



Plataforma de Interoperabilidad

Tutorial para Consumir servicios de la PGE sobre Plataforma Java

Control de Cambios

Fecha	Versión	Descripción	Autor	Aprobado Por
2015	1.0	Versión inicial		

Nombre actual del archivo: AGESIC-Plataforma-Tutorial-Java-v01-00.odt



Plataforma de Interoperabilidad

Este documento ha sido elaborado por AGESIC (Agencia para el Desarrollo del Gobierno de Gestión Electrónica y la Sociedad de la Información y el Conocimiento)

Usted es libre de copiar, distribuir, comunicar y difundir públicamente este documento así como hacer obras derivadas, siempre y cuando tengan en cuenta citar la obra de forma específica y no utilizar esta obra para fines comerciales. Toda obra derivada de esta deberá ser generada con estas mismas condiciones.



Índice de contenido

<u>1 - Introducción.....</u>	<u>4</u>
<u>1.1 - Objetivo.....</u>	<u>4</u>
<u>1.2 - Prerrequisitos.....</u>	<u>4</u>
<u>1.3 - Requerimientos del software.....</u>	<u>4</u>
<u>1.4 - Preparación del ambiente.....</u>	<u>4</u>
<u>1.4.1 - Sobrecribir bibliotecas nativas de Java.....</u>	<u>5</u>
<u>1.4.2 - Definir el JBoss AS Runtime.....</u>	<u>5</u>
<u>1.4.3 - Definir el JBossWS Runtime.....</u>	<u>7</u>
<u>2 - Descripción de un escenario.....</u>	<u>9</u>
<u>3 - Consumo de un servicio.....</u>	<u>11</u>
<u>3.1 - Descargar los materiales necesarios.....</u>	<u>11</u>
<u>3.2 - Crear proyecto Java Faceted.....</u>	<u>11</u>
<u>3.3 - Incluir Librerías y Otros Archivos Necesarios.....</u>	<u>13</u>
<u>3.4 - Obtención del token de Seguridad emitido por la PGE.....</u>	<u>13</u>
<u>3.5 - Invocación al Servicio.....</u>	<u>17</u>
<u>3.5.1 - Crear las clases para consumir el servicio.....</u>	<u>17</u>
<u>3.5.2 - Especificar en el mensaje SOAP el servicio y método a invocar.....</u>	<u>20</u>
<u>3.5.3 - Adjuntar en el mensaje SOAP el token SAML firmado por la PGE.....</u>	<u>21</u>
<u>3.5.4 - Adjuntar las propiedades necesarias para establecer la comunicación SSL.....</u>	<u>21</u>
<u>3.5.5 - Consumir el Servicio.....</u>	<u>22</u>
<u>3.5.6 - Probar el cliente programado.....</u>	<u>22</u>

1 Introducción

1.1 Objetivo

El objetivo de este tutorial es proveer una guía paso a paso para el desarrollo de un cliente stand-alone de la Plataforma de Gobierno Electrónico (PGE) sobre la plataforma Java para consumir un servicio web que ya se encuentra publicado, para lo cual se utilizará un ejemplo concreto.

1.2 Prerrequisitos

Se asume que el usuario conoce, a un nivel básico, las especificaciones WS-Security [1], WS-Trust [2] y SAML 1.1 [3]. Además, se asume que el usuario está familiarizado con el uso de certificados, keystores, aplicaciones JavaEE y servicios web.

Se debe descargar la carpeta materiales SFTP.

1.3 Requerimientos del software

La tabla 1 presenta las herramientas y productos de *software* requeridos para desarrollar y ejecutar la Aplicación Cliente. Si bien pueden usarse otras herramientas, y obtener el mismo resultado, en este documento se asumirá el uso de las mencionadas en la tabla.

Producto	Versión
Java Developer Kit (JDK)	6.0
JBoss Application Server	5.1
JBoss Web Services	3.2.2.GA
Eclipse	3.5 /Galileo
JBossWS Tools	3.1 GA
OpenSAML	2.3.1

Tabla 1 – Requerimientos de Software

1.4 Preparación del ambiente

A continuación se describe paso a paso la preparación del ambiente para el desarrollo del tutorial Java. Se asume que el lector ya ha descargado e instalado en su equipo el JRE, el entorno de desarrollo Eclipse y el servidor de aplicaciones JBoss AS según fue detallado en los requerimientos en la sección anterior.

La preparación del ambiente incluye las siguientes etapas:



1. Reemplazar bibliotecas nativas de Java.
2. Definir el JBoss AS Runtime.
3. Definir el JBossWS Runtime.

1.4.1 Sobrecribir bibliotecas nativas de Java

La versión 6 de Java provee una implementación nativa para Web Services que no es compatible con la tecnología JBoss utilizada para desarrollar el cliente en este tutorial. Por lo tanto, es necesario reemplazar esta biblioteca utilizando los mecanismos tradicionales que provee Java:

