Manual de Integradores de viafirma platform

Integración con Java

v3.6 rev.03

ÍNDICE

CONTROL DE DOCUMENTO

Título:	Manual de Integradores de viafirma platform		
Asunto:	Integración con Java		
Versión:	v3.6 rev.03	**Fecha:**	31-01-2014
Código:		**Revisión anterior:**	08-08-2013
Idioma:		**Núm. Páginas:**	41

CONTROL DE CAMBIOS Y VERSIONES		
Fecha	**Versión**	**Motivo del Cambio**
24-06-13	v3.6	Primera versión para la v3.6 de viafirma platform. Respecto a la última revisión de la v3.5, se incorporan los siguientes cambios: • Nuevo punto 4.4.7 Validación de Certificados
02-07-13	v3.6 rev01	Nuevas funcionalidades incorporadas en los clientes v2.9.71 y v3.0.41 para viafirma platform v3.6.1 y superior: • Punto 4.2.2.3: nuevos params para firma PDF con Sello. • Punto 4.3.3: Ejemplo de firma PDF con sello. • Punto 4.4.1.1: Ejemplo generación PDF417.

08-08-13	v3.6 rev02	Nuevas funcionalidades incorporadas en los clientes v2.9.73 y v3.0.42 para viafirma platform v3.6.2 y superior:
		• 4.2.2.2 Nuevo param DIGITALIZED_SIGNATURE_FORMAT.
		• 4.2.2.5. Params para personalización del Applet
		  ○ filtro de numberUserId
		  ○ filtro de CAs
		• 4.2.2.4 Nuevo punto para params de anotaciones en PDF.

31-01-14	v3.6 rev03	Nuevas funcionalidades incorporadas para el cliente v2.9.86 compatibles con viafirma platform v3.6.5, principalmente relacionados con la gestión del stamper de firma PDF. Ver PolicyParams. • Nuevo punto 1.5 con referencia a la Javadoc publicada. • Actualización de los puntos 4.2.1 OptionalRequest y 4.2.2 PolicyParams, haciendo referencia en ambos caso a la javadoc con los detalles de los nuevos params incorporados.

1. INTRODUCCIÓN

1.1. Público Objetivo

Este documento tiene como misión facilitar a usuarios integradores la utilización de las librerías de cliente de viafirma platform para dotar a sistemas externos de las funcionalidades de autenticación, firma, verificación y custodia de documentos.

Está orientado a un perfil de desarrollador habituado a desarrollos JEE, por lo que el nivel de detalles de algunos aspectos tratados necesitan de un conocimiento previo en esta tecnología. También se requiere un mínimo conocimiento sobre el tratamiento de certificados digitales (X.509v3).

1.2. Requisitos Previos

Este manual de integración hace referencia a distintos recursos que serán necesarios para la puesta en marcha de los ejemplos descritos. Todos estos recursos están disponibles de forma gratuita en el portal para desarrolladores http://developers.viafirma.com , en la sección “descargas”.

1.3. Credenciales: API-KEY

Al mismo tiempo, y para el uso inmeidato de los servicios contra nuestro entorno centralizado serán necesarias unas credenciales para desarrollo, que serán proporcionadas también de forma gratuita desde http://developers.viafirma.com

1.4. Otra Documentación Relacionada

A pesar de que en este documento se explican las principales funcionalidades y servicios disponibles es recomendable tener a mano la javadoc de la versión del cliente JEE más reciente, diponible también en el portal de desarrolladores http://developers.viafirma.com

Al mismo tiempo, existen documentos específicos para cada tecnología soportada en la integración de servicios con viafirma platform, manuales de instalación de la propia plataforma y otra documentación que tal vez te pueda resultar de interés. Consulta de igual forma en el portal de desarrolladores y si no encuentras lo que buscas te ayudaremos.

1.5. Javadoc Relacionada

Este manual corresponde a la siguiente versión de Javadoc:

http://www.viafirma.com/download/clients/JEE/javadoc/javadoc_2_9_86/index.html

Revisar la última versión disponible.

2. CONFIGURACIÓN DEL SERVICIO

2.1. Init por defecto

A continuación se muestra la inicialización estándar del servicio.

```
ViafirmaClientFactory.init("urlViafirma","urlViafirmaWS","urlAplicacion",”api-key”,”api-password”);
```

- *urlViafirma(obligatorio)*: URL donde se encuentran los servicios OpenID/TokenConnector para la autenticación. Debe ser el nombre de máquina.
    - *Ejemplo:* https://services.viafirma.com/viafirma
- *urlViafirmaWS(obligatorio)*: URL donde se encuentran los **WebServices para firma**. En este caso es recomendable que se informe la IP y no el nombre de máquina.
    - *Ejemplo*: https://services.viafirma.com/viafirma
- *urlAplicación(opcional)*: URL a la que viafirma platform retornará el resultado de la operación (autenticación, firma, etc.).
    - *Ejemplo*: http://myapp.com/myServlet/
- *Credenciales;*
    - *Están compuestas por dos parámetros API-KEY y API-PASSWORD, que identificarán a la aplicación que desea integrar con los servicios de viafirma platform.*

- *El acceso a los servicios de aplicaciones que no informen estos parámetros en sus llamadas podrá ser rechazado en función del entorno al que se esté accediendo.*

2.2. Init para entornos con proxies y/o SSL

En entornos en los que la URL de retorno se pueda ver afectada por proxies o cambios en las cabeceras SSL, se recomendia el uso de un parámetro adicional que lo solventa, tal y como se explica a continuación.

```
ViafirmaClientFactory.init("urlViafirma","urlViafirmaWS","urlAplicacion",https://*,"api-key","api-password");
```

Esta máscara indica que si la URL de retorno no regresa con https, se sobreescribe y se fuerza a https con la máscara indicada.

