PEPPOL
PAN-EUROPEAN PUBLIC PROCUREMENT ONLINE

cip competitiveness and innovation framework programme 2007-2013

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: 2.2.0
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
0.1.0	2009-04-23	CUP	Accenture	Initial draft
0.1.1	2009-04-30	CUP	Accenture	Added step by step build guide
0.2.0	2009-10-15	CUP	Accenture	Updated to 0.9 specification
0.3.0	2009-10-19	CUP	Accenture	Updated host name
0.4.0	2010-01-21	CUP	Accenture	Updated to match 1.0 specification
2.2.0	2012-03-12	PH	BRZ	Updated to peppol-silicone 2.2.0
	2012-03-21	PH	BRZ	Added additional descriptions Updated to new layout
	2012-03-29	PH	BRZ	Fixed some typos

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This deliverable contains original unpublished work except where clearly indicated otherwise.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

Contributors

Organisations

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

Persons

PH: Philip Helger (PEPPOL.AT/BRZ)
CUP: Christian Uldall Pedersen (Accenture)
DS: Dennis Søgaard (Accenture)

Table of Contents

1 Introduction

1.1 Objective and Scope

This document is a developer guide to the components of the PEPPOL Java Service Metadata Publisher (SMP) service. The components described in this document are:

- A SMP REST service (project name: smp-webapp)
- A SMP client library (project name: smp-client-library)

This version of the document is for peppol-silicone v2.2.0 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 infrastructure" document for the machine setup.

1.2 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

2 Prerequisites

2.1 Environment

Software components required:

- JDK 6 or later
- Tomcat 6 server with Metro framework
- Apache Maven 3.x
- MySQL

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

2.2 Resources and libraries

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

2.2.1 Prerequisites for smp-webapp

Resources:

- A MySQL database with the SMP database installed.
- A keystore with a PEPPOL SMP key

PEPPOL projects used:
- SML client library (sml-client-library)
- Commons PEPPOL (commons-peppol)
- Commons BusDox (commons-busdox)
- Commons JPA (commons-jpa)

Third-party libraries used:
- EclipseLink 2.3.2 as JPA2 handler
- MySQL JDBC connector (GPL v2!)
- Jersey Server 1.11
- Metro WS runtime
- log4j 1.2.16 via SLF4J 1.6.2
- phloc commons 3.4.2

2.2.2 Prerequisites for smp-client-library

PEPPOL projects used:
- Commons PEPPOL (commons-peppol)
- Commons BusDox (commons-busdox)

Third-party libraries used:
- Jersey Client 1.11
- Metro WS runtime
- phloc commons 3.4.2
- SLF4J 1.6.2

3 Getting and Compiling the Source Code

3.1 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.

3.2 smp-webapp

The trunk of the SMP REST service project source code is located at:
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-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 SMP locally from within the IDE. To run the SMP web application from within your IDE run the class `at.peppol.smp.server.jetty.RunInJettySMP`. It will spawn the SMP on your local machine on port 80. Direct your browser to the URL `http://localhost` and you should see the text "PEPPOL SMP".

3.3 smp-client-library

The trunk of the SMP REST service project source code is located at:
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-smp-client-library

4 REST service

The SMP service has been implemented as a REST interface with a database backend. It is possible to change the backend, but the following description is based on a database backend. A copy of the database can be found in the `/src/etc/database_backups/` folder.

4.1 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-smp-webapp-x.y.z` directory.
- Start the application `src/test/java/at/peppol/smp/server/jetty/RunInJettySMP` from within Eclipse. Than the application will be running on port 80.

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 SMP service must be deployed as the ROOT web application (at path "/") on the application server, since this is a prerequisite in the DNS lookup scheme. Further it must be deployed on port 80 (standard http port) and may not use SSL to secure the transport.

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!

4.2 Configuring the service

The service is configured using a single configuration file `src/main/resources/config.properties`. The following list describes all the possible configuration items:

- **dataManager.class**: The data manager implementation to use. The data manager is for retrieving the data to use in the REST service. The default class is `at.peppol.smp.server.data.dbms.DBMSDataManager`
- **registrationHook.class**: The type of registration hook to use. The hook is used for notifying the SML of the creation or deletion of business identifiers. For testing purposes you may use `at.peppol.smp.server.hook.DoNothingRegistrationHook` which does not communicate with the SML. For production use the class `at.peppol.smp.server.hook.RegistrationServiceRegistrationHook` must be used, as it communicates with the SML and adds, updates or deletes participant DNS entries
- **regServiceRegistrationHook.id**: The SMP ID to use when using the SML interface. Note: it must be the same ID that was used for the initial registration of the SMP to the SML.
  Note: is only required if class RegistrationServiceRegistrationHook is used.
- **regServiceRegistrationHook.regLocatorUrl**: The URL of the SML manage business identifier service. For production purposes (SML) use `https://sml.peppolcentral.org/manageparticipantidentifier`. For the test SML (SMK) use the URL `https://smk.peppolcentral.org/manageparticipantidentifier`.
  Note: is only required if class RegistrationServiceRegistrationHook is used.
- **regServiceRegistrationHook.keystore.classpath**: The classpath - relative to the project - where the Java key store (of type JKS) with the SMP certificate is located. An empty directory `src/main/resources/keystore` is present which could contain the key store. In this case the properties entry should start with "keystore/".
  Note: The key store should contain exactly one certificate entry with an arbitrary name and the certificate must have the same password as the whole key store!
  Note: is only required if class RegistrationServiceRegistrationHook is used.
- **regServiceRegistrationHook.keystore.password**: The password used to access the key store.
  Note: is only required if class RegistrationServiceRegistrationHook is used.
- **xmldsig.keystore.classpath**: Has the same semantics as regServiceRegistrationHook.keystore.classpath and should therefore have the same value

- **xmldsig.keystore.password**: Has the same semantics as regServiceRegistrationHook.keystore.password and should therefore have the same value
- **xmldsig.keystore.key.alias**: The alias of the key within the key store. Is case sensitive and may not be empty.
- **xmldsig.keystore.key.password**: The password of the certificate with the above specified alias. Should be the same as the password of the whole key store (see xmldsig.keystore.password).
- **jdbc.driver**: The JDBC driver class to be used by JPA. For MySQL use `com.mysql.jdbc.Driver`
- jdbc.url: The JDBC URL of the database to connect to. For a local MySQL database called "smp" the string would look like this: `jdbc:mysql://localhost/smp`
  Note: the URL depends on the JDBC driver used!
- **jdbc.user**: The database user to be used when connecting to the database.
- **jdbc.password**: The password of the JDBC user to be used when connecting to the DB
- **target-database**: The JPA target database type to be used. For MySQL this value should be `MySQL`
  Note: Please see the documentation of EclipseLink for other target database systems!
- **jdbc.read-connections.max**: The maximum number of JDBC connections to be used for reading. Usually 10 should be suitable for most use cases.

Example of a development config.properties file using a local MySQL database called "smp" (without an SML connector for ease of use):

```
## DBMS handler
dataManager.class=at.peppol.smp.server.data.dbms.DBMSDataManager

## Registration callback - do not communicate with SML
registrationHook.class=at.peppol.smp.server.hook.DoNothingRegistrationHook

## XMLDSIG response signing:
xmldsig.keystore.classpath    = keystore/smp_keystore.jks
xmldsig.keystore.password     = peppol
xmldsig.keystore.key.alias    = austrian smp brz
xmldsig.keystore.key.password = peppol

## JDBC configuration for DB
jdbc.driver = com.mysql.jdbc.Driver
jdbc.url = jdbc:mysql://localhost/smp
jdbc.user = smp
jdbc.password = smp
target-database = MySQL
jdbc.read-connections.max = 10
```

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!

4.3 Implementation overview

The service is implemented as a REST interface using the Jersey framework (http://jersey.java.net/). See the BusDox SMP specification for details on the intention of the REST URL paths.

4.3.1 Public REST API

The implementation contains four main REST classes, one for each path for which one can make queries:

- `/{ServiceGroupId}` is implemented in the `at.peppol.smp.server.ServiceGroupInterface` class.

- `/{ServiceGroupId}/services/{DocumentTypeId}` is implemented in the `at.peppol.smp.server.ServiceMetadataInterface` class.
- `/list/{UserId}` is implemented in the `at.peppol.smp.server.ListInterface` class.
- `/complete/{ServiceGroupId}` is implemented in the `at.peppol.smp.server.CompleteServiceGroupInterface` class.

The `list` interface is not part of the SMP specification and is used for getting a list of registered service groups for a given user. The complete interface is also not part of the specification and is used for getting the service group as well as all the service metadata for that group in a single call. Further the PUT and DELETE operations has also been implemented for `CompleteServiceGroupInterface`, `ServiceGroupInterface` and `ServiceMetadataInterface`.
These four interfaces make use of the `at.peppol.smp.server.data.IDataManager` interface to access the SMP data. The implementation currently contains one implementation of this interface which has a database as the underlying data source. The database version of the interface is implemented in the class `at.peppol.smp.server.data.dbms.DBMSDataManager` and makes use of JPA2 (EclipseLink) for the main database access.