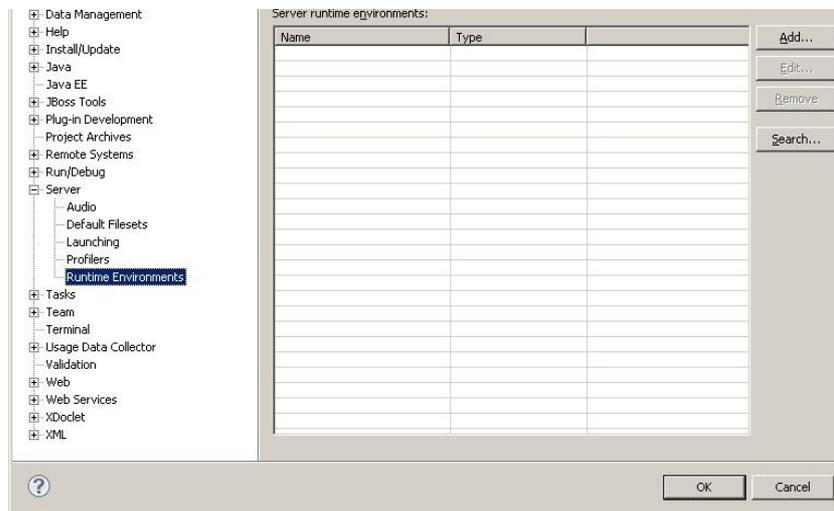
1. Para ello, se debe copiar los archivos de la carpeta `endorsed` del archivo de materiales descargado a la carpeta `<JRE_HOME>\lib\endorsed`.
2. En caso de no existir dicha carpeta debe crearla.

Nota: `<JRE_HOME>` debe ser reemplazado por la ruta completa a la carpeta donde está instalada el Java Runtime Environment (JRE).

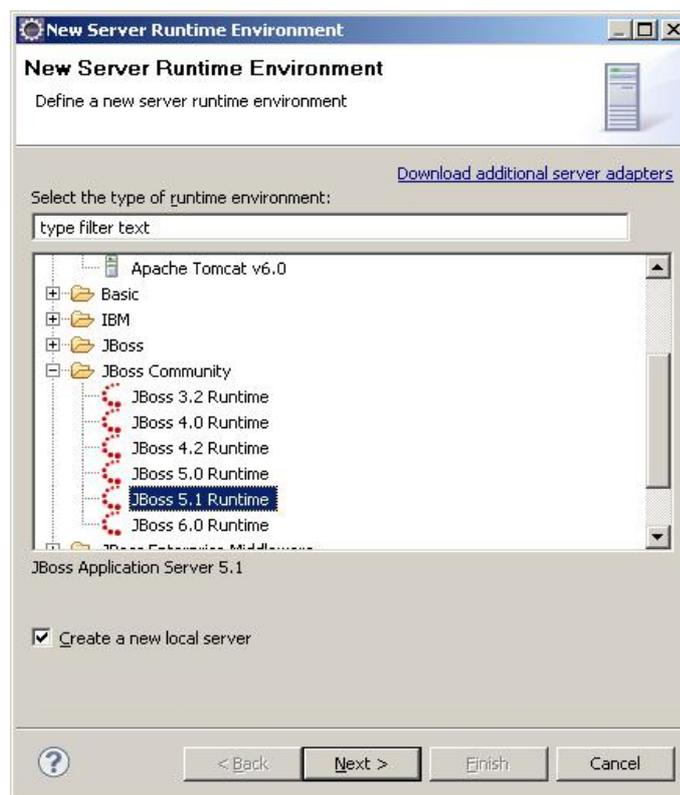
1.4.2 Definir el JBoss AS Runtime

El segundo paso consiste en registrar el servidor de aplicaciones JBoss AS en el entorno de desarrollo Eclipse, de forma de poder crear aplicaciones asociadas a dicho servidor. Para ello, se debe seguir los siguientes pasos:

1. Seleccionar del menú de Eclipse la opción *Windows* → *Preferences*.
2. Seleccionar la opción *Server* → *Runtime Environments* como se muestra en la siguiente imagen:

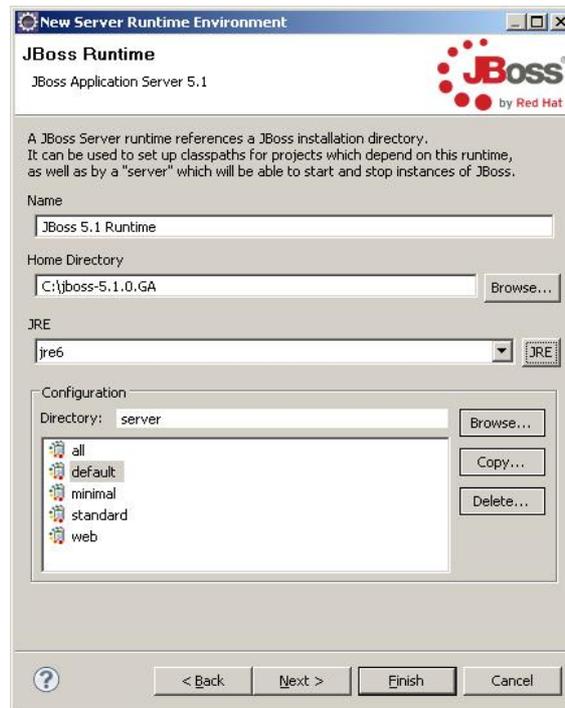


- Hacer clic en el botón Add..., luego la opción JBoss Community → JBoss 5.1 Runtime como se muestra en la figura siguiente y luego hacer clic en el botón Next.



- Completar los datos solicitados: en el campo *Name* ingresar un nombre descriptivo del servidor, por ejemplo, JBoss Tutorial
- en el campo *Home Directory* ingresar la ruta completa al directorio donde se encuentra instalado el servidor de aplicaciones JBoss AS
- en el campo *JRE* seleccionar la *Java Runtime Environment* deseada (debe ser 1.6 o superior para este tutorial) y luego en el campo *configuración* seleccionar la configuración "default".

