Manual de integración

Port@firmas v3.1.1

Versión: v01r01

Fecha: 03/06/2019

HOJA DE CONTROL

Título	Manual de integración		
Entregable	Port@firmas v3.1.1		
Nombre del Fichero	PF311-PRO-Manual_de_integración_v01r01.odt		
Autor			
Versión/Edición	v01r01	**Fecha Versión**	03/06/2019
Aprobado por		**Fecha Aprobación**	03/06/2019
		Nº Total Páginas	53

REGISTRO DE CAMBIOS

Versión	Causa del Cambio	Responsable del Cambio	Área	Fecha del Cambio
v01r00	Creación del documento		CHIE	14/03/2019
v01r01	Revisión del documento		CHIE	03/06/2019

CONTROL DE DISTRIBUCIÓN

Nombre y Apellidos	Cargo	Área	Nº Copias
Manuel Perera Domínguez	Jefe de Servicio	CHIE	1
Francisco Mesa Villalba	Director de Proyecto	CHIE	1

ÍNDICE

1 INTRODUCCIÓN

El objetivo del presente documento es servir de guía en el uso de los servicios web de la aplicación Port@firmas v3.1.1 por parte de los programadores que vayan a desarrollar aplicaciones clientes que hagan uso de los mismos.

2 SERVICIOS WEB PORT@FIRMAS

Port@firmas dispone de dos fachadas Web Service para atacar a las API's desde terceras aplicaciones. Dichas fachadas son la de los servicios de Port@firmas v2 con una funcionalidad ampliada y una fachada para compatibilidad con los servicios de Port@firmas v1.

Cada fachada contiene varios servicios que serán descritos:

- Fachada Web Services v1
  - Servicio de envío de peticiones de firma.
- Fachada Web Services v2
  - Servicio de consulta.
  - Servicio de modificación.

Para poder hacer uso de las nuevas funcionalidades de Port@firmas v3 es necesario hacer uso de la fachada de servicios de v2, que irá evolucionando con cada nueva funcionalidad que se añada a la aplicación. La fachada de servicios de v1 no ofrece nuevas funcionalidades ni evolucionará dado que su función es únicamente ofrecer retrocompatibilidad con versiones anteriores de Port@firmas. La fachada que ofrecía la funcionalidad de firma remota se ha eliminado definitivamente.

Los servicios v1 serán discontinuados en futuras versiones de la herramienta y no se garantiza su futura presencia.

2.1 Nuevas funcionalidades

Se ha eliminado el servicio de creación de una petición a partir de una firma que se mantenía por compatibilidad con versiones previas (el servicio se llama "createRequestFromSign"). Se ha eliminado también la funcionalidad de la firma remota en la v1.

En el caso de que no se utilice este servicio los usuarios que estén realizando una migración de una v3.0.X a la v3.1.1 y que utilicen los servicios de la v2 no tendrán que realizar modificación alguna para adaptarse a la v3.1.1.

2.2 Fachada Web Services v1

Tal como ya se ha indicado, esta fachada solo se ofrece por motivos de compatibilidad y debe evitarse su uso. Las aplicaciones que aún hagan uso de esta fachada v1 de servicios web deben adaptarse a la fachada de servicios v2.

El presente documento de integración se limitará a elencar los descriptores de la fachada pero no documentará su uso. Para obtener información detallada, se deberá consultar el manual de integración de las versiones previas de Port@firmas.

La fachada ofrece dos servicios, uno para el envío de peticiones de firma y otro para la firma remota de documentos para terceras aplicaciones.

2.2.1 Servicio de peticiones de firma

El WSDL descriptor del servicio se encuentra en la siguiente URL:

http://servidor:puerto/pfirmav3/services/PfServicioWS?wsdl

2.3 Fachada Web Services v2

La fachada ofrece dos servicios, uno para consulta y otro para modificación.

2.3.1 Servicio de consulta

Proporciona métodos de consulta de peticiones y datos asociados, usuarios y puestos de trabajo. El WSDL descriptor del servicio se encuentra en la siguiente URL:

http://servidor:puerto/pfirmav3.1.1/servicesv2/QueryService?wsdl

El javadoc de la interfaz webservice en java se puede encontrar dentro del fichero ***javadocWS.zip*** el cual se encuentra en el kit de integración en la carpteta “Kit de desarrollo”.

2.3.2 Servicio de modificación

Ofrece métodos para la creación, edición y envío de peticiones. El WSDL descriptor del servicio se encuentra en la siguiente URL:

http://servidor:puerto/pfirmav3.1.1/servicesv2/ModifyService?wsdl

El javadoc de la interfaz webservice en java se puede encontrar dentro del fichero ***javadocWS.zip*** el cual se encuentra en el kit de integración en la carpteta “Kit de desarrollo”.

2.3.3 Ejemplo de integración Web Services

Se incluye una aplicación de test llamada ***pfirma-test-ws-v2-3.1.1-jar-with-dependencies (***el cual se encuentra en el kit de integración en la carpteta “Kit de desarrollo”***)*** para ilustrar el uso de los servicios ofrecidos en la fachada web service v2. Se trata de una aplicación con estructura maven/eclipse y hace uso de unos clientes generados a través de CXF.

Es posible ejecutar la aplicación de pruebas de dos formas:

- Desde Eclipse. La aplicación contiene una clase main que se encarga de lanzar las suites de test JUnit definidos.
- Como una aplicación java (.jar).

2.3.3.1 Requisitos previos

Las especificaciones del servidor a desarrollar las pruebas deben ser las indicadas en el documento de instalación de Port@firmas v3.1.1.

En cuanto a los requisitos a nivel de equipo cliente, que tenga conexión contra el servidor donde está desplegada la aplicación y tener instalada una JRE 1.8 o superior.

2.3.3.2 Configuración del entorno de pruebas

La aplicación cliente de los servicios web contiene un fichero de propiedades que debe ser configurado para la correcta realización de las pruebas (*ws.properties*). El fichero se encuentra en la carpeta raíz.

```
## WS Adresses ##
ws.query.address=http://server:port/pfirmav3.1.1/servicesv2/QueryService?wsdl
ws.mofify.address=http://server:port/pfirmav3.1.1/servicesv2/ModifyService?wsdl

## Redaction data ##
redaction.user.id=12345678Z
redaction.docType=TD99
redaction.application=APPID
redaction.subject=Asunto Prueba WS
redaction.reference=Referencia Pruebas WS
redaction.text=Texto Pruebas WS

## Metadata ##
metadata.param1=PRIORIDAD
metadata.param2=IMPORTE

## Action ##
request.webAction=http\://www.google.com

## Signers ##
request.newUserSigner=11111111H

## Request ##
request.hash=f6eEHvpfZv
doc.hash=Z78KvdJQSo
```

```
## Security ##
security.enabled=true
security.pass=12345

## Protocol configuration ##
## HTTP, CMIS, ENIDOCWS
protocol=ENIDOCWS
## Para HTTP y CMIS
protocol.file.path=/pdf/
protocol.file.name=pdf-test.pdf

## ENIDOCWS
protocol.identifier=REPOS901APPUID7SgegNnqrC8m5ylX

## Local file
protocol.file.local=E:/pdf-test.pdf
## Algoritmo de hash
hash.algorithm=SHA1
## Numero de documentos a incluir en la peticion
request.num.documents=5
```

2.3.3.3 Parámetros del entorno de pruebas

A nivel de servidor no se requiere ningún requisito especial para Port@firmas. La aplicación de pruebas se lanzará a través de línea de comandos.

Las propiedades a editar en *ws.properties* son las siguientes:

ws.query.address	Url del servicio de consulta.
ws.mofify.address	Url del servicio de modificación.
redaction.user.id	Identificador de usuario que actuará como remitente y destinatario de las peticiones.
redaction.docType	Tipo de documento que se enviará en la petición.
redaction.application	Nombre de la aplicación en el sistema de Port@firmas a la cual pertenecerá la petición.

redaction.subject	Asunto de la petición.
redaction.reference	Referencia de la petición.
redaction.text	Texto de la petición
metadata.param1	Nombre del primer parámetro para la petición.
metadata.param2	Nombre del segundo parámetro para la petición.
request.webAction	URL de web que será invocada al ejecutar la acción de la petición.
request.newUserSigner	Nuevo firmante (usuario) para cambios de firmantes en peticiones.
request.hash	Hash de petición.
doc.hash	Hash del documento.
request.hash.1	Hash de la petición 1 para la consulta masiva de peticiones
request.hash.2	Hash de la petición 2 para la consulta masiva de peticiones
request.hash.3	Hash de la petición 3 para la consulta masiva de peticiones
security.enabled	Parámetro para establecer si la aplicación para la que se van a enviar peticiones tiene activada la seguridad en servicios web.
security.pass	Contraseña para autenticación en servicios web.
protocol	Protocolo para el envío de documentos por referencia.
protocol.file.path	Ruta del documento en el repositorio externo, para el envío de documentos por referencia.
protocol.file.name	Nombre del documento en el repositorio externo, para el envío de documentos por referencia.
protocol.identifier	Identificador del documento en el repositorio externo, para el envío de documentos por referencia.
protocol.file.local	Ruta local al fichero vinculado a una petición por referencia. El cliente lo necesita para calcular su resumen HASH y poderlo enviar como un parámetro en el método de creación de peticiones por referencia.
hash.algorithm	Algoritmo de Hash.
request.num.docume	Número de documentos para adjuntar a la petición.

nts	

Para ejecutar la aplicación de test bastará con ejecutar la siguiente instrucción desde línea de comandos:

```
java -jar pfirma-test-ws-v2-3.1.1-jar-with-dependencies.jar
```

Tras esto se mostrará el menú principal de la aplicación.