También se usa este parámetro aicional cuando existe algún proxy (-proxy inverso por ej.-) que cambia la url.

2.2.1. Proxy con user/pass

En caso de usar proxy con credenciales, a partir del cliente 2.9.68 se incorpora el soporte para recuperar las propiedades http.proxyUser y http.proxyPass del sistema.

Estas propiedades NO son setteadas en el cliente de viafirma, el cual únicamente recupera esta información del sistema. En versiones anteriores a la 2.9.58 sólo se recuperaba las propiedades http.proxyHost y http.proxyPort.

3. GUÍA RÁPIDA

3.1. Introducción

Aunque viafirma platform ofrece todos sus servicios mediante métodos estándares de Servicios Web y OpenID, también disponemos de un cliente para JEE que permite de una forma muy sencilla integrar aplicaciones desarrolladas en esta tecnología.

En este apartado mostraremos cómo añadir las dependencias necesarias a un proyecto web Java para hacer uso de los diferentes servicios de autenticación, firma, custodia y verificación.

3.2. Dependencias

3.2.1. Maven: añadir dependencias

Viafirma platform está preparado para trabajar con Maven; en este tipo de proyectos sólo será necesario añadir la dependencia a viafirma-client de la siguiente manera:

```
<!-- Dependencias para el cliente viafirma con soporte de OpenID -->
<dependency>
    <groupId>org.viafirma</groupId>
    <artifactId>viafirma-client</artifactId>
    <version>[2.2.3,2.4.0)</version>
</dependency>
```

E indicar el repositorio adecuado: http://repositorio.viavansi.com/rep

3.2.2. Empotrar dependencias

Si el proyecto no está basado en Maven necesitaremos añadir manualmente los .jar que se incluyen en el directorio dependency dentro del distribuible de viafirma-client. En este caso se debe obtener el kit adecuado del tipo “all-in-one” el cual ya trae todas las dependencias empotradas.

3.3. Configuración del Cliente JEE

El cliente podrá ser inicializado de distintas formas, por ejemplo a través la inicialización de variables de contexto dentro de la aplicación que lo va a contener el cliente, tal y como se muestra en el siguiente ejemplo:

```
<!-- Configuracion de Viafirma -->
<Environment description="Configuracion del cliente de Viafirma"
name="CONFIG_VIAFIRMA_CLIENT" override="false" type="java.lang.String"
value="urlViafirma;urlViafirmaWS;urlAplicacion,api-key,api-pass"/>
```

O bien inicializando el cliente de forma explícita cuando se requiera, tal y como se muestra en el siguiente ejemplo:

```
/**
Configura el cliente de Viafirma:
@param urlViafirma: url pública en la que se encuentra viafirma platform (debe ser visible
por los usuarios finales)
@param urlViafirmaWS: url privada de viafirma para acceso a los WS)
@param urlAplicacion: url de la aplicación, parámetro opcional y utilizado por el skin en caso
de que no sea detectada automáticamente la URL)
*/
public synchronized static void init(String urlViafirma,String urlViafirmaWS,String
urlAplicacion) {
    Properties propiedades=new Properties();
    propiedades.setProperty(Constantes.PARAM_URL_PROVIDER_VIAFIRMA, urlViafirma);
    propiedades.setProperty(Constantes.PARAM_URL_CONECTOR_FIRMA_RMI,
    urlViafirmaWS+URLWS);
    if(!StringUtils.isEmpty(urlAplicacion)){
        propiedades.put(Constantes.PARAM_URL_APLICACION, urlAplicacion);
        propiedades.put("VIAFIRMA_CLIENT_APP_ID","ApiKey");
        propiedades.put("VIAFIRMA_CLIENT_APP_PASS","PassKey");
    }
    init(propiedades);
}
```

O bien de forma directa:

```
ViafirmaClientFactory.init(urlViafirma, urlViafirmaWS, urlAplicacion, apiKey, apiPass);
```

3.4. Ejemplo de Autenticación

A modo de ejemplo básico vamos a crear una jsp que inicialice el cliente de viafirma platform y permita al usuario iniciar el proceso de autenticación pulsando en un enlace.

```
<%@page import="org.viafirma.cliente.ViafirmaClientFactory"%>
<%@page import="org.viafirma.cliente.ViafirmaClient"%>
<body>
<%
if (!ViafirmaClientFactory.isInit()) {
    // Configuración básica del cliente.
     ViafirmaClientFactory.init("http://services.viafirma.com/viafirma",
    "http://services.viafirma.com/viafirma");
    }
if(request.getParameter("autenticar")!= null) {
    ViafirmaClient viafirmaClient = ViafirmaClientFactory.getInstance();
    // Iniciamos la autenticación indicando la uri de retorno.
    viafirmaClient.solicitarAutenticacion(request,
   response,"/viafirmaClientResponseServlet");
   }
%>
<p><a href="?autenticar=true">Solicitar autenticación</a></p>
</body>
```

Cuando el usuario pulse sobre el enlace "Solicitar autenticación" el usuario será redirigido a viafirma platform, donde se le solicitará su certificado digital. Viafirma platform validará y tratará el certificado del cliente y retornará el resultado de la autenticación a la aplicación cliente que estamos desarrollando. En la jsp de ejemplo le indicaremos a viafirma platform que la url de retorno (donde viafirma platform debe mandarnos el resultado de la autenticación) es **/viafirmaClientResponseServlet** .