7. Finalmente, seleccionar el botón *Finish*. Se debe alcanzar un resultado similar al de la figura 4.



1.4.3 Definir el JBossWS Runtime

El tercer y último paso es registrar el JBoss WS Runtime en el entorno de desarrollo Eclipse. Para esto, se deben completar los siguientes pasos:

1. Seleccionar del menú de Eclipse la opción *Windows* → *Preferences*.
2. En el menú izquierdo, seleccionar *Web Services* → *JBossWS Preferences* y presionar el botón *Add...*

2 Descripción de un escenario

La figura 1 presenta el escenario de ejemplo de consumo, en el cual intervienen dos organismos: el Banco de Previsión Social (BPS) que será el Organismo Cliente (quien consume el servicio) y AGESIC que será el Organismo Proveedor.

El MSP provee el servicio “Certificado de Nacidos Vivos”. Cuando se registró el servicio en la PGE, se desplegó un Servicio Proxy en ella para que las Aplicaciones Cliente accedieran al servicio a través de él. Además, mediante la configuración de políticas de control de acceso, el MSP autorizó a los usuarios con rol “doctor” de la sección “prestaciones” del BPS a consumir todos los métodos del servicio.

Por otro lado, en el BPS hay una Aplicación Cliente que está siendo utilizada por el usuario Pruebas que tiene el rol mencionado. La aplicación necesita acceder al servicio de AGESIC para lo cual, utilizando las credenciales del usuario Pruebas y a través de una Aplicación Emisora de Tokens interna al BPS, obtiene un *token* de seguridad SAML firmado por el BPS (pasos 1.a y 1.b).

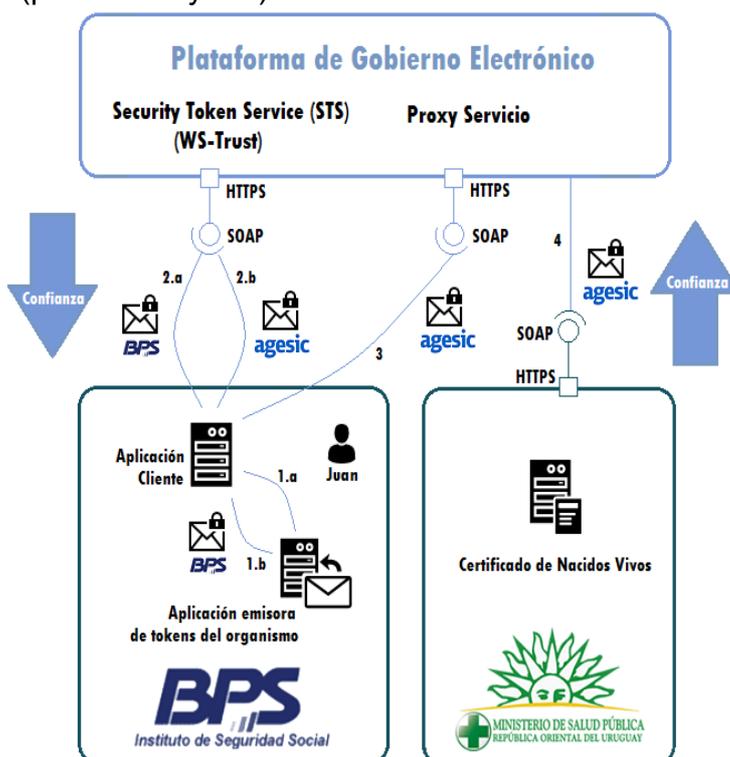


Figura 1: Escenario de uso

Luego con el *token* recibido obtiene del STS de la PGE, utilizando el estándar WS-Trust, otro *token* de seguridad firmado por la plataforma (pasos 2.a y 2.b). Para emitir este *token* la PGE verifica la firma digital del *token* enviado por la aplicación y la existencia del rol “ou=gerencia de proyectos,o=agesic”.

Por último, la Aplicación Cliente invoca al Servicio del MSP a través del Servicio Proxy de la PGE (los clientes nunca acceden al servicio final directamente, siempre lo hacen a



través del proxy creado en la PGE; existe un proxy específico para cada servicio disponible a través de la PGE). En la invocación se incluye el *token* firmado por la PGE y se especifican el servicio y el método a invocar.

3 Consumo de un servicio

En esta sección se describe, paso a paso, la implementación de una Aplicación Cliente Java para el consumo del servicio *Timestamp*.

La implementación del escenario comprende las siguientes etapas:

1. Obtener los materiales necesarios.
2. Crear proyecto Java Faceted
3. Obtención del *token* de Seguridad emitido por la PGE
4. Invocación del Servicio

En las siguientes subsecciones se describe en detalle cada una de ellas.

3.1 Descargar los materiales necesarios

Como primer paso se deben descargar los materiales necesarios del FTP de Agesic.

3.2 Crear proyecto Java Faceted

1. Seleccionar *File* → *New* → *Other* → *General* → *Faceted Project*, crear un nuevo proyecto con el nombre *Tutorial_PGE* y los facets *Java 6.0*, *JBoss Web Service Core 3.0* y *Dynamic Web Module 2.4* según las figuras 2 y 3.