Existe la opción de ejecutar el proyecto de pruebas indicando por parámetro el test que se desea realizar y el número de veces que debe ejecutarse la prueba, por ejemplo:

```
java -jar pfirma-test-ws-v2-3.1.1-jar-with-dependencies.jar 4 3
```

Con la instrucción anterior se ejecutará el test 4, 3 veces.

También se posible abrir el proyecto a través del IDE Eclipse, ejecutando el siguiente comando desde la línea de comandos y situándose en el raíz del proyecto:

```
mvn clean eclipse:eclipse
```

La clase que contiene el método main ejecutable es la siguiente:

```
es.juntadeandalucia.pfirmav2.ws.test.TestLauncher
```

2.3.3.4 Detalle de pruebas

A continuación se detallan cada uno de los casos probados en la aplicación.

La clase ***ClientManager.java*** es la encargada de generar los clientes del servicio web a través de sus respectivas URL definidas en el fichero ***ws.properties***. Dicha clase pertenece al componente pfirmav2CXFClient. Port@firmas incluye la posibilidad de activar la autenticación mediante usuario y clave para los servicios web. Ésta funcionalidad es configurable por aplicación y afecta al servicio Modify, por lo que según a qué aplicación vayan vinculadas las peticiones que se van a modificar habrá que obtener el cliente de una forma u otra (con o sin seguridad). Si se obtiene un cliente sin seguridad y se intenta modificar una petición cuya aplicación tenga la seguridad activa, el sistema devolverá un error indicando que es necesaria una autenticación.

Obtención de clientes sin autenticación

Para obtener los clientes para operar con peticiones asociadas a peticiones con la autenticación desactivada basta con saber la url de los servicios a emplear y llamar a los correspondientes métodos de obtención de los clientes, tal y como se muestra en el fragmento de código a continuación.

```
ClientManager cm = new ClientManager();
QueryService queryServiceClient = cm.getQueryServiceClient("http://servidor:puerto/pfirmav3.1.1/servicesv2/QueryService?wsdl");
ModifyService modifyServiceClient = cm.getModifyServiceClient("http://servidor:puerto/pfirmav3.1.1/servicesv2/ModifyService?wsdl");
```

Obtención de clientes con autenticación

Para obtener los clientes autenticados son necesarios además de la url de los servicios, el código de la aplicación en Port@firmas (hará las funciones de usuario) y la clase de autenticación (clase en la que se establece la contraseña para autenticar).

```
ClientManager cm = new ClientManager();

QueryService queryServiceClient =
cm.getQueryServiceClientWithSecurity
("http://servidor:puerto/pfirmav3.1.1/servicesv2/QueryService?wsdl", "app", WSAuthenticate.class);

ModifyService modifyServiceClient =
cm.getModifyServiceClientWithSecurity("http://servidor:puerto/pfirmav3.1.1/servicesv2/ModifyService?wsdl", "app", WSAuthenticate.class);
```

El identificador de la aplicación debe coincidir con el de la aplicación a la que van asociadas las peticiones que se vayan a gestionar a través del servicio Modify.

La clase WSAuthenticate es una clase de manejo de callback en la que se establece la contraseña para que sea insertada en la petición SOAP y así poder autenticar en los servicios web. Por ello la clase debe implementar la interface CallbackHandler y en concreto el método handle, en el que se establece la contraseña. A continuación se muestra un ejemplo simple de implementación:

```
public class WSAuthenticate implements CallbackHandler {
```

```
        public void handle(Callback[] callbacks) throws IOException,
                        UnsupportedCallbackException {
                WSPasswordCallback pc = (WSPasswordCallback) callbacks[0];
                pc.setPassword("12345");
        }
}
```

Una vez explicada la forma de obtención de los clientes, se pasa a detallar uno a uno los distintos ejemplos existentes en la aplicación de ejemplo.

Opción 1 - Envío de una petición de firma a un único firmante con acción asociada

Esta modalidad de envío debe evitarse, en favor de la que se describe a continuación como "opción 2" de envío por referencia.

Seleccionando la primera opción desde el menú de la aplicación se lanzará dicho test. En este ejemplo se crea una petición y se envía a un único destinatario (el destinatario es el mismo que el remitente). El tipo de firma será de primer firmante y se le asociará una acción web a la petición que se lance cuando la petición sea leída por el firmante. El funcionamiento de las acciones así como sus distintos tipos se describe más adelante en este mismo documento.

Una vez generados los clientes, se obtendrá el usuario que actuará a su vez como remitente y destinatario de la petición. Para ello, se hará una consulta de usuarios a través del método ***queryUsers*** pasando como filtro el dni del usuario que se pretende obtener.

```
UserList userList = queryServiceClient.queryUsers("12345678Z");
User user = userList.getUser().get(0);
```

Tras obtener el usuario, se asignará el mismo como remitente y firmante de la petición. Remitente(s) y firmante(s) son añadidos a la petición a través de listas de sus respectivas clases por lo que es necesario instanciar una lista de remitentes y otra de firmantes y añadirles el usuario obtenido anteriormente.

```
RemitterList remitterList = new RemitterList();
remitterList.getUser().add(user);
```

```
Signer signer = new Signer();
signer.setUserJob(user);
SignerList signerList = new SignerList();
signerList.getSigner().add(signer);
```

El siguiente paso en la creación de la petición es añadirle una línea de firma que contenga al firmante que se definió anteriormente. Tal como se comentó en el punto anterior, los firmantes han de añadirse a las líneas de firma a través de listas por lo que se instanciará una lista de firmantes para añadirlas a la línea de firma y ésta será añadida también a una lista de líneas de firma. Si no se especifica el tipo de línea de firma, ésta toma por defecto el valor de firma.

```
SignLine signLine = new SignLine();
signLine.setSignerList(signerList);
SignLineList signLineList = new SignLineList();
```

```
signLineList.getSignLine().add(signLine);
```

Tras crear la línea de firma se procederá a obtener el tipo del documento que se añadirá a la petición. Los tipos de documentos válidos se encuentran definidos en el Anexo I del presente documento

```
DocumentTypeList docTypeList = queryServiceClient.queryDocumentTypes("TD99");
DocumentType docType = docTypeList.getDocumentType().get(0);
```

Y se adjuntará el documento a la petición. Mediante el método ***setSign*** se especifica que el documento será firmable o anexo. El contenido binario del mismo se pasa a través de un objeto ***DataHandler***. Los documentos de la petición también se almacenarán en una lista para la creación de la petición.

```
Document doc = new Document();
doc.setSign(true);
doc.setDocumentType(docType);
doc.setMime("text/plain");
doc.setName("fichero_a_firmar");
DataSource ds = new FileDataSource("/home/user/fichero.txt");
DataHandler dh = new DataHandler(ds);
doc.setContent(dh);
DocumentList docList = new DocumentList();
docList.getDocument().add(doc);
```

Tras crear la lista de documentos se procederá a crear la lista de acciones a ejecutar añadiendo la acción tipo web que invocará la web indicada en el fichero ***ws.properties*** cuando la petición haya pasado a estado leída. Para ello basta con instanciar un objeto ***Action*** y otro ***State*** que marcará el estado en el que se ejecutará la acción y establecer sus propiedades como se indica en el fragmento de código a continuación.

```
Action action = new Action();
action.setType("WEB");
action.setAction(properties.get(Constants.WEB_ACTION_PROP));
State stateRead = new State();
stateRead.setIdentifier("LEIDO");
action.setState(stateRead);
ActionList actionList = new ActionList();
actionList.getAction().add(action);
```

Una vez creada la lista de acciones de la petición, el siguiente paso es instanciar un objeto ***Request*** y establecer sus propiedades con los datos obtenidos anteriormente además de indicar la aplicación a la que irá asociada dicha petición, el asunto, texto, referencia y su tipo de firma.

```
Request req = new Request();
req.setApplication(application);
req.setDocumentList(docList);
req.setReference("Referencia");
req.setRemitterList(remitterList);
req.setSignLineList(signLineList);
req.setSignType("PRIMER FIRMANTE");
req.setSubject("Asunto");
req.setText("Texto de la petición");
req.setActionList(actionList);
```

Una vez creado el objeto, se creará la petición a través del método ***createRequest*** del servicio de modificación pasándole el objeto ***Request***, devolviendo el hash de la petición creada.

```
String requestHash = modifyServiceClient.createRequest(req);
```

Y para enviar la petición que ha sido creada, basta con invocar el método sendRequest del servicio de modificación, pasándole su código hash asociado.

```
modifyServiceClient.sendRequest(requestHash);
```

Tras este proceso, se habrá creado y enviado la petición. En caso de producirse algún error, el servició lanzará una ***PfirmaException*** con los detalles de dicho error.