En esta ubicación la aplicación que estamos integrando deberá tener un un servlet escuchando la respuesta que nos retornará viafirma platform para su correspondiente procesado.

3.4.1. Procesar respuesta de Autenticación

Para procesar la respuesta de viafirma platform nos ayudaremos de un servlet que tendremos a la eschucha en la aplicación, y que deberá extender de **org.viafirma.cliente.ViafirmaClientServlet**, y tendrá que sobrescribir los siguientes métodos:

- **authenticateOK**; viafirma platform ha recuperado correctamente los datos del certificado digital del usuario final y nos lo devuelve para que nuestra aplicación los procese y decida qué hacer con ellos.

- **cancel**; método invocado por viafirma platform en su respuesta cuando el usuario final ha cancelado el proceso voluntariamente (pulsó sobre el botón "cancelar" del applet).

- **error**; viafirma platform ha lanzado algún error y la autenticación no ha podido ser completada con éxito. Nuestra aplicación procesará el mensaje de error y poder continuar con la lógica necearia. Ej. "CA no reconocida, certificado revocado, certificado caducado, etc.".

En la respuesta recibida, se invocará al método ViafirmaClientResponse, que se encuentra dentro del directorio Viafirma. Este método será sobrescrito para implementar la lógica de negocio deseada con los datos recuperados del certificado digital, los cuales vendrán contenidos en el objeto **UsuarioGenericoViafirma**.

En el siguiente ejemplo, si la autenticación ha sido correcta, guardamos los datos del usuario en la request y redireccionamos al usuario final a una página de destino.

```
public class ViafirmaClientResponseServlet extends ViafirmaClientServlet {

    @Override
    public void authenticateOK(UsuarioGenericoViafirma usuario,HttpServletRequest request,
     HttpServletResponse response) {
        // Lógica específica de cada aplicación para gestionar el resultado de la autenticación
                try {
         request.setAttribute("usuarioAutenticado", usuario);
         request.getRequestDispatcher("/resultadoAutenticacion.jsp").forward(request, response);
         } catch (ServletException e) {
                    e.printStackTrace();
         } catch (IOException e) {
                    e.printStackTrace();
    }
```

```
}

    @Override
    public void cancel(HttpServletRequest request, HttpServletResponse response) {
        // Gestión de cancelación del usuario al autenticar o firmar
        request.setAttribute("error", "El usuario ha cancelado la autenticación");
        request.getRequestDispatcher("/resultadoAutenticacion.jsp").forward(request,
        response);
    }

    @Override
    public void error(CodigoError codError, HttpServletRequest request, HttpServletResponse
    response) {
        // Gestión de error al autenticar o firmar
        request.setAttribute("codError", codError);
        request.getRequestDispatcher("/resultadoAutenticacion.jsp").forward(request,
        response);
    }
}
```

3.5. Ejemplo de Firma

A continuación se muestra un ejemplo de jsp donde preparamos todo lo necesario antes de la invocación del servicio de firma.

```
<%@page import="org.viafirma.cliente.ViafirmaClientFactory"%>
<%@page import="org.viafirma.cliente.ViafirmaClient"%>
<%@page import="org.viafirma.cliente.firma.TypeFile"%>
<%@page import="org.viafirma.cliente.firma.TypeFormatSign"%>
<%@page import="org.apache.commons.io.IOUtils"%>
<body>
<%
if (!ViafirmaClientFactory.isInit()) {

    // Configuración básica del cliente.
     ViafirmaClientFactory.init("http://services.viafirma.com/viafirma",
    "http://services.viafirma.com/viafirma","http://localhost/myApp/","api-key","api-
pass");
}

if(request.getParameter("firmar")!= null){
```

```
    //Instanciamos al cliente
    ViafirmaClient viafirmaClient = ViafirmaClientFactory.getInstance();

    // Definimos la política de firma deseada
     Policy policy = new Policy();
     policy.setTypeFormatSign(TypeFormatSign.XADES_EPES_ENVELOPED);
     policy.setTypeSign(TypeSign.ENVELOPED);

     //Creamos Documento
     Documento documento = new Documento("example", datosAFirmar, TypeFile.PDF,
    TypeFormatSign.PDF_PKCS7);


    // Registramos el documento que deseamos firmar, obteniendo un id temporal.
    String idTemporal= viafirmaClient.prepareSignWithPolicy(Policy policy, Documento
documento);

    //Solicitamos la firma
    viafirmaClient.solicitarFirma(idTemporal,request, response,
    "/viafirmaClientResponseServlet");
}
%>
<p><a href="?firmar=true"> Firmar el documento </a></p>
</body>
```

Del código de ejemplo anterior destacamos lo siguiente:

- Preparar la firma; el "prepareFirma" devuelve el id temporal que necesitaremos para solicitar la firma. Este id no coincide con el código de firma generado tras finalizar el proceso de firma.