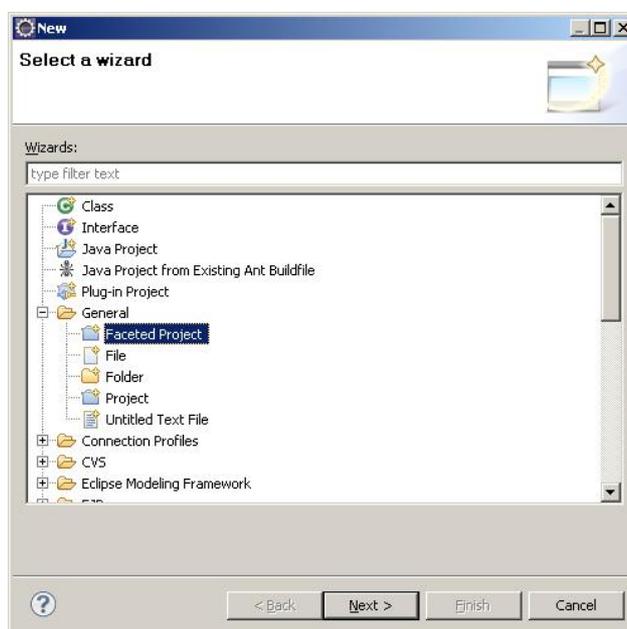


Figura 2: Creación de un proyecto Faceted

Nota: La aplicación Java que se está desarrollando no es una aplicación Web. Sin embargo, JBossWS Tools requiere que se utilice el faceted Web.

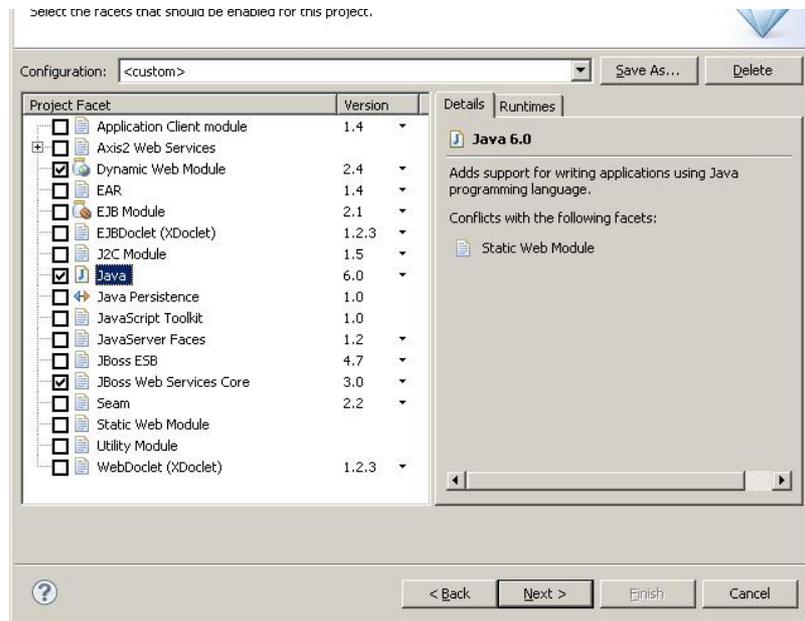


Figura 3: Selección de los facets

2. Configurar la carpeta destino del código fuente (src) y compilado (build), así como también el directorio de contenido Web.
3. Seleccionar el *JBossWS Runtime* como se ilustra en la figura 4 y presionar el botón *Finish*.

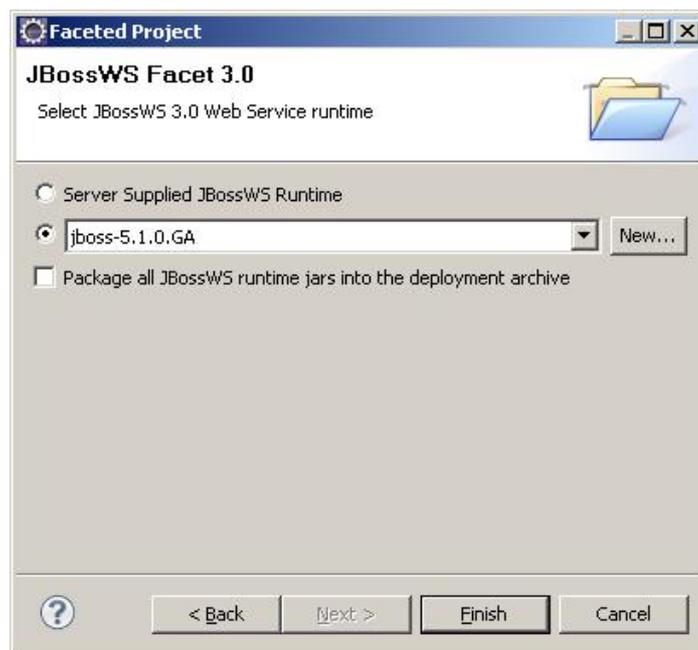


Figura 4: Configuración del JBossWS Runtime del proyecto

3.3 Incluir Librerías y Otros Archivos Necesarios

La Aplicación Cliente requiere las librerías de JBossWS y OpenSAML, así como la Librería PGEClient.jar implementada por AGESIC (versión 1.5 o posterior). A su vez, es necesario incluir el WSDL del servicio Timestamp. Para ello, se deben seguir los siguientes pasos:

1. Hacer clic derecho en el proyecto, seleccionar *New* → *Folder* y crear una carpeta llamada *lib*. Copiar en dicha carpeta **todos** los archivos que se encuentran en las carpetas *lib/agesic*, *lib/http-components*, *lib/jbossws* y *lib/saml* del archivo obtenido del FTP de AGESIC.
2. Agregar todas las bibliotecas copiadas en el paso anterior al Java Build Path del proyecto, haciendo clic derecho sobre el proyecto y luego seleccionado *Properties* → *Java Build Path* → *Libraries* → *Add JARs...*
3. Colocar la biblioteca JBossWS Runtime en el último lugar del classpath. Para ello, seleccionar la solapa *Order and Export*, seleccionar la biblioteca JBossWS Runtime y presionar el botón *Bottom* como se muestra en la figura 5.