Opción 2 - Envío de una petición de firma a un único firmante con acción asociada y documentos por referencia

El envío de documentos por referencia será la opción de uso preferente. Seleccionando la segunda opción desde el menú de la aplicación se lanzará dicho test. En este ejemplo se crea una petición y se envía a un único destinatario (el destinatario es el mismo que el remitente). El tipo de firma será de primer firmante y se le asociará una acción web a la petición que se lance cuando la petición sea leída por el firmante. Cabe destacar que el método para adjuntar el fichero en este caso de uso será el envío por referencia, este método de envío se explicará más detalladamente en este mismo documento. El funcionamiento de las acciones así como sus distintos tipos se describe más adelante en este mismo documento.

Una vez generados los clientes, se obtendrá el usuario que actuará a su vez como remitente y destinatario de la petición. Para ello, se hará una consulta de usuarios a través del método ***queryUsers*** pasando como filtro el DNI del usuario que se pretende recuperar.

```
UserList userList = queryServiceClient.queryUsers("12345678Z");
User user = userList.getUser().get(0);
```

Tras obtener el usuario, se asignará el mismo como remitente y firmante de la petición. Remitente(s) y firmante(s) son añadidos a la petición a través de listas de sus respectivas clases por lo que es necesario instanciar una lista de remitentes y otra de firmantes y añadirles el usuario obtenido anteriormente.

```
RemitterList remitterList = new RemitterList();
remitterList.getUser().add(user);
```

```
Signer signer = new Signer();
signer.setUserJob(user);
SignerList signerList = new SignerList();
signerList.getSigner().add(signer);
```

El siguiente paso en la creación de la petición es añadirle una línea de firma que contenga al firmante que se definió anteriormente. Tal como se comentó en el punto anterior, los firmantes han de añadirse a las líneas de firma a través de listas por lo que se instanciará una lista de firmantes para añadirlas a la línea de firma y ésta será añadida también a una lista de líneas de firma. Si no se especifica el tipo de línea de firma, ésta toma por defecto el valor de firma.

```
SignLine signLine = new SignLine();
signLine.setSignerList(signerList);
SignLineList signLineList = new SignLineList();
signLineList.getSignLine().add(signLine);
```

Tras crear la línea de firma se procederá a obtener el tipo del documento que se añadirá a la petición. Los tipos de documentos válidos se encuentran definidos en el Anexo I del presente documento

```
DocumentTypeList docTypeList = queryServiceClient.queryDocumentTypes("TD99");
DocumentType docType = docTypeList.getDocumentType().get(0);
```

Y se adjuntará el documento a la petición a través del método de Envío por Referencia. Mediante el método ***setSign*** se especifica que el documento será firmable o anexo. Los documentos de la petición también se almacenarán en una lista de los mismos para la creación de la petición.

```
DocumentReferenceList documentReferenceList = new DocumentReferenceList();

DocumentReference doc = new DocumentReference();
doc.setDocumentType(docType);
doc.setMime("text/plain");
doc.setSign(true);
doc.setName("fichero_a_firmar.txt");
doc.setHashToSign("6df21fb3842b829098b5be84f9271a14d3d62731");
doc.setHashAlgorithm("SHA1");
doc.setSize("4345667"); //Tamaño (en bytes) del documento a firmar
doc.setCsv("PFDES748PFIRMA6XDM9sUuRtA4l29p");

ParameterList params = new ParameterList();
params.getParameter().add(new Parameter("REPO_REF_PATH", "/context/"));
params.getParameter().add(new Parameter("REPO_REF_FILENAME",
"fichero_a_firmar.txt"));
doc.setReferenceParams(params);

documentReferenceList.getDocumentReferenceList().add(doc);
```

Tras crear la lista de documentos se procederá a crear la lista de acciones a ejecutar añadiendo la acción tipo web que invocará la web indicada en el fichero ***ws.properties*** cuando la petición haya pasado a estado leída. Para ello basta con instanciar un objeto ***Action*** y otro ***State*** que marcará el estado en el que se ejecutará la acción y establecer sus propiedades como se indica en el fragmento de código a continuación.

```
Action action = new Action();
action.setType("WEB");
```

```
action.setAction(properties.get(Constants.WEB_ACTION_PROP));
State stateRead = new State();
stateRead.setIdentifier("LEIDO");
action.setState(stateRead);
ActionList actionList = new ActionList();
actionList.getAction().add(action);
```

Una vez creada la lista de acciones de la petición, el siguiente paso es instanciar un objeto ***Request*** ByReference y establecer sus propiedades con los datos obtenidos anteriormente además de indicar la aplicación a la que irá asociada dicha petición, el asunto, texto, referencia y tipo de firma de la misma.

```
RequestByReference req = new RequestByReference();
req.setApplication(application);
req.setDocumentReferenceList(documentReferenceList);
req.setReference("Referencia");
req.setRemitterList(remitterList);
req.setSignLineList(signLineList);
req.setSignType("PRIMER FIRMANTE");
req.setSubject("Asunto");
req.setText("Texto de la petición");
req.setActionList(actionList);
```

Una vez creado el objeto, se creará la petición a través del método ***createRequestByReference*** del servicio de modificación pasándole el objeto ***RequestByReference***, devolviendo el hash de la petición creada.

```
String requestHash = modifyServiceClient.createRequestByReference(req);
```

Y para enviar la petición que ha sido creada, basta con invocar el método sendRequest del servicio de modificación, pasándole el hash de la misma.

```
modifyServiceClient.sendRequest(requestHash);
```

Tras este proceso, se habrá creado y enviado la petición. En caso de producirse algún error, el servicio lanzará una ***PfirmaException*** con los detalles de dicho error.

Opción 3 - Envío de una petición de firma a dos firmantes con acción asociada

Seleccionando la tercera opción desde el menú de la aplicación se lanzará dicho test. En este ejemplo se crea una petición y se envía a dos usuarios (el primer firmante es el mismo que el remitente). El tipo de firma será en cascada.

Una vez generados los clientes, se obtendrá el usuario que actuará a su vez como remitente y destinatario de la petición. Para ello, se hará una consulta de usuarios a través del método ***queryUsers*** pasando como filtro el DNI del usuario que se pretende recuperar.

```
UserList userList = queryServiceClient.queryUsers("12345678Z");
User user = userList.getUser().get(0);
userList = queryServiceClient.queryUsers("00000000T");
User user2 = userList.getUser().get(0);
```

Tras obtener ambos usuarios se procede a definir los firmantes y el remitente.

```
Signer signerUser_1 = new Signer();
signerUser_1.setUserJob(user);
Signer signerUser_2 = new Signer();
signerUser_2.setUserJob(user2);
SignerList signerList = new SignerList();
signerList.getSigner().add(signerUser_1);
signerList.getSigner().add(signerUser_2);
```

Una vez obtenidos los firmantes se añaden a una nueva línea de firma.

```
Signer signerUser_1 = new Signer();
signerUser_1.setUserJob(user);
Signer signerUser_2 = new Signer();
signerUser_2.setUserJob(user2);
SignerList signerList = new SignerList();
signerList.getSigner().add(signerUser_1);
signerList.getSigner().add(signerUser_2);
```

Tras ello, mediante un proceso igual que en el ejemplo anterior, se crea el objeto petición y se le añaden documentos, firmantes y remitente y se establecen el texto, asunto, referencia y tipo de firma.

```
DocumentTypeList docTypeList = queryServiceClient.queryDocumentTypes("TD99");
DocumentType docType = docTypeList.getDocumentType().get(0);
```

```
Document doc = new Document();
doc.setSign(true);
doc.setDocumentType(docType);
doc.setMime("text/plain");
doc.setName("fichero_a_firmar");
DataSource ds = new FileDataSource("/home/user/fichero.txt");
DataHandler dh = new DataHandler(ds);
doc.setContent(dh);
DocumentList docList = new DocumentList();
docList.getDocument().add(doc);
```

```
Request req = new Request();
req.setApplication(application);
req.setDocumentList(docList);
req.setReference("Referencia");
req.setRemitterList(remitterList);
req.setSignLineList(signLineList);
req.setSignType("CASCADA");
req.setSubject("Asunto");
req.setText("Texto de la petición");
```

Una vez creado el objeto, se creará la petición a través del método ***createRequest*** del servicio de modificación pasándole el objeto ***Request***, devolviendo el hash de la petición creada.

```
String requestHash = modifyServiceClient.createRequest(req);
```

Y para enviar la petición que ha sido creada, basta con invocar el método sendRequest del servicio de modificación, pasándole el hash de la misma.

```
modifyServiceClient.sendRequest(requestHash);
```

Tras este proceso, se habrá creado y enviado la petición. En caso de producirse algún error, el servicio lanzará una ***PfirmaException*** con los detalles de dicho error.

