MANUAL DE USUARIO
FIRMA Y ACTUALIZACIÓN EN UN PASO

Izenpe

Beato Tomás de Zumárraga
71 - 1ª Planta
01008
Vitoria - Gasteiz

www.izenpe.com
info@izenpe.com
Tel.: 945 017 490

Contenido

1.	Introducción	3
1.1.	Objetivo	3
1.2.	Audiencia	3
2.	Descripción general	4
3.	Kit de bienvenida signupdateKit en Java	5
3.1.	Objetivo	5
3.2.	Requisitos	5
3.3.	Dependencias	5
3.4.	Instalación	6
3.5.	Configuración	6
3.5.1.	Cabecera WS-Security	9
3.5.2.	Plantilla para firma PAdES	9
3.6.	Ejecución	10
3.6.1.	Objeto *SignupdateResponse*	11

1. Introducción

1.1. Objetivo

Explicar al usuario final cómo utilizar el nuevo servicio de **Firma y actualización en un paso** implementado en el Smartgateway de Zain. Todo ello, de la manera más concisa y clara posible, a fin de que la comprensión sea adecuada, facilitando el uso y manejo posterior de este nuevo servicio.

1.2. Audiencia

Este manual está dirigido a todos los usuarios finales que necesiten realizar llamadas al nuevo servicio de **Firma y actualización en un paso**, a través del pipeline del Smartgateway de Zain.

2. Descripción general

El servicio de **Firma y actualización en un paso** realiza, de manera transparente y con una única llamada a la plataforma Zain, firma de documentos a formato AdES- XL (con política de firma, opcionalmente, para el caso de las firmas con perfil XAdES).

Para poder utilizar este nuevo servicio y facilitar la integración y manejo, se ha creado la siguiente funcionalidad:

- Librería Java para simplificar el desarrollo de las aplicaciones cliente sin adentrarse en la complejidad de las llamadas a servicios Web.

Gracias a que las llamadas al servicio se realizan a través de la plataforma Zain, el nivel de seguridad global se incrementa, ya que sólo existe un único punto de entrada, además de que es el propio Zain el encargado de validar la autenticación y autorización del usuario final para consumir el nuevo servicio expuesto.

3. Kit de bienvenida signupdateKit en Java

3.1. Objetivo

El objetivo de este apartado es instruir en el uso de la API (*singupdateClient.jar*) de Java para el uso del servicio **Firma y actualización en un paso**, a través de ejemplos representativos.

3.2. Requisitos

La librería *singupdateClient* es una API desarrollada sobre Axis para crear aplicaciones cliente. Permite generar aplicaciones Java muy sencillas con muy pocas líneas de código.

Para facilitar el uso de la API *singupdateClient*, se ha desarrollado un kit de bienvenida (*signupdateKit*), con el objetivo de ponerla en funcionamiento fácilmente. Este kit de bienvenida se distribuye en un fichero comprimido con el nombre *signupdateKit.zip*.

Para poner en marcha este kit de bienvenida será necesario cumplir los siguientes requisitos:

- El sistema deberá tener instalado Java 1.6.
- El sistema deberá tener instalado el IDE de desarrollo Eclipse.

3.3. Dependencias

La librería *singupdateClient.jar* está desarrollada con la tecnología Axis, por lo que será necesario incluir las siguientes dependencias en el classpath del proyecto. Estas dependencias se encuentran dentro del directorio */lib* del kit de bienvenida de ejemplo. Son las siguientes:

- activation.jar
- axis.jar
- commons-discovery-0.2.jar
- commons-httpclient-3.0.1.jar
- commons-logging-1.0.4.jar
- jaxrpc.jar
- log4j-1.2.8.jar
- saaj.jar

- trustedx-client-axis-stub.jar
- trustedx-provider.jar
- wsdl4j-1.5.1.jar
- wss4j-1.5.10.jar
- xalan-2.7.0.jar
- xercesImpl-2.8.1.jar
- xml-apis.jar
- xmlsec-1.4.4.jar

3.4. Instalación

Para instalar el kit de bienvenida *signupdateKit* se deberán seguir los siguientes pasos:

- Abrir el Eclipse y en la zona de proyectos pulsar el botón derecho y seleccionar la opción Importar proyecto
- Seleccionar la opción *General -> Existing Proyect into Workspace* y pulsar el botón *Next*.
- Seleccionar la opción *Select archive file* y pulsar el botón *Browse…* para poder seleccionar el fichero comprimido que contiene el kit de bienvenida *signupdateKit* que se distribuye en el fichero comprimido *signupdateKit.zip* y pulsar el botón *Finish*.

3.5. Configuración

Para el correcto funcionamiento del kit de bienvenida se deberá realizar previamente a la ejecución de sus ejemplos, una pequeña configuración para determinar una serie de parámetros de la parte cliente de servicios Web.