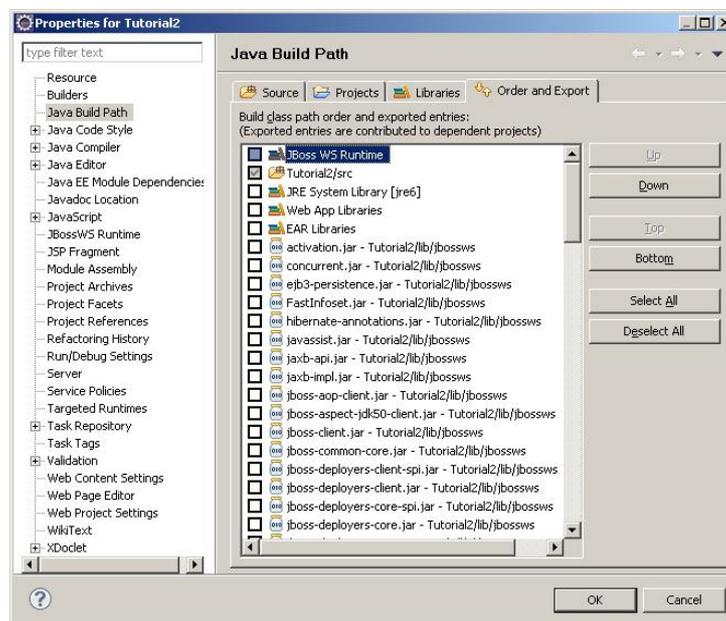


Figura 5 – Clase PGEClientTest

4. Crear una carpeta denominada *wSDL* y agregar todos los archivos de la carpeta *wSDL* del archivo obtenido del FTP de AGESIC.
5. Crear una carpeta denominada *keystores* y copiar todos los archivos creados en el tutorial *Certificados_Java*.

3.4 Obtención del token de Seguridad emitido por la PGE

Para realizar esta tarea, se utiliza el adaptador PGEClient.jar desarrollado por AGESIC. Los pasos a seguir son los siguientes:

1. Crear el package *test*. Para ello, seleccionar en el proyecto y luego clic derecho → new → package.
2. Crear la clase *PGEClientTest* en el package *test* de forma tal que contenga un método public static void *main(String[] args)* como se presenta en la figura 6.

```
package test;

public class PGEClientTest {
    public static void main(String[] args){
        //Aqui se pondra el codigo para invocar el servicio
    }
}
```

Figura 6 – Clase PGEClientTest

3. Importar las clases a utilizar como se muestra en la figura 7.

```
package test;

import uy.gub.agesic.beans.RSTBean;
import uy.gub.agesic.beans.SAMLAssertion;
import uy.gub.agesic.beans.StoreBean;
import uy.gub.agesic.exceptions.RequestSecurityTokenException;
import uy.gub.agesic.sts.client.PGEClient;

public class PGEClientTest {
    public static void main(String[] args){
        //Aqui se pondra el codigo para invocar el servicio
    }
}
```

Figura 7 – Importar las clases requeridas

4. Colocar en el método *main* el código que se muestra en la figura 8. Este código crea un RSTBean especificando los datos para enviar el pedido al STS de la PGE. En el pedido se carga la información relativa al usuario, organismo, su rol dentro del organismo, la dirección lógica del servicio que se desea consumir y el tipo de política de emisión de token. Por último, se especifica la dirección del STS, la cual, en el ambiente de testing, es siempre <https://testservicios.pge.red.uy:6051/TrustServer/SecurityTokenServiceProtected>.

```
String service = "http://testservicios.pge.red.uy/timestamp";
String policyName = "urn:tokensimple";
String issuer = "{{coloque nombre organismo aquí, ejemplo: BPS}}";

//Crear un bean con la información para generar el token SAML
RSTBean bean = new RSTBean();
bean.setUsername(userName);
bean.setRole(role);
bean.setService(service);
bean.setPolicyName(policyName);
bean.setIssuer(issuer);

//Definir la url del STS para obtener el token SAML
String stsUrl =
"https://testservicios.pge.red.uy:6051/TrustServer/SecurityTokenServiceProtected";
```

Figura 8 – Clase PGIClientTest

5. Crear tres *StoreBeans*, como muestra la figura 9, para almacenar los datos de acceso a los almacenes de claves que contienen los certificados y claves requeridas (dos keystores, y un trustore). Las rutas que se especifican en las variables *keyStoreFilePath* y *trustStoreFilePath* y deben apuntar al keystore y trustore, respectivamente, que se encuentran ubicados en la carpeta keystores recientemente creada (son los archivos creados en el tutorial *Certificados_Java*).

Nota: para consumir un servicio en el ambiente de producción será necesario contar con dos almacenes de certificados digitales:

Uno para establecer la comunicación mediante SSL, que debe contener un certificado proporcionado por AGESIC específicamente para el Organismo Cliente

Otro para identificar al Organismo Cliente para lo cual debe contener el certificado emitido por El Correo para el propio Organismo Cliente.

El primero contiene el certificado para establecer una comunicación segura vía SSL con la Plataforma, mientras que el segundo contiene un certificado de Persona Jurídica necesario para firmar las transacciones sobre la Plataforma.