Opción 4 - Envío de una petición con documento anexo de visto bueno a un usuario y firma posterior de otro

Seleccionando la cuarta opción en el menú de la aplicación se lanzará dicho test. En este ejemplo se crea una petición que contiene una línea de visto bueno para un usuario y una segunda línea de firma para otro usuario. El tipo de firma es en cascada y lleva un documento anexo.

Una vez generados los clientes, se obtendrán los usuarios que actuará a su vez como remitente y destinatarios de la petición. Para ello, se hará una consulta de usuarios a través del método ***queryUsers*** pasando como filtro el DNI del usuario que se pretende recuperar.

```
UserList userList = queryServiceClient.queryUsers("12345678Z");
User user = userList.getUser().get(0);
userList = queryServiceClient.queryUsers("00000000T");
User user2 = userList.getUser().get(0);
```

Tras obtener los 2 usuarios se procede a definir los firmantes y el remitente.

```
Signer signerUser_1 = new Signer();
signerUser_1.setUserJob(user);
Signer signerUser_2 = new Signer();
signerUser_2.setUserJob(user2);
SignerList signerList = new SignerList();
signerList.getSigner().add(signerUser_1);
signerList.getSigner().add(signerUser_2);
```

```
RemitterList remitterList = new RemitterList();
remitterList.getUser().add(user);
```

Una vez obtenidos los firmantes se procederá a crear las líneas de firma y visto bueno.

```
SignLine signLine = new SignLine();
SignLine signLine2 = new SignLine();
SignerList signerList = new SignerList();
SignerList signerList2 = new SignerList();
signerList.getSigner().add(signerUser_1);
signerList2.getSigner().add(signerUser_2);
signLine.setSignerList(signerList);
signLine2.setSignerList(signerList2);
signLine.setType("VISTO BUENO");
```

```
signLine2.setType("FIRMA");
SignLineList signLineList = new SignLineList();
signLineList.getSignLine().add(signLine2);
signLineList.getSignLine().add(signLine);
```

A continuación se procede a crear los documentos que se incluirán en la petición, obteniendo primero el tipo de documento de ambos. Los tipos de documentos válidos se encuentran definidos en el Anexo I del presente documento

```
DocumentTypeList docTypeList = queryServiceClient.queryDocumentTypes("TD99");
DocumentType docType = docTypeList.getDocumentType().get(0);
```

Se crea el primer documento para firma.

```
Document doc = new Document();
doc.setSign(true);
doc.setDocumentType(docType);
doc.setMime(properties.get("plain/txt");
doc.setName(properties.get("Documento para firma");
DataSource ds = new FileDataSource("home/user/prueba.txt");
DataHandler dh = new DataHandler(ds);
doc.setContent(dh);
```

Y posteriormente se crea el documento anexo (Se utiliza el mismo objeto ***DataHandler*** dado que se va a usar el mismo contenido binario para ambos documentos.

```
Document doc2 = new Document();
doc2.setSign(false);
doc2.setDocumentType(docType);
doc2.setMime("plain/text");
doc2.setName("Documento anexo");
doc2.setContent(dh);
```

Y finalmente se añaden a la lista de documentos a adjuntar en la petición.

```
DocumentList docList = new DocumentList();
docList.getDocument().add(doc);
docList.getDocument().add(doc2);
```

Tras ello, mediante un proceso igual que en el ejemplo anterior, se crea el objeto petición y se le añaden firmantes y remitente y se establecen el texto, asunto, referencia y tipo de firma.

```
Request req = new Request();
req.setApplication(application);
req.setDocumentList(docList);
req.setReference("Referencia");
req.setRemitterList(remitterList);
req.setSignLineList(signLineList);
req.setSignType("CASCADA");
req.setSubject("Asunto");
req.setText("Texto de la petición");
```

Una vez creado el objeto, se creará la petición a través del método ***createRequest*** del servicio de modificación pasándole el objeto ***Request***, devolviendo el hash de la petición creada.

```
String requestHash = modifyServiceClient.createRequest(req);
```

Y para enviar la petición que ha sido creada, basta con invocar el método sendRequest del servicio de modificación, pasándole el hash de la misma.

```
modifyServiceClient.sendRequest(requestHash);
```

Tras este proceso, se habrá creado y enviado la petición. En caso de producirse algún error, el servicio lanzará una ***PfirmaException*** con los detalles de dicho error.

Opción 5 - Eliminación de una petición

Seleccionando la quinta opción en el menú de la aplicación se lanzará dicho test. En este ejemplo se crea una petición con los datos especificados en el fichero *ws.properties* y la elimina a continuación.

Una vez generados los clientes, se creará y enviará la petición que será eliminada a continuación.

```
String requestHash = createRequest(queryServiceClient,modifyServiceClient);
log.info("Request created correctly with hash " + requestHash);
// Send request
modifyServiceClient.sendRequest(requestHash);
log.info("Request sent");
```

Una vez creada la petición procedemos a eliminarla a través del servicio ***deleteRequest*** pasándole como parámetro el hash de dicha petición de firma.

```
modifyServiceClient.deleteRequest(requestHash);
log.info("Request with hash " + requestHash + " deleted correctly");
```

Si la petición se ha completado por parte de los usuarios retornará un error y no podrá eliminarse.

En el caso de si las firmas/VBs de la petición se han comenzado pero no se han completado, la petición quedará en estado “ABORTADO” y aparecerá en la bandeja de terminadas de los firmantes en este estado.

Tras este proceso, se habrá eliminado la petición. En caso de producirse algún error, el servicio lanzará una ***PfirmaException*** con los detalles de dicho error.

Opción 6 - Creación de una petición con metadatos

Seleccionando la sexta opción en el menú de la aplicación se lanzará dicho test. En este ejemplo se crea una petición con los datos especificados, entre los que se encuentran los nombres de 2 parámetros de la aplicación que se añadirán como información adicional a la petición enviada.

Una vez generados los clientes, se creará y se llamará al servicio de consulta de petición pasándole como parámetro el hash de la petición creada para así obtener toda la información de la petición en un objeto ***Request***.

```
// Create request
String requestHash = createRequest(queryServiceClient,modifyServiceClient);
// Obtain request
Request req = queryServiceClient.queryRequest(requestHash);
```

Una vez obtenida la petición, pasamos a añadirle al objeto ***Request*** los dos parámetros cuyos nombres están especificados en el fichero ***ws.properties*** y estableciendo como valor una cadena de prueba.

```
// Add parameters
Parameter param1 = new Parameter();
param1.setIdentifier(properties.get(Constants.PARAMETER_1));
param1.setValue("Testing value 1");
Parameter param2 = new Parameter();
param2.setIdentifier(properties.get(Constants.PARAMETER_2));
param2.setValue("Testing value 2");
```

Una vez creados los parámetros se añaden a un nuevo objeto ***ParameterList*** que se asignará al objeto ***Request*** que estamos editando.

```
ParameterList paramList = new ParameterList();
paramList.getParameter().add(param1);
paramList.getParameter().add(param2);
req.setParamterList(paramList);
```

Una vez añadida la información adicional a la petición bastará con enviar la petición. En el código de ejemplo se recorre la lista de parámetros de la petición y se muestran por consola para comprobar que se han añadido correctamente.

```
// Send request
modifyServiceClient.sendRequest(requestHash);
```

```
log.info("Request metadata:");
for (Parameter param : req.getParamterList().getParameter()) {
       log.info(param.getIdentifier() + ": " + param.getValue());
}
```

En la interfaz web, las peticiones con metadatos incluyen un icono con el texto alternativo "Contiene información adicional" en el campo "Asunto" de su respectiva bandeja. Una vez dentro del detalle de la petición, en el caso de las peticiones con metadatos aparecerá una pestaña "Información adicional". En esta pestaña aparecerán los metadatos incluidos en la petición y sus respectivos valores.