- Preparar la política de firma; a través del objeto Policy podremos setear las características de la firma que vamos a solicitar. Este objeto es explicado en profundidad en capítulos posteriores.

- Solicitar la firma; el último paso consiste en solicitar la firma con el método solicitarFirma.

3.5.1. Procesar Respuesta de Firma

Al igual que para el proceso de autenticación, para procesar la respuesta de nos ayudaremos del servlet que tendremos a la eschucha en la aplicación, y que deberá extender de `org.viafirma.cliente.ViafirmaClientServlet`, y que en este caso tendrá que sobrescribir el siguiente método:

- **signOK**; viafirma platform ha firmado correctamente y nos devuelve en la response el objeto "firma" para que nuestra aplicación los procese y continúe con la lógica de negocio correspondiente.

```
[…]
        @Override
        public void signOK(FirmaInfoViafirma arg0, HttpServletRequest arg1,
        HttpServletResponse arg2) {
                // Lógica específica de cada aplicación para gestionar el resultado de la
                // firma
                request.setAttribute("resultado", firma);
                try {
                request.getRequestDispatcher("/resultadoFirma.jsp").forward(request,
                response);
                } catch (ServletException e) {
                e.printStackTrace();
                } catch (IOException e) {
                e.printStackTrace();
                }
        }

        @Override
            public void cancel(HttpServletRequest request, HttpServletResponse response) {
                // el usuario final canceló el proceso de firma
                request.setAttribute("error", "Cancelado por el usuario");
                request.getRequestDispatcher("/resultadoAutenticacion.jsp").forward(request,
                response);
            }

            @Override
            public void error(CodigoError codError, HttpServletRequest request,
        HttpServletResponse
```

```
            response) {
                // Gestión de error al autenticar o firmar
                request.setAttribute("codError", codError);
                request.getRequestDispatcher("/resultadoAutenticacion.jsp").forward(request,
                response);
            }

[…]
```

3.6. Look & Feel de la página de integración

La página donde se solicita el certificado digital al usuario final reside en viafirma platform. Sin embargo, a través de CSS podremos conseguir que el usuario no aprecie un cambio de interfaz, de forma que el salto de la aplicación a viafirma platform parezca transparente a nivel estético, manteniendo en todo momento el look & feel de la aplicación original.

Debido a las amplias capacidades de abstracción de viafirma platform, se permite al integrador modificar la capa cliente al gusto de sus necesidades. Esta operación será realizada mediante hojas de estilo (CSS). Simplemente con colocar un fichero llamado "viafirmaStyle.css" en el directorio raíz de la aplicación, viafirma platform automáticamente se visualizará con el aspecto configurado por esta hoja de estilo.

En el **ANEXO** se describe con detalles el funcionamiento de esta CSS y se muestran algunos ejemplos reales de integración.

4. USO PRÁCTICO DEL CLIENTE JAVA

4.1. Firma con Policy

Viafirma platform permite la creación de formatos de firma y su configuración en el momento de solicitarlas al servidor a través de políticas de firma.

Para ello usaremos el objeto Policy, su nomenclatura java completa es org.viafirma.cliente.firma.Policy, y los nuevos métodos en los que podemos usar el objeto Policy son los siguientes:

```
• public String prepareSignWithPolicy(Policy policy, Documento docume
throws InternalException

• public String signByServerWithPolicy(Policy policy, Documento docume
String alias, String password) throws InternalException
```

4.2. Objeto Policy

El objeto Policy permite configurar algunas opciones de la firma a realizar de un modo más sencillo y potente que utilizando otros de nuestros métodos de firma. Aún así los métodos que usan Policy no sustituyen a la totalidad del resto de métodos, que para algunos casos seguirán siendo usados.

A continuación un ejemplo sencillo de la configuración de firma a través de policy:

```
Policy policy = new Policy();
policy.setTypeFormatSign(TypeFormatSign.XADES_EPES_ENVELOPED);
policy.setTypeSign(TypeSign.ENVELOPED);
```

Como se puede ver en el recorte de código anterior lo básico de un Policy será el formato de firma y el tipo de firma: el formato de firma podrá ser cualquiera de los que se encuentran en el enumerado org.viafirma.cliente.firma.TypeFormatSign, en cambio el tipo de firma por el

momento deberá de usarse siempre TypeSign.ENVELOPED, si se necesitase otro tipo se deberá consultar previamente para comprobar compatibilidades formato y tipo. Estos dos parámetros del Policy serán siempre obligatorios.

Además de los métodos **setTypeFormatSign** y **setTypeSign**, el objeto Policy dispone de dos métodos que permiten realizar configuraciones de firma más avanzadas.

- setOptionalRequest(List<String> optionalRequest)
- setParameters(Map<String, String> parameters)

y para lo que nos ayudaremos de los enumerados correspondientes y que se detallan a continuación:

4.2.1. Enum OptionalRequest

Nuestro objeto Policy puede ser seteado con distintos parámetros opcionales que están enumerados en OptionalRequest. A continuación se listan las funcionalidades asociadas al conjunto de optionalRequest disponibles, y que podrán consultarse en detalle en la última **javadoc** publicada y referenciada en el capítulo **1.5 "**Javadoc Relacionada**"** de este manual.

