has the signature

1. **public** **static** ESuccess send (@Nonnull **final** String sAddressURL,
2. @Nonnull **final** IMessageMetadata aMetadata,
3. @Nonnull **final** Document aXMLDoc)

Parameters:

* sAddressURL is the URL of the receicing START AP. It must be determined using the SMPServiceCaller class from the smp-client-library project and must include the WebService context path (e.g. /accessPointService) but must not be the WSDL path
* aMetaData contains the message metadata (message ID, channel ID, sender ID, receiver ID, document type ID and process ID). The default implementation class if IMessageMetadata is the class at.peppol.transport.MessageMetadata from the transport-api project.
* aXMLDoc the XML document to be send. As PEPPOL can only handle XML documents this must be an instance of org.w3c.dom.Document.

Return value: the returned value is either ESuccess.SUCCESS or ESuccess.FAILURE which are just wrappers for boolean values with a predefined semantics. In case of a failure, please look into the log file for details.

1. Detailed information on SPI can be found at <http://docs.oracle.com/javase/6/docs/api/java/util/ServiceLoader.html> [↑](#footnote-ref-1)