En caso de producirse algún error, el servicio lanzará una ***PfirmaException*** con los detalles de dicho error.

Opción 7 - Actualización de firmantes de una petición de firma

Seleccionando la séptima opción en el menú de la aplicación se lanzará dicho test. En este ejemplo se crea una petición con los datos especificados, y se actualizan las líneas de firma eliminando y añadiendo nuevos firmantes antes y después de haber enviado la petición.

Una vez generados los clientes, se creará la petición y se llamará al servicio de consulta para obtener un objeto ***Request*** con todos los datos de la misma.

```
String requestHash = createRequest(queryServiceClient,
                                   modifyServiceClient);
Request req = queryServiceClient.queryRequest(requestHash);
```

A continuación se eliminará el único firmante que tiene la petición a través del servicio ***deleteSigners*** pasándole el hash de petición y la lista de firmantes a eliminar(en este caso la que contiene la única línea de firma de la petición) y volvemos a llamar al servicio de consulta de peticiones para actualizar el objeto ***Request***.

```
modifyServiceClient.deleteSigners(requestHash,
             req.getSignLineList().getSignLine().get(0).getSignerList());
req = queryServiceClient.queryRequest(requestHash);
```

Tras haber dejado la petición sin líneas de firma ni firmantes, se procederá a insertar una nueva línea de firma con un firmante indicado en el fichero ***ws.properties***. Para ello basta con llamar al servicio ***insertSigners*** pasándole el hash de la petición, el número de la línea de firma (contando desde 0) y la lista de firmantes a insertar. Para ello creamos los objetos ***Signer*** y ***SignerList*****n** y los cargamos con la información especificada en el fichero properties y se envía. Tras ello volvemos a actualizar el objeto ***Request*** a mediante el servicio de consulta.

Al no existir la línea de firma 0, ésta se crea automáticamente. Una vez enviada la petición se procederá a eliminar el firmante insertado anteriormente para verificar que tras esta operación, la petición pasará a estado “Nueva”. El procedimiento es igual al descrito anteriormente.

```
modifyServiceClient.deleteSigners(requestHash, req
             .getSignLineList().getSignLine().get(0).getSignerList());
```

Opción 8 - Eliminación de última línea de firma en petición cuya penúltima línea de firma es de tipo visto bueno

Seleccionando la octava opción en el menú de la aplicación se lanzará dicho test. En este ejemplo se crea una petición cuya primera línea de firma es de visto bueno y la segunda de tipo firma (cada una con un único firmante). Tras ello se intenta eliminar el último firmante y el sistema devolverá un error indicando que una petición no puede contener como última línea de firma una de tipo visto bueno.

Una vez generados los clientes, se creará la petición, se enviará y se llamará al servicio de consulta para obtener el objeto ***Request*** con todos los datos de la petición.

```
JobList jobList = queryServiceClient.queryJobs(properties
                                    .get(Constants.JOB_ID_PROP));
Job job = jobList.getJob().get(0);
// Creating signers
Signer signerUser = new Signer();
signerUser.setUserJob(user);
Signer signerJob = new Signer();
signerJob.setUserJob(job);
// Creating sign lines
log.info("Creating sign line");
SignLine signLine = new SignLine();
SignLine signLine2 = new SignLine();
SignerList signerList = new SignerList();
SignerList signerList2 = new SignerList();
signerList.getSigner().add(signerUser);
signerList2.getSigner().add(signerJob);
signLine.setSignerList(signerList);
signLine2.setSignerList(signerList2);
signLine.setType(Constants.SIGN_TYPE);
signLine2.setType(Constants.PASS_TYPE);
SignLineList signLineList = new SignLineList();
signLineList.getSignLine().add(signLine2);
signLineList.getSignLine().add(signLine);
// Create request
String requestHash = createRequest(queryServiceClient,
                                    modifyServiceClient);
modifyServiceClient.sendRequest(requestHash);
// Get request
Request req = queryServiceClient.queryRequest(requestHash);
```

Tras esto se actualiza el objeto request con las líneas de firma en la disposición comentada anteriormente y se vuelve a enviar (ya que al eliminar los firmantes, ésta pasa de nuevo a estado "no enviada").

```
// Delete signers
modifyServiceClient.deleteSigners(requestHash,
              req.getSignLineList().getSignLine().get(0).getSignerList());
req.setSignType(Constants.SIGN_TYPE_CASCADE);
req.setSignLineList(signLineList);
// Update request
modifyServiceClient.updateRequest(req);
// Resend request
modifyServiceClient.sendRequest(requestHash);
```

Tras ello intentaremos eliminar la línea de firma tipo "firma" para que a posteriori nos salte una excepción indicando que la petición no puede contener como última línea de firma una línea tipo "visto bueno".

```
// Try to delete last sign line (must fail)
modifyServiceClient.deleteSigners(requestHash, signerList);
```

Opción 9 - Consulta de comentarios e histórico de petición

Seleccionando la novena opción en el menú de la aplicación se lanzará dicho test. En este ejemplo se consultan el histórico y los comentarios de una petición a partir de su hash. A continuación se muestran por consola los datos obtenidos.

Una vez generado el cliente de consulta, se llama al servicio de consulta de comentarios **queryComments** de una petición pasándole su hash y una vez obtenido el objeto **CommentList** se itera sobre la lista de comentarios contenida en caso de que ésta no venga vacía. De cada objeto **Comment** obtenido se muestran la fecha, el identificador del usuario que comentó y el texto del comentario.

```
log.debug("Obtaining comments for request: "
              + properties.get(Constants.REQUEST_HASH_PROP));
CommentList commentList = queryServiceClient
              .queryComments(properties.get(Constants.REQUEST_HASH_PROP));
if (commentList != null && commentList.getComment() != null
              && !commentList.getComment().isEmpty()) {
       log.info("Comments for request " +
       properties.get(Constants.REQUEST_HASH_PROP));
       for (Comment comment : commentList.getComment()) {
              log.info(comment.getFcomment() + " - "
                     + comment.getUser().getIdentifier() + " - "
                     + comment.getText());
       }
} else {
       log.info("No comments for request " +
              properties.get(Constants.REQUEST_HASH_PROP));
}
```

Una vez mostrados los comentarios, se hace lo propio con el histórico de petición. Se obtiene el objeto HistoricList que contiene la lista de entradas en el histórico de la petición a través del servicio queryHistoric pasándole su hash y se itera sobre ellas. De cada objeto Historic se muestra el texto del mismo por consola.

```
HistoricList historicList = queryServiceClient
       .queryHistoric(properties.get(Constants.REQUEST_HASH_PROP));
log.debug("Obtaining historic for request: "
       + properties.get(Constants.REQUEST_HASH_PROP));
if (historicList != null && historicList.getHistoric() != null
       && !historicList.getHistoric().isEmpty()) {
       log.info("Historic for request "
+ properties.get(Constants.REQUEST_HASH_PROP));
       for (Historic historic : historicList.getHistoric()) {
```

```
                log.info(historic.getText());
        }
}
```

Las aplicaciones que generen un alto número de peticiones se configurarán específicamente para que los cambios de estados de las peticiones no se notifiquen a través de una invocación de URL, sino que será la aplicación que ha generado la petición la encargada de consultar el estado de cada una de estas peticiones. Ya que hacer esta consulta de histórico acerca de numerosas peticiones de forma iterativa podría suponer un alto tiempo de ejecución, Port@firmas ofrece un servicio de consulta masiva de estado, a través del cual se podrán consultar los históricos de numerosas peticiones en una sola invocación al webservice, haciendo uso del método **queryHistoricList** el cual nos devolverá un listado de objetos **HistoricList** es decir, un objeto que será una lista que contendrá todos los históricos de las peticiones consultadas (**HistoricListList)**.

```
RequestIdList requestIdList = new RequestIdList();


requestIdList.getRequestId().add(properties.get(Constants.REQUEST_HASH_PROP_1));


requestIdList.getRequestId().add(properties.get(Constants.REQUEST_HASH_PROP_2));


requestIdList.getRequestId().add(properties.get(Constants.REQUEST_HASH_PROP_3));
HistoricListList historicList = queryServiceClient.queryHistoricList(requestIdList);
                    log.debug("Obtaining historicList for requests: "
                                        + properties.get(Constants.REQUEST_HASH_PROP_1) + ", "
+ properties.get(Constants.REQUEST_HASH_PROP_2) + ", " +
properties.get(Constants.REQUEST_HASH_PROP_3));
if (historicList != null && historicList.getHistoricObjectResponseList() != null
&& !historicList.getHistoricObjectResponseList().isEmpty()) {
log.info("Historic for requests " + properties.get(Constants.REQUEST_HASH_PROP_1) + ", " +
properties.get(Constants.REQUEST_HASH_PROP_2) + ", " +
properties.get(Constants.REQUEST_HASH_PROP_3));
for (HistoricObjectResponse historicObjectResponse :
historicList.getHistoricObjectResponseList()) {


if(historicObjectResponse.getHistoricList() != null){
for(Historic historic: historicObjectResponse.getHistoricList().getHistoric()){
log.info(historic.getText());
                }
        }
}
}
```