- OptionalRequest.PEM_X509
- OptionalRequest.CMS:
- OptionalRequest.DOCUMENT_HASH
- OptionalRequest.SKIP_VALIDATION_FAIL
- OptionalRequest.AUTO_SEND

4.2.2. Enum PolicyParams

Nuestro objeto Policy puede ser seteado con distintos parámetros opcionales que están enumerados en PolicyParams. A continuación se listan las funcionalidades asociadas al conjunto de params disponibles, y que podrán consultarse en detalle en la última **javadoc** publicada y referenciada en el capítulo **1.5** "Javadoc Relacionada" de este manual.

4.2.2.1. Params para formatos de firma

- DETACHED_REFERENCE_URL: Url pública del documento original - Type: String

4.2.2.2. Params para firma digitalizada

Ver en la JAVADOC los PolicyParams **DIGITALIZED_SIGN*** , asociados todos ellos a la gestión del stamper de firma en PDF.

La última javadoc publicada puede consultarse en el capítulo **1.5** "Javadoc Relacionada" de este manual.

4.2.2.3. Params la gestión del stamper de firma en PDF

Ver en la JAVADOC los PolicyParams **DIGITAL_SIGN_*** , asociados todos ellos a la gestión del stamper de firma en PDF.

La última javadoc publicada puede consultarse en el capítulo **1.5** "Javadoc Relacionada" de este manual.

4.2.2.4. Params para la gestión de Anotcaciones en PDF

Ver en la JAVADOC los PolicyParams **PDF_ANNOTATION_*** , asociados todos ellos a la gestión del stamper de firma en PDF.

La última javadoc publicada puede consultarse en el capítulo **1.5** "Javadoc Relacionada" de este manual.

4.2.2.5. Params para firma con PKCS#1

- CALLBACK_URL: Internal comunication callback url between viafirma and other apps.

La última javadoc publicada puede consultarse en el capítulo **1.5** "Javadoc Relacionada" de este manual.

4.2.2.6. Params para personalización del Applet

Ver en la JAVADOC los siguientes PolicyParams:

- APPLET_STYLE
- CLIENT_LOCALE
- FILTER_CA_NAME
- FILTER_NUMBER_USER_ID

La última javadoc publicada puede consultarse en el capítulo **1.5** "Javadoc Relacionada" de este manual.

4.2.3. TypeFormatSign

El enumerado org.viafirma.cliente.firma.TypeFormatSign contiene todos los formatos de firma soportados por viafirma platform:

1. CAdES_A: CMS Advanced Electronic Sicnature (CAdES).
2. CAdES_BES: CMS Advanced Electronic Sicnature (CAdES).
3. CAdES_C: CMS Advanced Electronic Sicnature (CAdES).
4. CAdES_EPES: CMS Advanced Electronic Sicnature (CAdES).
5. CAdES_T: CMS Advanced Electronic Sicnature (CAdES).
6. CAdES_XL: CMS Advanced Electronic Sicnature (CAdES).
7. CMS: Formato de firma para CMS (comportamiento por defecto).
8. CMS_ATTACHED: Formato de firma para CMS (Attached).
9. CMS_DETACHED: Formato de firma para CMS (Detached).
10. DIGITALIZED_SIGN: Firma digitalizada
11. PAdES_BASIC: PAdES: PAdES-PK7 form (PAdES Basic, based on ISO 32000-1).
12. PAdES_BES: PAdES: PDF Advanced Electronic Signature (ETSI 102 778).
13. PAdES_EPES: PAdES: PDF Advanced Electronic Signature (ETSI 102 778).
14. PAdES_LTV: PAdES: PDF Advanced Electronic Signature (ETSI 102 778).
15. PDF_PKCS7: Firma formato CMS para PDF.
16. PDF_PKCS7_T: Firma formato CMS para PDF con Timestamp.
17. XADES_A_ENVELOPED: Formato XADES-A Necesita tener TimeStamp habilitado en servidor
18. XADES_EPES_ENVELOPED: Formato XADES 1.2.2_ 1.3.2 ETSI TS 101 903.
19. XADES_T_ENVELOPED: Formato XADES-T Necesita tener TimeStamp habilitado en servidor
20. XADES_XL_ENVELOPED: Formato XADES-XL Necesita tener TimeStamp habilitado en servidor
21. XMLDSIG: Firma en formato XMLDSIG (firma xml simple)
22. XMLSIG_ENVELOPING: formado XMLSignature basado en formato original de OpenOces.
23. PKCS1_SHA1_RSA: Raw Sign.

4.3. Ejemplo de Firmas con Policy

4.3.1. Firma Simple

(aplicable a cualquier formato de firma válido)

```
ViafirmaClient viafirmaClient = ViafirmaClientFactory.getInstance();
// Datos documento a firmar
byte[] datosAFirmar =
  IOUtils.toByteArray(getClass().getResourceAsStream("/exampleSign.pdf"));

//Creamos un policy simple
Policy policy = new Policy();
policy.setTypeFormatSign(TypeFormatSign.PAdES_BASIC);
//policy.setTypeFormatSign(TypeFormatSign.XADES_EPES_ENVELOPED);
policy.setTypeSign(TypeSign.ATTACHED);

//Creamos Documento
Documento documento = new Documento("example", datosAFirmar, TypeFile.
TypeFormatSign.PDF_PKCS7);

//Preparamos la firma
String idFirma = viafirmaClient.prepareSignWithPolicy(policy, documento);

// Iniciamos la firma enviando al usuario a Viafirma indicando la uri
retorno.
viafirmaClient.solicitarFirma(idFirma,          request,          respo
"/viafirmaClientResponseServlet");
```