Esta configuración se realiza a través de un fichero de propiedades (*smartwrapper.properties*) que deberá encontrarse en el classpath del proyecto. En caso de tener un kit de bienvenida anterior, se podrán reaprovechar las mismas propiedades del fichero mencionado.

Este fichero de propiedades deberá contener los siguientes valores:

- **Keystore.active:** dependiendo el valor se actúa de una forma u otra (por defecto, false):
  - **true**: para la conexión HTTPS se utilizará el almacén de claves definido en el parámetro *Keystore.path*.
  - **false**: se utilizará el almacén de claves configurado en la máquina virtual de Java (si lo hay).

- **Keystore.path:** directorio donde se encuentra el almacén de claves que se utilizará en la conexión HTTPS. Sólo para SSL Mutuo.

- **Keystore.password:** contraseña del almacén de claves definido en Keystore.path. Sólo para SSL Mutuo.

- **Keystore.type:** tipo del almacén de claves definido en Keystore.path (PKCS12). Sólo para SSL Mutuo.

- **Truststore.active:** dependiendo el valor se actúa de una forma u otra (por defecto, false):

  - **true**: para la conexión HTTPS se utilizará el almacén de certificados definido en el parámetro *Truststore.path*.
  - **false**: se utilizará el almacén de certificados configurado en la máquina virtual de Java (si lo hay).

- **Truststore.path:** directorio donde se encuentra el almacén de certificados que se utilizará en la conexión HTTPS.

- **Truststore.password:** contraseña del almacén de certificados definido en Truststore.path.

- **Proxy.active:** indica si para la conexión HTTP se utilizará un servidor Proxy (true) o para indicar si no se utilizará un servidor Proxy (false).

- **Proxy.host**: servidor proxy utilizado (si el valor de Proxy.active es true).

- **Proxy.port:** puerto del servidor proxy.

- **Proxy.username:** nombre de usuario para acceder al servidor proxy (si requiere autenticación HTTP básica).

- **Proxy.password:** contraseña para acceder al servidor proxy (si requiere autenticación HTTP básica).

- **Timeout:** tiempo de espera de la conexión HTTPS (en milisegundos).

- **req-log.active:** indica si se guardarán en disco las peticiones SOAP de las llamadas al servicio.
- **req-log.savePath:** directorio donde se guardarán las diferentes peticiones SOAP.
- **res-log.active:** indica si se guardarán en disco las respuestas SOAP de las llamadas al servicio.
- **res-log.savePath:** directorio donde se guardarán las diferentes respuestas SOAP.
- **authN.policy**: indica la política de autenticación solicitada (opcional).

Con el objetivo de mantener la misma filosofía que los kits de bienvenida anteriores, se ha creado un objeto denominado *SignUpdateConfig* que permite configurar de manera dinámica todas las propiedades necesarias para el correcto funcionamiento de las llamadas al servicio. De esta forma, indicando la siguiente línea de código dentro del ejemplo, las propiedades que se utilizarán serán las del propio objeto *SignUpdateConfig*, en vez de las del fichero de propiedades *smartwrapper.properties*.

SignUpdateConfig.setCurrent(ZAIN_CONFIG);

Donde, el parámetro *ZAIN_CONFIG* es una variable estática del tipo *SignUpdateConfig,* con métodos *setter* para alimentar todos los valores necesarios para el buen funcionamiento.

NOTA: Un ejemplo completo de su uso se puede encontrar dentro del ejemplo *XadesEpesEnveloped*.

3.5.1. Cabecera WS-Security

El kit de bienvenida permite generar una cabecera WS-Security del tipo *UsernameToken* para poder invocar al servicio de **Firma y actualización en un paso**, en vez de realizarlo a través de SSL Mutuo.

Para ello, será necesario crear un objeto *UsernameTokenHeader*, tal y como se muestra en el siguiente ejemplo:

```
UsernameTokenHeader header = new UsernameTokenHeader();
header.setUsername("username");
header.setPassword("password");
```

NOTA 1: En caso de utilizar la cabecera WS-Security, no será necesario configurar las propiedades relativas al keystore (*keystore.path*, *keystore.password* y *keystore.type*), ya que no se hará uso de él.

NOTA 2: Un ejemplo completo de su uso se puede encontrar dentro del ejemplo *XadesEpesEnveloped*.

3.5.2. Plantilla para firma PAdES

Dentro del kit de bienvenida *signupdateKit* se suministra un ejemplo de plantilla para utilizar en los casos en que se realice una firma del perfil PAdES. Este ejemplo se encuentra en la carpeta */resources* y su nombre es *plantilla-pades.xml*.