Opción 10 - Descarga de un documento, de su firma, de su informe de firma y consulta de su código de verificación

En este caso de uso se supone creado un documento vinculado a una petición, la cual ha sido firmada previamente, conociéndose el código identificador del documento y de la petición.

```
String documentCHash = properties.get(Constants.DOCUMENT_HASH_PROP);
String requestCHash = properties.get(Constants.REQUEST_HASH_PROP);
```

Aunque el documento original sea custodiado por la aplicación que generó la petición, existe la posibilidad de descargar este documento a través de los servicios web de Port@firmas:

```
byte [] originalDocumentContent = queryServiceClient.downloadDocument(documentCHash);
```

De igual forma es posible descargar el fichero correspondiente a la firma de dicho documento:

```
byte [] signContent = queryServiceClient.downloadSign(documentCHash);
```

También se puede acceder al informe de firma, indicando determinados parámetros de configuración acerca de la posibilidad de escalar, rotar o reducir a A4, al igual que se puede hacer a través de la interfaz web de la herramienta:

```
boolean scale = true;
boolean rotate = false;
boolean a4 = true;
byte [] signReportContent = queryServiceClient.downloadReport(documentCHash, requestCHash, scale, rotate, a4);
```

Por último, la consulta del código seguro de verificación de un documento se puede realizar a través del siguiente método:

```
String csv = queryServiceClient.queryCsv(documentCHash);
```

Opción 11 - Acceso al texto que contiene el motivo indicado por un usuario que ha rechazado la firma de una petición

En este caso de uso se supone rechazada una petición y conocido su código Hash.

```
String requestCHash = properties.get(Constants.REQUEST_HASH_PROP);
```

En primer lugar se debe consultar el histórico de la petición:

```
HistoricList historicList = queryServiceClient.queryHistoric(requestCHash);
```

Una vez que tenemos el histórico de la petición, debemos iterar sobre todos los elementos y buscar el que se corresponde con el estado “DEVUELTO”:

```
for (Historic historic : historicList.getHistoric()) {
  String state = historic.getState().getIdentifier();
  if(StringUtils.equalsIgnoreCase(state, "DEVUELTO")){
    String text = historic.getText();
  }
}
```

Dicho texto se corresponde con un texto descriptivo del rechazo de la petición. El texto introducido por el usuario se puede extraer parseando dicho texto:

```
Pattern patron = Pattern.compile("\"(.*)\"");
Matcher matcher = patron.matcher(text);
matcher.find();
String rejectionText = matcher.group(1);
```

2.3.4 Generación de cliente de WS con Apache CXF

En la aplicación de ejemplo de integración del apartado anterior se facilita un cliente de servicio web de Port@firmas generado con Apache CXF (**pfirmav2CXFClient.jar**).

Normalmente en integraciones con Port@firmas se suele utilizar este jar.

Para utilizar este jar, es necesario que incluya en su pom.xml la siguiente dependencia:

```
<dependency>
	<groupId>es.juntadeandalucia.pfirma</groupId>
	<artifactId>pfirmav2CXFClient</artifactId>
	<version>3.1.0</version>
</dependency>
```

Esta librería se encuentra en el repositorio Maven de SCAE en la siguiente URL:

https://ws024.juntadeandalucia.es/maven/internal

Si por cualquier motivo se quiere generar los clientes desde el wsdl de la aplicación hay que tener en cuenta el siguiente aspecto.

Al crear los clientes debemos hacerlo con la línea:

```
wsdl2java -b bindings.xml
```

siendo **bindings.xml**:

```
<jaxb:bindings version="2.1"
	xmlns:jaxb="http://java.sun.com/xml/ns/jaxb"
	xmlns:xjc="http://java.sun.com/xml/ns/jaxb/xjc"
	xmlns:xs="http://www.w3.org/2001/XMLSchema">
		<jaxb:globalBindings generateElementProperty="false" />
</jaxb:bindings>
```

3 INTERFAZ DE CUSTODIA DE FICHEROS

Port@firmas ofrece la posibilidad de implementar un sistema propio de almacenamiento de documentos y firmas que no sea el que ofrece la aplicación por defecto.

Para ello basta con implementar dos interfaces:

- **CustodyServiceInput**

Contiene los métodos de subida de documentos y firmas.

- **CustodyServiceOutput**

Contiene los métodos de descarga de documentos y firmas.

3.1 Interfaz CustodyServiceInput

Se encuentra en el paquete:

es.juntadeandalucia.pfirma.utils.document.service

Contiene los métodos a implementar para la subida de ficheros. Si se produjese cualquier error durante la ejecución de cualquiera de sus métodos deberá lanzarse una excepción de tipo CustodyServiceException.

Dicha excepción se encuentra definida en el paquete:

es.juntadeandalucia.pfirma.utils.document

```
public interface CustodyServiceInput extends Serializable {

	public void initialize(Map<String, Object> parameterMap)
			throws CustodyServiceException;

	public String uploadFile(CustodyServiceInputDocument document,
			InputStream input) throws CustodyServiceException;

	public String uploadSign(CustodyServiceInputSign sign, InputStream
			input)
			throws CustodyServiceException;

}
```

El método initialize sirve para establecer los valores que sean necesarios en la implementación a través de un mapa de parámetros. En la implementación de ficheros, en

principio sería necesario tener la ruta donde serán almacenados los ficheros, por lo que recibiría dicha ruta en el mapa como un parámetro.

El método uploadFile se emplea para subir el documento. Para ello recibe una clase tipo CustodyServiceInputDocument con datos útiles para realizar la subida. Dicha clase no es más que un bean con propiedades y sus métodos de acceso y se encuentra en el paquete es.juntadeandalucia.pfirma.utils.document.bean.

El otro parámetro que recibe es un objeto InputStream con el contenido binario a subir.

A través del método uploadSign se suben al sistema los ficheros de firma generados tras el proceso de firma de peticiones. El funcionamiento es el mismo que en el método uploadFile salvo que el objeto que recibe es de tipo CustodyServiceInputSign que se encuentra en el paquete es.juntadeandalucia.pfirma.utils.document.bean.

A continuación se muestra la implementación de fichero para la subida de documentos.

```
public class FilePathCustodyServiceInputImpl implements CustodyServiceInput {
	private static final long serialVersionUID = 1L;

	public static final String C_PARAMETER_FILEPATH_PATH = "RUTA";

	private org.slf4j.Logger log = LoggerFactory
			.getLogger(FilePathCustodyServiceInputImpl.class);

	private String path;
	private Map<String, String> signExtensions;

	public void initialize(Map<String, Object> parameterMap)
			throws CustodyServiceException {
		this.path = (String) parameterMap.get(C_PARAMETER_FILEPATH_PATH);
		if (path == null || path.equals("")) {
			throw new CustodyServiceException(
					"No path to store files defined. Check
configuration parameters");
		}
		signExtensions = Util.getInstance().loadSignExtensions();
	}

	public String uploadFile(CustodyServiceInputDocument document,
			InputStream input) throws CustodyServiceException {
		log.info("uploadFile init");
		String filePath = null;
		// Get file name
		String fileName = Util.getInstance().getNameFile(document.getName());
```

```
		// Replace fileName with hash code
		fileName = Util.getInstance().replaceFileNameWithHash(fileName,
				document.getCheckHash());
		String separator = path.substring(path.length() - 1).equals(
				File.separatorChar + "") ? "" : File.separatorChar + "";
		filePath = path + separator + getYear() + separator + getWeek()
				+ separator;
		uploadFile(filePath, fileName, input);
		log.info("uploadFile end");
		return filePath + fileName;
	}

	public String uploadSign(CustodyServiceInputSign sign, InputStream input)
			throws CustodyServiceException {
		log.info("uploadSign init");
		String separator = path.substring(path.length() - 1).equals(
				File.separatorChar + "") ? "" : File.separatorChar + "";
		// Set file path
		String filePath = path + separator + getYear() + separator +
		getWeek() + separator;
		// Set sign extension
		String fileName = sign.getTransaction() + "."
				+ signExtensions.get(sign.getFormat());
		uploadFile(filePath, fileName, input);
		log.info("uploadFile end");
		return filePath + fileName;
	}

	private void uploadFile(String path, String name, InputStream input)
			throws CustodyServiceException {
		try {
			File f = new File(path);
			if (!f.exists()) {
				f.mkdirs();
			}
			OutputStream outputStream = new FileOutputStream(path + name);
			int read;
			byte[] buffer = new byte[Constants.BUFFER_SIZE];
			while ((read = input.read(buffer)) > 0) {
				outputStream.write(buffer, 0, read);
```

```
                    }
                    input.close();
                    outputStream.close();
              } catch (FileNotFoundException e) {
                    throw new CustodyServiceException(e.getMessage());
              } catch (IOException e) {
                    throw new CustodyServiceException(e.getMessage());
              }
       }

       private String getYear() {
              Calendar cal = Calendar.getInstance();
              SimpleDateFormat sdfYear = new SimpleDateFormat("yyyy");
              return sdfYear.format(cal.getTime());
       }

       private String getWeek() {
              Calendar cal = Calendar.getInstance();
              SimpleDateFormat sdfWeek = new SimpleDateFormat("ww");
              return sdfWeek.format(cal.getTime());

       }

}
```