4.3.2. Firma Digitalizada

```
//Cliente
ViafirmaClient viafirmaClient = ViafirmaClientFactory.getInstance();

//Formato de firma
TypeFormatSign format = TypeFormatSign.DIGITALIZED_SIGN;

// Datos documento a firmar y logo
byte[] datosAFirmar =
  IOUtils.toByteArray(getClass().getResourceAsStream("/exampleSign.pdf"));

byte[] logoStamp =
  IOUtils.toByteArray(this.getClass().getResourceAsStream("/logoStamp.jpg"))

//Documento
Documento documento =
  new Documento("prueba.pdf",datosAFirmar,TypeFile.PDF,         format);

//Certificado a utilizar para el cifrado de los datos biometricos
String pem =
  IOUtils.toString(this.getClass().getResourceAsStream("/xnoccio.pem"));

//Policy
Policy policy = new Policy();
policy.setTypeSign(TypeSign.ATTACHED);
//Indica el formato (en este caso digitalizada)
policy.setTypeFormatSign(format);

//Seteo de parametros
//Indica el color de fondo de la pantalla de firma (para app movil)
obligatorio)
policy.addParameter(PolicyParams.DIGITALIZED_SIGN_BACK_COLOUR.getKey(),
"#0000FF");
//Indica el color de la firma de la pantalla (para app movil) (No obligato

policy.addParameter(PolicyParams.DIGITALIZED_SIGN_COLOUR.getKey(), "#FF0000"
//Indica el texto de ayuda que aparece en la pantalla (para app movil)
obligatorio)
policy.addParameter(PolicyParams.DIGITALIZED_SIGN_HELP_TEXT.getKey(), "Texto
ayuda aportado por el integrador");
//Logo a mostrar (para app movil) (No obligatorio)
policy.addParameter(PolicyParams.DIGITALIZED_SIGN_LOGO.getKey(), logoStamp);
//Rectangulo donde se fija la firma (No obligatorio)
policy.addParameter(PolicyParams.DIGITALIZED_SIGN_RECTANGLE.getKey(),
Rectangle(400,60,160,120));
//Biometric alias - pass son utilizados para firmar los datos biometrico
servidor (el alias debe existir en el servidor) (No obligatorio)
policy.addParameter(PolicyParams.DIGITALIZED_SIGN_BIOMETRIC_ALIAS.getKey(),
"xnoccio");
policy.addParameter(PolicyParams.DIGITALIZED_SIGN_BIOMETRIC_PASS.getKey(),
"12345");
```

```
//Clave publica en formato pem con la que cifrar los datos biometricos, si n
indica no se cifran (No obligatorio)
policy.addParameter(PolicyParams.DIGITALIZED_SIGN_BIOMETRIC_CRYPTO_PEM.getKe
pem);
//Pagina donde insertar la firma, -1 para la ultima pagina, si no se indica
móviles se permitirá seleccionar la pagina/s manualmente, en Topaz se pondr
la última página
policy.addParameter(PolicyParams.DIGITALIZED_SIGN_PAGE.getKey(), -1);
//Alias/Pass de certificado en servidor para firmar el PDF con certifi
electrónico en servidor (No obligatorio)
policy.addParameter(PolicyParams.DIGITALIZED_SIGN_ALIAS.getKey(), "xnoccio")
policy.addParameter(PolicyParams.DIGITALIZED_SIGN_PASS.getKey(), "12345");

// Indicamos a la plataforma que deseamos firmar el fichero
String idFirma=viafirmaClient.prepareSignWithPolicy(policy, documento);
// Solicitamos la firma
viafirmaClient.solicitarFirma(idFirma,request,response,
"/viafirmaClientResponseServlet");
```

4.3.3. Firma PDF con Sello

Ejemplo con estampado de QRCode, Barcode y Texto. En el ejemplo de integración facilitado con la JDK se pueden consultar más ejemplos.

```
        ConfigureUtil.init();


        ViafirmaClient viafirmaClient = ViafirmaClientFactory.getInstance();

// Datos documento a firmar
byte[] datosAFirmar =
IOUtils.toByteArray(getClass().getResourceAsStream("/exampleSign.pdf"));

byte[] logoStamp =
IOUtils.toByteArray(getClass().getResourceAsStream("/stamperWatermark.png"));
//Creamos el documento
Documento doc = new
Documento("exampleSign.pdf",datosAFirmar,TypeFile.PDF,TypeFormatSign.PAdES_BA
SIC);

//Seteamos la politica
Policy policy = new Policy();

        policy.setTypeFormatSign(TypeFormatSign.PAdES_BASIC);

        policy.setTypeSign(TypeSign.ATTACHED);

        policy.addParameter(PolicyParams.DIGITAL_SIGN_PAGE.getKey(), "1");
```

```
        policy.addParameter(PolicyParams.DIGITAL_SIGN_RECTANGLE.getKey(), new
org.viafirma.cliente.vo.Rectangle(40,10,550,75));

        policy.addParameter(PolicyParams.DIGITAL_SIGN_STAMPER_HIDE_STATUS.getKe
y(), "true");

        //policy.addParameter(PolicyParams.DIGITAL_SIGN_IMAGE_STAMPER.getKey(),
logoStamp);

        policy.addParameter(PolicyParams.DIGITAL_SIGN_STAMPER_TEXT.getKey(),
"Firmado por [CN] con DNI [SERIALNUMBER]\ntrabajador de [O] en el
departamento de [OU]");

        policy.addParameter(PolicyParams.DIGITAL_SIGN_STAMPER_TYPE.getKey(),
"QR-BAR-H");

// Indicamos a la plataforma que deseamos firmar el fichero
String idFirma = viafirmaClient.prepareSignWithPolicy(policy, doc);

// Iniciamos la firma enviando al usuario a Viafirma indicando la uri de
retorno.
viafirmaClient.solicitarFirma(idFirma,request,
response,"/viafirmaClientResponseServlet");
```