En el ambiente de testing, se permite utilizar el mismo certificado digital en ambos casos, el cual es proporcionado por AGESIC y al igual que el truststore se encuentra en el archivo descargado del SFTP de AGESIC.

```
String alias = "010201823ca559/ce5953188d1628ae_be45a113-4156-4128-8332-77080b0c1c08"; //Colocar alias que usó en el tutorial de certificados Java, ej: tutorial.agesic.red.uy

String keyStoreSSLFilePath="keystores\\agesictesting_v5.keystore";
String keyStoreSSLPwd = "agesic"; //password del keystore

String keyStoreOrgFilePath="keystores\\agesictesting_v5.keystore";
String keyStoreOrgPwd="agesic"; //password del keystore

String trustStoreFilePath="keystores\\agesictesting_v3.truststore";
String trustStorePwd="agesic"; //password del truststore

StoreBean keyStoreSSL = new StoreBean();
keyStoreSSL.setAlias(alias);
keyStoreSSL.setStoreFilePath(keyStoreSSLFilePath);
keyStoreSSL.setStorePwd(keyStoreSSLPwd);

//En el ambiente de testing se podría usar el mismo bean anterior, en producción es necesario crear otro, apuntando al keystore del organismo
StoreBean keyStoreOrg = new StoreBean();
keyStoreOrg.setAlias(alias);
keyStoreOrg.setStoreFilePath(keyStoreOrgFilePath);
keyStoreOrg.setStorePwd(keyStoreOrgPwd);

//El truststore no requiere alias
StoreBean trustStore = new StoreBean();
trustStore.setStoreFilePath(trustStoreFilePath);
trustStore.setStorePwd(trustStorePwd);
```

Figura 9 – Keystore y Truststore

6. Por último, crear una instancia de la clase PGEClient e invocar el método requestSecurityToken para obtener el *token* SAML firmado por la PGE, como se muestra en la figura 10.

```
PGEClient client = new PGEClient();
SAMLAssertion assertionResponse = null;
try {
    //Solicitar el token SAML pasando los datos de
    //autenticación, los tres beans con los keystores,
    //y la URL del STS
    assertionResponse = client.requestSecurityToken(bean,
        keyStoreSSL, keyStoreOrg, trustStore, stsUrl);
    String stringRepresentation= assertionResponse.toString();
    System.out.println(stringRepresentation);
} catch (Exception e) {
    e.printStackTrace();
    System.exit(1);
}
```

Figura 10 – Obtención del token SAML firmado por la PGE

7. Ejecutar el programa desarrollado hasta ahora, para verificar que el token puede

ser obtenido correctamente. Para hacerlo, seleccionar la clase PGEClientTest, hacer clic derecho y luego seleccionar *Run as* → *Java application*.

8. En caso de ejecutarse correctamente, se desplegará en consola un token SAML.

Nota importante: asegúrese que la hora de su PC se encuentra sincronizada con la hora nacional (de igual manera que los servidores de AGESIC se encuentran sincronizados con ella). Si la hora no coincide, ocurrirá un error en la ejecución ya que la PGE considerará que los mensajes intercambiados no son válidos (estarán vencidos o fechados en el futuro, siendo ambas situaciones invalidantes para obtener el token). En todo caso, si obtiene un error al obtener el token, pruebe ajustando la hora unos minutos antes o después de la hora nacional.

3.5 Invocación al Servicio

Una vez obtenido un *token* SAML firmado por la PGE, es posible consumir el servicio. Para ello, se envía un mensaje SOAP al Servicio Proxy del servicio TimeStamp, el cual debe incluir lo siguiente:

- Servicio y método a invocar (especificados a través de WS-Addressing)
- *Token* SAML firmado por la PGE (incluido a través de WS-Security)
- Información de negocio según el WSDL del servicio (datos a enviar como parámetros de la invocación).

Además, se deben configurar las propiedades para establecer una comunicación segura mediante SSL. En este ejemplo, la invocación al servicio consta de cuatro pasos:

1. Crear las clases para consumir el servicio a partir del WSDL que lo describe. A través de estas clases se creará el mensaje SOAP con la información de negocio.
2. Especificar en el mensaje SOAP el servicio y método a invocar.
3. Adjuntar al mensaje SOAP el *token* SAML firmado por la PGE obtenido en el paso anterior.
4. Configurar las propiedades necesarias para establecer una comunicación SSL.
5. Consumir el servicio.

3.5.1 Crear las clases para consumir el servicio

Para esta tarea se utiliza la herramienta de generación de clientes de Web Services provista por el entorno de desarrollo Eclipse. Los pasos a seguir son los siguientes:

1. Hacer clic derecho en el archivo TimestampServicewsdl.wsdl ubicado en la carpeta wsdl y seleccionar *Web Service* → *Generate Client* como se muestra en la figura 11.

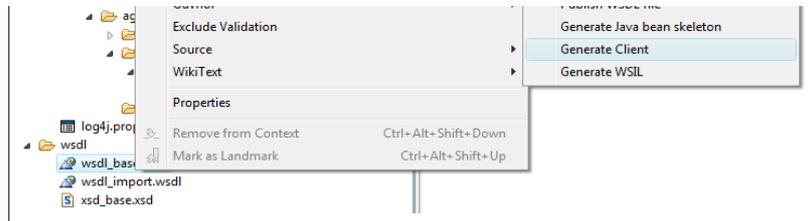


Figura 11 – Generar Clases para Consumir Web Service

2. Seleccionar *JBossWS* como *Web Service Runtime* y seleccionar el nivel de generación del cliente como “*Develop Client*”, según se muestra en la figura 12.