4.3.2 Modifying REST API

Some REST APIs are used to modify data within the SMP. Those APIs are not standardized by the PEPPOL SMP specifications and therefore are described here. All modifying REST APIs are either using HTTP PUT or HTTP DELETE for creating/updating or for deleting elements. As the SMP itself running without transport security these modifying methods require an HTTP BasicAuth header.

Note: to send an HTTP Basic Auth header you must provide an HTTP header called `Authorization` with a value like created by the following pseudo code:
`"Basic " + Base64.encode (`*`userName`*` + ":" + `*`password`*`)`

Please see chapter 4.4 on security considerations how to handle BasicAuth proper and secure.

- CompleteServiceGroupInterface
  - `DELETE /complete/{ServiceGroupId}`
    Expected body: none
    Description: Delete the whole service group with the specified service group ID
    Status: deprecated because it has the same semantics as using DELETE on the ServiceGroupInterface (see below)
- ServiceGroupInterface
  - `PUT /{ServiceGroupId}`
    Expected body: A `ServiceGroupType` object as specified by the SMP XSD
    Description: create a new service group
  - `DELETE /{ServiceGroupId}`
    Expected body: none
    Description: Delete the whole service group with the specified service group ID. Implicitly deletes all related elements as well
- ServiceMetadataInterface
  - `PUT /{ServiceGroupId}/services/{DocumentTypeId}`
    Expected body: A `ServiceMetadataType` object as specified by the SMP XSD
    Description: Define the AP endpoint URL etc. for a certain participant and a certain document type
  - `DELETE /{ServiceGroupId}/services/{DocumentTypeId}`
    Expected body: none
    Description: Delete all participant information for the specified document type

4.3.3 Web application specifics

A Java EE Filter is used for reverting changes to the SML, if the change to the SMP fails. If a HTTP status code different from 200 (OK) is returned from the REST interface, then the corresponding change made to the SML is reverted. This functionality is implemented in the class `at.peppol.smp.server.hook.PostRegistrationFilter`.
A Jersey Filter has been added for generating the XML-DSIG element which is part of the response from a GET on Service Metadata interface. This is implemented in the class `at.peppol.smp.server.util.SignatureFilter`.

4.4 Security considerations

As the SMP is publically available on HTTP port 80 and does not require a client certificate or anything the like it especially the modifying actions (HTTP PUT and DELETE) must be handled with special care to avoid man in the middle attacks. Even though HTTP BasicAuth is used this is not really added security, as the username and password are only Base64 encoded - which is easily decodable - and are therefore vulnerable to Man in the Middle attacks.
The recommended scenario is to additionally configure the SMP to run on HTTPS (port 443), and do the modifying actions only via HTTPS. BasicAuth is required anyway but the data is not readable by third-partys because of the underlying transport security. This is something that is currently technically not available but should be used as a convention when running an SMP with this implementation.
For a future release it may be of value when the modifying actions are presented with a separate path prefix (e.g. /secure) which can than easily be used to forward all HTTP request on /secure/* to HTTPS automatically.

5 Client library

5.1 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!

5.2 Using the library

The `at.peppol.smp.client.SMPServiceCaller` is the main class when using the library. The class contains methods for reading, saving and deleting both service groups and service metadata, as well as listing the service groups of a given user. The library contains both static and non-static methods for performing all of these actions. The class is documented using JavaDoc.

The following is an example code of getting a service metadata object of a service group (participant) for a certain document type:

```
1  // ServiceGroup == ParticipantIdentifier
2  final ParticipantIdentifierType aServiceGroupID =
   SimpleParticipantIdentifier.createWithDefaultScheme ("0088:5798000000001");
3  // Document type identifier from enumeration
4  final DocumentIdentifierType aDocumentTypeID =
   EPredefinedDocumentIdentifier.urn_oasis_names_specification_ubl_schema_xsd_Invoice_2__
   Invoice__urn_www_cenbii_eu_transaction_biicoretrdm010_ver1_0__urn_www_peppol_eu_bis_pe
   ppol4a_ver1_0__2_0.getAsDocumentIdentifier ();
5  // Main call to the SMP client with the correct SML to use
```

```
6  final SignedServiceMetadataType aMetadata =
   SMPServiceCaller.getServiceRegistrationByDNS (ESML.DEVELOPMENT_LOCAL, aServiceGroupID,
   aDocumentTypeID);
```