4.3.4. Firma XAdES Detached

```
ViafirmaClient viafirmaClient = ViafirmaClientFactory.getInstance();

// Datos documento a firmar
byte[] datosAFirmar =
  IOUtils.toByteArray(getClass().getResourceAsStream("/exampleSign.pdf"));
//Creamos el documento
Documento doc = new Documento("pruebaXMLDsig.xml",datosAFirmar,TypeFile.
TypeFormatSign.XADES_EPES_ENVELOPED);

//Seteamos la politica
Policy pol = new Policy();
pol.setTypeFormatSign(TypeFormatSign.XMLDSIG);
pol.setTypeSign(TypeSign.DETACHED);

//Parametros de politica
Map<String,String> params = new HashMap<String, String>();
params.put(PolicyParams.DETACHED_REFERENCE_URL.getKey(),
"http://viafirma.com/download/exampleSign.pdf");
pol.setParameters(params);

// Indicamos a la plataforma que deseamos firmar el fichero
String idFirma = viafirmaClient.prepareSignWithPolicy(pol, doc);

// Iniciamos la firma enviando al usuario a Viafirma indicando la uri
```

```
   retorno.
viafirmaClient.solicitarFirma(idFirma,request,response,
"/viafirmaClientResponseServlet");
```

Como se observa en los ejemplos para preparar la firma se utiliza un método que recibe la Policy como parámetro. Como dijimos antes los métodos del API de viafirma que permiten recibir un objeto Policy como parámetro son los siguientes:

```
   /**
    * Envía el fichero que deseamos firmar y devuelve un identificador tempora
    * ( No es el identificador final de la firma ).
    * @param Policy objeto en el que se definen los parametros de firma
    * @param documento En este objeto definimos los datos del fichero, su
    * nombre, el tipo de fichero…
    * @throws InternalException
    */
public String prepareSignWithPolicy(Policy policy, Documento documento)
throws InternalException
```

Tras este método, igual que en los demás métodos de preparacíon de firma, es necesario llamar al método `solicitarFirma(String idFirma, HttpServletRequest request, HttpServletResponse response)` para requerir la firma y que viafirma direccione al usuario hacia la página del applet.

```
   /**
    * Firma los datos utilizando un certificado almacenado en el servidor. Par
    * la utilizacion de este metodo es necesario tener un certificado de
    * usuario dentro del Cacert de Java. Nota: Este método no requiere
    * intervencion del usuario.
    * @param policy Objeto en el que se definen los parametros de firma
    * @param documento En este objeto definimos los datos del fichero, su
    * nombre, el tipo de fichero…
    * @param alias  Nombre del alias utilizado para realizar la firma.
    * @param password     Password del alias utilizado para realizar la firma.
    * @return El código de firma asignado a este documento firmado.
    * @throws InternalException  Problemas al realizar la firma o al conectar
    * con el servidor.
    */
public String signByServerWithPolicy(Policy policy, Documento documento,
String alias, String password) throws InternalException
```

4.3.5. Firma en Servidor

```
   ViafirmaClient viafirmaClient = ViafirmaClientFactory.getInstance();
   // Datos documento a firmar
   byte[] datosAFirmar =
     IOUtils.toByteArray(getClass().getResourceAsStream("/exampleSign.pdf"));

   //Creamos un policy simple
   Policy policy = new Policy();
   policy.setTypeFormatSign(TypeFormatSign.PAdES_BASIC);
   //policy.setTypeFormatSign(TypeFormatSign.XADES_EPES_ENVELOPED);
   policy.setTypeSign(TypeSign.ATTACHED);

   //Creamos Documento
   Documento documento = new Documento("example", datosAFirmar, TypeFile.
   TypeFormatSign.PDF_PKCS7);

   //Definimos alias y pass del certificado instalado en servidor
   String alias = "alias";
   String pass = "pass";

   // Firmamos con el alias/pass indicados
String idFirma = viafirmaClient.signByServerWithPolicy(policy,documento,
alias, pass);
```

Como se observa en los ejemplos para preparar la firma se utiliza un método que recibe la Policy como parámetro. Como se comentó anteriormente los métodos del API de viafirma platform que permiten recibir un objeto Policy como parámetro son los siguientes:

```
   /**
    * Envía el fichero que deseamos firmar y devuelve un identificador tempora
    * ( No es el identificador final de la firma ).
    * @param Policy objeto en el que se definen los parametros de firma
    * @param documento En este objeto definimos los datos del fichero, su
    * nombre, el tipo de fichero…
    * @throws InternalException
    */
public String prepareSignWithPolicy(Policy policy, Documento documento)
throws InternalException
```