3.2 Interfaz CustodyServiceOutput

Se encuentra en el paquete:

es.juntadeandalucia.pfirma.utils.document.service

Contiene los métodos a implementar para la descarga de ficheros. Si se produjese cualquier error durante la ejecución de cualquiera de sus métodos deberá lanzarse una excepción de tipo CustodyServiceException.

Dicha excepción se encuentra definida en el paquete:

es.juntadeandalucia.pfirma.utils.document

```
public interface CustodyServiceOutput extends Serializable {

	public void initialize(Map<String, Object> parameterMap)
			throws CustodyServiceException;

	public void downloadFile(CustodyServiceOutputDocument document,
			OutputStream outputStream) throws CustodyServiceException;

	public BigDecimal fileSize(CustodyServiceOutputDocument document)
			throws CustodyServiceException;

	public void downloadSign(CustodyServiceOutputSign sign,
			OutputStream outputStream) throws CustodyServiceException;

	public BigDecimal signSize(CustodyServiceOutputSign sign)
			throws CustodyServiceException;
}
```

El método initialize sirve para establecer los valores que sean necesarios en la implementación a través de un mapa de parámetros. En la implementación de ficheros, en principio no sería necesario ningún parámetro.

Mediante el método downloadFile se descargan los documentos en la aplicación. Dicho método recibe un objeto CustodyServiceOutputDocument con datos útiles para realizar la descarga del documento. Dicha clase no es más que un bean con propiedades y sus métodos de acceso y se encuentra en el paquete es.juntadeandalucia.pfirma.utils.document.bean.

El otro parámetro que recibe el método no es más que un objeto OutputStream para cargar con el contenido binario a descargar.

El método fileSize sirve para calcular el tamaño del contenido binario del documento a partir de los datos recibidos en el objeto CustodyServiceOutputDocument.

A través del método downloadSign se descargan del sistema las firmas almacenadas en el sistema tras el proceso de firma de peticiones. El funcionamiento es el mismo que en el método downloadFile salvo que el objeto que recibe es de tipo CustodyServiceOutputSign que se encuentra en el paquete es.juntadeandalucia.pfirma.utils.document.bean.

El método signSize es utilizado para calcular el tamaño del contenido binario de las firmas a través de los datos recibidos en el objeto CustodyServiceOutputSign.

A continuación se muestra la implementación de fichero para la subida de documentos.

```
public class FilePathCustodyServiceOutputImpl implements CustodyServiceOutput {

	private static final long serialVersionUID = 1L;

	private org.slf4j.Logger log = LoggerFactory
			.getLogger(FilePathCustodyServiceOutputImpl.class);

	public void initialize(Map<String, Object> parameterMap)
			throws CustodyServiceException {
		// nothing
	}

	public void downloadFile(CustodyServiceOutputDocument document,
			OutputStream outputStream) throws CustodyServiceException {
		log.info("downloadFile init");
		download(document.getUri(), outputStream);
		log.info("downloadFile end");
	}

	public BigDecimal fileSize(CustodyServiceOutputDocument document)
			throws CustodyServiceException {
		log.info("fileSize init");
		return size(document.getUri());
	}

	public void downloadSign(CustodyServiceOutputSign sign,
			OutputStream outputStream) throws CustodyServiceException {
		log.info("downloadSign init");
		download(sign.getUri(), outputStream);
		log.info("downloadSign init");

	}
```

```
	public BigDecimal signSize(CustodyServiceOutputSign sign)
			throws CustodyServiceException {
		log.info("fileSize end");
		return size(sign.getUri());
	}

	private BigDecimal size(String uri) throws CustodyServiceException {
		File file = new File(uri);
		int size = (int) file.length();
		return new BigDecimal(size);
	}

	private void download(String uri, OutputStream outputStream)
			throws CustodyServiceException {
		try {

			FileInputStream fileIn = new FileInputStream(uri);
			int read;
			byte[] buffer = new byte[Constants.BUFFER_SIZE];
			while ((read = fileIn.read(buffer)) > 0) {
				outputStream.write(buffer, 0, read);
			}
			fileIn.close();
		} catch (Exception e) {
			throw new CustodyServiceException(e.getMessage());
		}
	}
}
```

4 ENVÍO DE DOCUMENTOS POR REFERENCIA

A partir de la versión 3 de Port@firmas se ofrece la posibilidad de que los documentos se alojen en repositorios externos a Port@firmas mediante la creación de peticiones con envío de documentos por referencia. Dichos repositorios deben ofrecer a Port@firmas un protocolo de acceso a los documentos que custodian, denominado ENIDOCWS, el cual se adecua y alinea con los requisitos definidos por el Esquema Nacional de Interoperabilidad.

En la zona de administración de Port@firmas y para cada aplicación, se definirá su sistema de custodia:

- **Custodiado por Port@firmas**: no se admitirá el envío de documentos por referencia ya que todos los documentos deberán estar alojados en Port@firmas
- **Custodiado en repositorios externos**: solo se admitirá el envío de documentos por referencia. En este caso será necesario configurar adicionalmente la URL del servicio ENIDOCWS de acceso a los documentos custodiados por el repositorio externo, así como el usuario y la contraseña con los que Port@firmas se autenticará ante este servicio.

A través de los servicios web es posible crear peticiones con documentos alojado en este tipo de repositorio a través del método "**createRequestByReference**" publicado en el servicio de modificación perteneciente al servicio web v2.

Para invocar este método es necesario construir una lista de documentos, donde cada uno de ellos se genera a partir de:

- **Hash a firmar**: lo debe aportar la aplicación que envíe la petición a Port@firmas. Debe de ser codificado en hexadecimal y se genera con el algoritmo de hash SHA1 a partir del contenido del documento.
- **Algoritmo de Hash**: algoritmo usado para generar el anterior Hash. Por ahora sólo está disponible el algoritmo "SHA1".
- **Tamaño del fichero**
- **CSV del Documento**: debe corresponder con el Código de Verificación que se usará para la posterior verificación del documento. Éste deberá ser interpretable por la Herramienta Centralizada de Verificación (HCV):

  <5_CHARS_ID_REPOSITORIO><25_DOCUMENT_CSV>

  - 5 caracteres que identifiquen al repositorio en el que se encuentre el documento
  - 25 caracteres que identifiquen al documento

  Puede consultar más información acerca del **sistema HCV** en:

https://ws024.juntadeandalucia.es/ae/adminelec/areatecnica/herramientacentralizadadeverificacion

- **Datos de la referencia del documento**:

PROTOCOLO	PARÁMETROS DE REFERENCIA
ENIDOCWS	No es necesario indicar ningún parámetro extra. Se usará el **CSV indicado** como identificador del Documento.

En cualquier caso, si a través del cliente webservice se indica un protocolo incorrecto o no se indican todos los parámetros necesarios para la localización del fichero, Port@firmas devolverá un error identificando de qué error se trata.

⚠ A partir de la versión 3 de Port@firmas se ofrece la posibilidad de trabajar con Documentos ENI, los cuales están orientados a la Interoperabilidad de los Sistemas de información.

5 ACCIONES

Port@firmas ofrece la posibilidad de asociar una acción de retorno a una petición de firma para que se ejecute cuando dicha petición cambia de estado. Dicha funcionalidad permite a terceras aplicaciones tener actualizado en todo momento el estado de sus peticiones. De todas formas se recomienda que la aplicación cliente pregunte el estado de la petición mediante el ws ya que el envío de acciones puede fallar por URL incorrecta, permisos, etc.