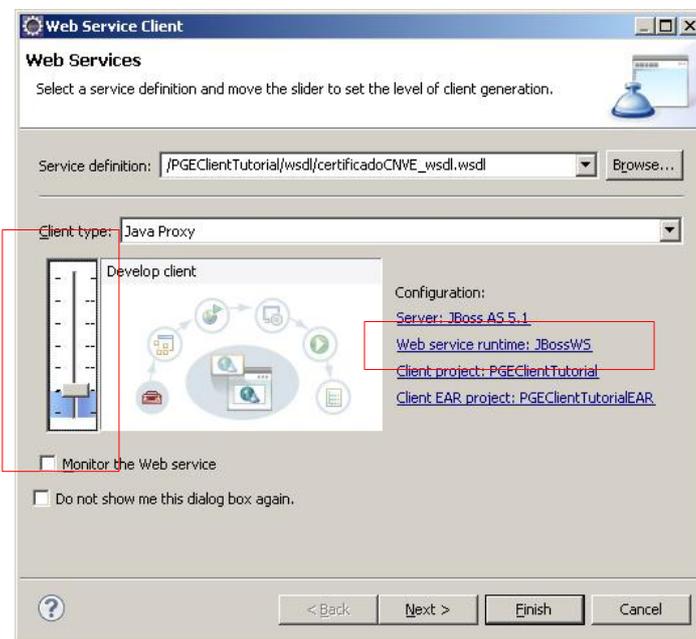


Figura 12 – Generar Clases para Consumir Web Service (parte 2)

3. Presionar el botón “*Next*” y si se desea, modificar el nombre del paquete donde se colocarán las clases generadas. La figura 13 ilustra el campo donde colocar el nombre del package.

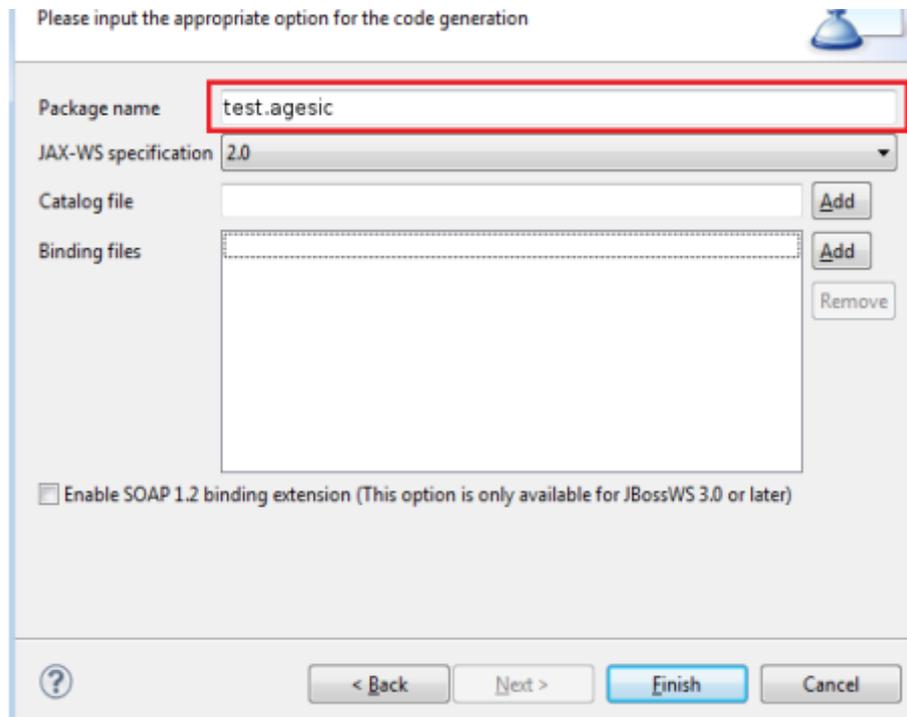


Figura 13 – Generar Clases para Consumir Web Service (parte 3)

Una vez generadas las clases se puede proceder a consumir el servicio. Una vez creada la instancia del servicio a invocar (*timestampService*), se debe obtener el puerto (port), sobre el cual se podrán invocar las operaciones del servicio; cada operación estará representada por un método en el puerto. Sobre éste, se aplicarán las propiedades de WS-Addressing, WS-Security y SSL tal como se ilustrará más adelante.

En la figura 14 se ilustran las líneas de código necesarias para construir el puerto para invocar a el servicio.

```
TimestampService_Service timestampService = new
TimestampService_Service();
    TimestampService port = timestampService
        .getTimestampServiceImplPort();
```

Figura 14 – Creación de puerto para invocar al servicio

La figura 15 ilustra qué clases importar.

```
import test.agesic.TimestampService;
import test.agesic.TimestampService_Service;
```

Figura 15 – Creación de puerto para invocar al servicio

3.5.2 Especificar en el mensaje SOAP el servicio y método a invocar.

Como se mencionó anteriormente, la PGE requiere que en la invocación al servicio se especifique el servicio y método a invocar. Para esto, se utilizan los cabezales de WS-Addressing “To” y “Action”, respectivamente. La figura 16 muestra cómo especificar esta información utilizando los cabezales WS-Addressing requeridos por la PGE. El valor aplicable al campo “to” debe ser provista por AGESIC, mientras que el valor aplicable al campo “action” puede obtenerse a partir del WSDL.

```
//Propiedades para WS-Addressing
AddressingBuilder addrBuilder =
    SOAPAddressingBuilder.getAddressingBuilder();
SOAPAddressingProperties addrProps =
    (SOAPAddressingProperties)addrBuilder.newAddressingProperties();
String actionStr =

"http://www.agesic.gub.uy/soa/TimestampService/TimestampServiceImplPort/GetTimestamp";

addrProps.setTo(new AttributedURIImpl(service));
addrProps.setAction(new AttributedURIImpl(actionStr));

BindingProvider bindingProvider = (BindingProvider)port;
Map<String, Object> reqContext = bindingProvider.getRequestContext();
reqContext.put(JAXWSConstants.CLIENT_ADDRESSING_PROPERTIES, addrProps);

//Construir la cadena de handlers, en el orden especificado
List<Handler> customHandlerChain = new ArrayList<Handler>();
customHandlerChain.add(new WSAddressingClientHandler());
customHandlerChain.add(new WSSecurityHandlerServer());
```

Figura 16 – Agregar los cabezales WS-Addressing al mensaje

La figura 17 muestra como importar las clases a utilizar.