Si se opta por crear una plantilla diferente, hay que tener en cuenta dos aspectos importantes y que son de ***carácter obligatorio*** para el correcto funcionamiento del servicio de **Firma y actualización en un paso**. Se indican a continuación:

1. En el nodo <*signatureAlg*> hay que indicar el valor *ETSI.CAdES.detached*.

2. El namespace que aparece en el nodo raíz (<*PdfSignatureInfo*>) tiene que llevar prefijo de manera obligatoria. En caso contrario, el servicio no interpreta correctamente la plantilla y el resultado final no será el correcto.

3.6. Ejecución

A continuación se detallan los pasos a seguir para la ejecución de los ejemplos del kit de bienvenida *signupdateKit*:

La API de integración *signupdateKit* ofrece la posibilidad de realizar operaciones con tres perfiles posibles de firma:

- CAdES Attached
- PAdES
- XAdES Enveloped y Detached (con/ sin política de firma)

> ***NOTA***: Todas las firmas devueltas por el servicio corresponden a una AdES- XL, a excepción de la PAdES, que, por definición del estándar devuelve una -A.

Para poder lanzar peticiones correctamente al nuevo servicio, dentro de cada uno de los ejemplos, habrá que indicar el valor de las siguientes constantes:

- **END_POINT**: url para poder invocar al servicio. En este caso:
    - SSL Mutuo: https://psf.izenpe.com:8443/trustedx-sgw/SignUpdateGateway
    - WS-Security: https://psf.izenpe.com:8080/trustedx-sgw/SignUpdateGateway
- **INPUT_PATH**: ruta del documento a firmar
- **OUTPUT_PATH**: ruta del documento firmado

Para el caso de una firma con un perfil PAdES, se puede asignar una plantilla específica. Para ello, habrá que indicar la siguiente constante:

- **TEMPLATE_PATH**: ruta de la plantilla a utilizar para la firma

Además, es necesario configurar, a través de los métodos *setter*, una serie de parámetros dentro del objeto *SignupdateRequestSender* para poder realizar las llamadas correctamente. Son los siguientes:

- Parámetros comunes y obligatorios:
    - **Endpoint**: se indicará el valor de la constante END_POINT
    - **SignatureProfile**: se indicará el perfil de firma que se va realizar
    - **Language**: idioma en el que va a realizar la petición
    - **SignatureDocumentBytes**: bytes del documento enviado a firmar
- Parámetros comunes y opcionales:
    - **ZainSigner**: indica un DN (Distinguish Name) alternativo al del firmante que se indica en la propiedad *Keystore.path*.

- **UsernameTokenHeader**: contiene el nombre de usuario y la contraseña de la cabecera WS-Security. Utilizarlo sólo si la comunicación contra el servicio no va a ser mediante SSL Mutuo.

- Parámetros exclusivos de las firmas XAdES Detached:

  - **SignaturePlacement**: indica la ubicación de la firma (obligatorio)
  - **SignPropertyPolicy**: indica la política de firma que se va a aplicar (opcional)
  - **InputRefUri**: identificador para poder enlazar la firma con el documento original. Si no se indica nada, el valor por defecto será:

    urn:detached

- Parámetros exclusivos de las firmas XAdES Enveloped:

  - **XmlEnvelopedXPathPosition**: nombre del nodo xml donde irá ubicada la firma dentro del documento XML a firmar (obligatorio)
  - **SignPropertyPolicy**: indica la política de firma que se va a aplicar (opcional)

- Parámetros exclusivos de las firmas PAdES:

  - **PdfTemplate**: ruta de la plantilla que se va a utilizar para la firma (opcional)

3.6.1. Objeto *SignupdateResponse*

El objeto *SignupdateResponse* permite recuperar la información más relevante de la respuesta recibida por el servicio de **Firma y actualización en un paso**. Los métodos *getters* que posee se enumeran a continuación:

- **String getResultMajor()**: contiene el resultado mayor (señala situaciones de error en el protocolo de infraestructura).

- **String getResultMinor()**:contiene el resultado menor (detalla el error reportado por el resultado mayor).

- **String getResultMessage()**:contiene el mensaje genérico (describe el protocolo de servicio utilizado).

- **byte[] getSignature()**: devuelve el documento con la(s) firma(s). En caso de una petición al servicio para firma Detached, devuelve la propia firma.

- **Document[]getCertificates()**: devuelve una array con todos los certificados involucrados en la firma (1 - n).

- **Date getTimestampDate()**: devuelve la hora de la TSA, es decir, la hora en la que se ha realizado el sellado de tiempo.

- **OptionalOutputs getOptionalOutputs()**: permite acceder al objeto interno para poder recuperar otros elementos que mediante los métodos de la librería *singupdateClient* no se pueden. Aporta flexibilidad a las necesidades de cada cliente.