5.1 Estados asociados

Las acciones van siempre ligadas a un estado, de tal forma que a una misma petición se pueden asociar varias acciones, una para ser ejecutada en cada cambio de estado. Los estados contemplados para las acciones web son:

- **LEIDO** – Se ejecuta cada vez que algún firmante lee la petición (pasa de estado nueva a leída).
- **FIRMADO** – Se ejecuta cada vez que un firmante definido firma la petición.
- **VISTOBUENO** – Se ejecuta cada vez que algún firmante realiza el Visto Bueno sobre la petición.
- **DEVUELTO** – Se ejecuta cuando algún firmante devuelve la petición (a diferencia de las anteriores solo se ejecuta una vez ya que cuando un firmante devuelve una petición, ésta pasa a estado devuelto para todos los firmantes).

5.2 Tipos

Las acciones deben ser de tipo web: se definen como una URL que el sistema llama remotamente mediante método GET de HTTP. El servidor sobre el que se despliega Port@firmas ha de tener conectividad con la dirección definida o bien puede hacer uso de un proxy para acceder a dicha máquina remotamente por el puerto indicado. Su identificador de tipo es WEB.

Si la ejecución es correcta, debe devolver un código HTTP 200 para indicarlo a Port@firmas. En caso de error deberá devolver otro código distinto, en cuyo caso el sistema realizará una serie de reintentos a través de su gestor de procesos asíncronos.

5.3 Parámetros

Las acciones tipo web ofrecen la posibilidad de sustituir parámetros antes de ejecutar la acción con sus valores correspondientes. Los parámetros que pueden ser empleados son los siguientes:

- **EDNI**: Identificador del firmante que realiza el cambio de estado.
- **ESTADO**: Nuevo estado al que ha pasado la petición.
- **HASHPET**: Hash de la petición.
- **EHASH**: Hash del documento (Sólo para peticiones de Port@firmas v1, ya que éstas iban asociadas a documentos y no a peticiones).
- **ASUNTO**: Asunto de la petición.

- **APLICACIÓN**: Aplicación asociada a la petición.
- **FCADUCIDAD**: Fecha de caducidad de la petición.
- **F_EST**: Fecha en que se produce el cambio de estado.
- **FIRMANTES**: Representación en XML de las líneas de firma y firmantes de la petición codificados en base64. La estructura sería la siguiente:

```
<Request>
      <SignLine>
            <Signer id="DNI_FIRMANTE/ID_PUESTO"></Signer>
      </SignLine>
</Request>
```

- **PET_CONTENT**: Texto de la petición.
- **TEXT**: Comentario del firmante en el cambio de estado.

A continuación se expone un ejemplo de acción web que hace uso de parámetros:

```
http://servidorAplicaciones:puerto/aplicacion/accion?
EHASH&EDNI&HASHPET&ESTADO
```

6 OBTENCIÓN DE INFORMES DE FIRMA

Port@firmas ofrece la posibilidad de descargar el/los informe/s de firmas de una determinada petición a través de una URL a la que se puede acceder conociendo determinados datos de una petición.

Dado que una petición puede contener varios documentos y por cada uno de ellos existe un informe de firma, para poder descargar un informe de firmas se necesitan tanto el hash de la petición como el del documento del cual se quiere obtener el informe. Ambos parámetros se identifican de la siguiente forma:

idDocument o **docHASH** – Hash del documento.

idRequest o **petHASH** – Hash de la petición.

La dirección desde la que se puede descargar el informe de firma es la siguiente, sustituyendo los campos marcados en rojo por sus valores correspondientes:

```
http://servidor[:puerto]/pfirmav3.1.1/informeFirma?
idRequest=hashPet&idDocument=hashDoc
```

Esta formar de recuperar el documento se considera obsoleta, y no se garantiza su funcionamiento en próximas versiones. En lugar de ello, se deberán descargar los informes de firma a través del servicio de consulta ofrecido por los Servicios Web V2. En todo caso, ha de considerarse que la confección de informes de firma es costosa computacionalmente y por tanto en la medida de lo posible se debe recabar de Port@firmas en horarios de escasa actividad del sistema.

7 CONSULTA EXTERNA DE PETICIONES

Otra funcionalidad que ofrece el sistema es la de consultar los datos de una petición a través de una URL externa. Accediendo a dicha URL se mostrará en el navegador una pantalla similar a la de detalle de petición en la que se muestran los datos básicos de la petición (asunto, referencia, texto y remitentes) así como el árbol completo de firmantes y la lista de documentos, ofreciendo la posibilidad de descargar los originales, las firmas y los informes de firma. También se podrá consultar desde esta pantalla el histórico de la petición, además de los comentarios, la información adicional y los documentos anexos si existiesen.

La dirección desde la que se puede acceder a dicha pantalla es la siguiente, sustituyendo el campo marcado en rojo por el hash de la petición a consultar:

http://servidorAplicaciones:puerto/pfirmav3.1.1/pf_comprobar_firma.jsp?peticion=hashPet

⚠ Se mantiene esta funcionalidad exclusivamente por motivos de compatibilidad con versiones previas, pero su uso se desaconseja ya que no se garantiza que próximas versiones de la herramienta conserven la funcionalidad.

8 FIRMA EN TRÁMITE

La firma en trámite es un servicio ofrecido por Port@firmas a terceras aplicaciones, para que éstas no tengan que implementar toda la compleja lógica de negocio relacionada con la firma electrónica. Port@firmas ofrece un punto de entrada vía web, la cual ofrece una interfaz de funcionalidad reducida, concebida únicamente para la firma de una determinada petición. Esta interfaz no ofrecerá funcionalidad de búsqueda, administración o redacción de nuevas peticiones, sino únicamente permitirá autenticarse al usuario y proceder a la firma de una petición determinada. Este interfaz web de Port@firmas puede ser embebido dentro de la aplicación que requiere hacer uso de la firma en trámite, a través de un marco o iFrame.

Para invocar la firma en trámite únicamente basta con saber el hash de la petición y acceder a la url que se muestra a continuación:

http://servidorAplicaciones:puerto/pfirmav3.1.1/external/external?sign=true&gotorequest=hashPet

El usuario iniciaría una solicitud a través de la aplicación X, la cual se encargaría de generar una serie de documentación a firmar por parte del usuario. La aplicación X generaría la petición de firma en Port@firmas a través de servicios web, adjuntando todos los documentos necesarios y obteniendo como resultado un código hash único vinculado a la petición. A través de este hash de petición, la aplicación X muestra al usuario el marco , iFrame o elemento web que contendrá la vista de la interfaz web de Port@firmas. El usuario realizará la firma a través de esa ventana de Port@firmas, que a su vez notificará a la aplicación de los eventos generados durante el proceso mediante las acciones web indicadas al crear la petición, o bien será la propia aplicación remitente la que tendrá que recabar de Port@firmas la información sobre el estado de la petición enviada, por ejemplo, si el usuario de esta aplicación indica mediante alguna opción que ha firmado o rechazado la firma de la petición en Port@firmas.

9 GLOSARIO

Término	Descripción
Apache CXF	Framework de servicios web.
MADEJA	Marco de Desarrollo de la Junta de Andalucía.
MAVEN	Herramienta para administración de proyectos.
JUnit	Framework de pruebas unitarias para clases Java.
AXIS	Framework de servicios web.
Eclipse	Entorno de desarrollo integrado (IDE)
ENIDOC - NTI	Norma Técnica de Interoperabilidad
HCV	Herramienta Centralizada de Verificación

10 ANEXO I. TIPOS DE DOCUMENTOS

Se enumera a continuación los posibles valores que puede adoptar el metadato "Tipo documental" que todo documento electrónico ha de tener vinculado conforme a lo establecido en la Norma Técnica de Interoperabilidad de Documento Electrónico.

Término	Descripción
TD01	Resolución
TD02	Acuerdo
TD03	Contrato
TD04	Convenio
TD05	Declaración
TD06	Comunicación
TD07	Notificación
TD08	Publicación
TD09	Acuse de recibo
TD10	Acta
TD11	Certificado
TD12	Diligencia
TD13	Informe
TD99	Otros

La versión 3.1.1 de Port@firmas requiere que las peticiones creadas indiquen un valor de entre los anteriores al establecer el tipo de cada uno de los documentos vinculados a una petición de firma. Por motivos exclusivamente de compatibilidad se aceptan adicionalmente los valores configurados en la administración de la herramienta, si bien las aplicaciones que hagan uso de estos otros valores configurados deben abandonar su uso y adaptarse a citada NTI de Documento Electrónico.