Figura 17 – Clases a importar para configurar WS-Addressing

```
import java.util.ArrayList;
import java.util.List;
import java.util.Map;
import javax.xml.datatype.XMLGregorianCalendar;

import javax.xml.ws.BindingProvider;
import javax.xml.ws.addressing.AddressingBuilder;
import javax.xml.ws.addressing.AttributedURI;
import javax.xml.ws.addressing.JAXWSConstants;
import javax.xml.ws.addressing.soap.SOAPAddressingBuilder;
import javax.xml.ws.addressing.soap.SOAPAddressingProperties;
import javax.xml.ws.handler.Handler;

import org.jboss.ws.core.StubExt;
import org.jboss.ws.extensions.addressing.AttributedURIImpl;
import org.jboss.ws.extensions.addressing.jaxws.WSAddressingClientHandler;
import org.jboss.ws.extensions.security.jaxws.WSSecurityHandlerServer;
```

3.5.3 Adjuntar en el mensaje SOAP el token SAML firmado por la PGE

Para adjuntar el token SAML utilizando WS-Security se procede de forma similar que para adjuntar los cabezales WS-Addressing. Sin embargo, en este caso AGESIC provee un handler específico (SAMLHandler) para adjuntar el token SAML al mensaje, dado que la plataforma JBoss no provee ninguno prefabricado. La figura 18 presenta cómo utilizar este mecanismo para adjuntar el token SAML requerido por la PGE.

```
//Esto debe colocarse después de los handlers ya configurados
//(WSAddressingClientHandler y WSSecurityHandlerServer), justo antes
//de bindingProvider.getBinding().setHandlerChain(customHandlerChain);
customHandlerChain.add(new SAMLHandler());

bindingProvider.getBinding().setHandlerChain(customHandlerChain);

//Y esto debe colocarse justo debajo de la línea
//reqContext.put(JAXWSConstants.CLIENT_ADDRESSING_PROPERTIES, addrProps);
reqContext.put(AgesicConstants.SAML1_PROPERTY, assertionResponse);
```

Figura 18 – Agregar token SAML al mensaje usando WS-Security

También se deben importar las clases a usar como se presenta en la figura 19.

```
import uy.gub.agesic.AgesicConstants;
import uy.gub.agesic.jbossws.SAMLHandler;
```

Figura 19 – Importar las clases necesarias para WS-Security

3.5.4 Adjuntar las propiedades necesarias para establecer la comunicación SSL

Para que la invocación al servicio pueda efectuarse a través de SSL deberán configurarse ciertas propiedades en el contexto de la invocación. Estas propiedades harán referencia a los almacenes de claves que se utilizaron para configurar la invocación al STS. En la figura 20 se ilustran las sentencias de código necesarias. Cabe mencionar que para el caso del ejemplo, se utiliza el mismo archivo de keystore que se usó anteriormente (de Organismo) para efectuar la comunicación SSL (esto es válido solo en el ambiente de testing, no así en el ambiente de producción ya que deben utilizarse certificados digitales diferentes, uno emitido por AGESIC y otro por El Correo).

```
//reqContext.put(JAXWSConstants.CLIENT_ADDRESSING_PROPERTIES, addrProps);
reqContext.put(StubExt.PROPERTY_AUTH_TYPE, StubExt.PROPERTY_AUTH_TYPE_WSSE);
reqContext.put(StubExt.PROPERTY_KEY_STORE, keyStoreSSLFilePath);
reqContext.put(StubExt.PROPERTY_KEY_STORE_PASSWORD, keyStoreSSLPwd);
reqContext.put(StubExt.PROPERTY_TRUST_STORE, trustStoreFilePath);
reqContext.put(StubExt.PROPERTY_TRUST_STORE_PASSWORD, trustStorePwd);

//Nota: lo anterior puede ser sustituido también por el siguiente código:
System.setProperty("javax.net.ssl.keyStore",
    keyStoreSSLFilePath);
System.setProperty("javax.net.ssl.keyStorePassword",
    sslKeyStore.getStorePwd());
System.setProperty("javax.net.ssl.trustStore",
    sslTrustStore.getStoreFilePath());
System.setProperty("javax.net.ssl.trustStorePassword",
    sslTrustStore.getStorePwd());
```

Figura 20 – Configuración de propiedades para establecer la comunicación SSL

3.5.5 Consumir el Servicio

Por último, se debe invocar el servicio. Para ello se debe agregar el código de la siguiente figura e importar las clases a utilizar como se presenta a continuación.

```
//Crear los parámetros de entrada
//Estas clases fueron generadas en el paso 1, cuando se importó el WSDL
XMLGregorianCalendar timestamp = port.getTimestamp();
System.out.println("Timestamp PDI: " + timestamp.toString());
```

3.5.6 Probar el cliente programado

Para ejecutar el cliente implementado, seleccionar la clase PGEClientTest, hacer clic derecho y ejecutar *Run as* → *Java Application*. En el *Apéndice 1* se puede ver el código fuente completo.