Tras este método, igual que en los demás métodos de preparacíon de firma, es necesario llamar al método `solicitarFirma(String idFirma, HttpServletRequest request, HttpServletResponse response)` para requerir la firma y que viafirma direccione al usuario hacia la página del applet.

```
    /**
     * Firma los datos utilizando un certificado almacenado en el servidor. Par
     * la utilizacion de este metodo es necesario tener un certificado de
     * usuario dentro del Cacert de Java. Nota: Este método no requiere
     * intervencion del usuario.
     * @param policy Objeto en el que se definen los parametros de firma
     * @param documento En este objeto definimos los datos del fichero, su
     * nombre, el tipo de fichero…
     * @param alias  Nombre del alias utilizado para realizar la firma.
     * @param password     Password del alias utilizado para realizar la firma.
     * @return El código de firma asignado a este documento firmado.
     * @throws InternalException  Problemas al realizar la firma o al conectar
     * con el servidor.
     */
public  String  signByServerWithPolicy(Policy  policy,  Documento  documento,
String alias, String password) throws InternalException
```

4.3.6. Firma en formato PKCS#1

A partir de la versión v3.5.4 de viafirma platform, se incorpora el formato de firma PKCS#1 que permitirá generar firmas a bajo nivel.

Compatibilidad	
Recurso	Versión Compatible
Servidor viafirma platform	v3.5.4_DEV ó superior.
Java viafirma client	• viafirma-cliente-2.9.67 ó superior. • viafirma-client-3.0.35 ó superior.
Java Kit de Desarrolo	Versión 2.9.67 ó superior.

Este formato de firma permite generar firmas a bajo nivel enviando directamente el documento a firmar a viafirma platform o implementando un **callback** que sea el responsable de generar los datos a firmar.

El uso de este formato de firma se podrá implementar de dos formas:

- Formato de firma **PKCS1_SHA1_RSA sin indicar el callback** (en este caso la operación de firma se solicita como en el resto de formatos)

- Formato de firma **PKCS1_SHA1_RSA con callback.**

4.3.6.1. Ejemplo de uso con callback:

```
ViafirmaClient viafirmaClient = ViafirmaClientFactory.getInstance();

Documento documento=new Documento();
documento.setNombre("test.txt");
documento.setDatos("".getBytes());
documento.setTipo(TypeFile.bin);
documento.setTypeFormatSign(TypeFormatSign.PKCS1_SHA1_RSA);

Policy policy=new Policy();
policy.setTypeFormatSign(TypeFormatSign.PKCS1_SHA1_RSA);
policy.setTypeSign(TypeSign.DETACHED);
String url=ConfigureUtil.getCallbackUrl();
policy.addParameter(PolicyParams.CALLBACK_URL.getKey(),url );
String idTemp=ViafirmaClientFactory.getInstance().prepareSignWithPolicy(policy, documento);

// Iniciamos la firma enviando al usuario a Viafirma indicando la uri de retorno.
viafirmaClient.solicitarFirma(idTemp,request, response,"/viafirmaClientResponseServlet");
```

Se puede encontrar un ejemplo en /example/firmaUsuario/firmarSha1.jsp

En los puntos **4.2.3 y 4.2.2.4** se explican los tipos de formatos contenidos en el ENUM así como los posibles PARAMS permitidos respectivamente.

4.3.6.2. Ejemplo de implementación del callback:

El callback será invocado durante el proceso de firma para conocer los datos a firmar, viafirma enviará en el callback el certificado digital seleccionado por el usuario y el identificador temporal del documento.

Ejemplo de url:

http://localhost:8080/ejemploViafirma/ViafirmaCallbackServlet?documentId=P6D4LQ2961363781422399&appKey=undefined&rnd=-8920485755815925821

Ejemplo de implementación **doPost**:

```
/**
         * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)
         */
        protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
                String documentId=request.getParameter("documentId");
                //String appKey=request.getParameter("appKey");
                byte [] data=IOUtils.toByteArray(request.getInputStream());
                String pem=new String(data);
                System.out.println("Data PEM:"+ pem); // cert

                byte [] dataToSign=getDataToSign(request,documentId);
                response.getOutputStream().write(dataToSign);// data to sign

        }
```

Se puede encontrar un ejemplo completo en:

com.viafirma.examples.util.ViafirmaCallbackServlet

4.3.7. Firma recibiendo un HASH

A continuación se explican dos ejemplos de firma en **cliente** y firma en **servidor** donde se recibe el hash del documento original en lugar del propio documento. Este tipo de firma está soportado para los formatos **XAdES** y **CAdES**.

Firma en cliente (XAdES):

```
//Obtenemos cliente de viafirma
ViafirmaClient viafirmaClient = ViafirmaClientFactory.getInstance();

//Creamos Policy: el TypeSign debe ser 'DETACHED'
Policy policy = new Policy();
policy.setTypeFormatSign(TypeFormatSign.XADES_EPES_ENVELOPED);
policy.setTypeSign(TypeSign.DETACHED);

//Creamos el Documento: el TypeFile debe ser 'hash'
Documento documento = new Documento("hashSigned", hashOriginal.getBytes(),
TypeFile.hash, TypeFormatSign.XADES_EPES_ENVELOPED);

// Indicamos a la plataforma que deseamos firmar el fichero
String idFirma = viafirmaClient.prepareSignWithPolicy(policy, documento);

// Iniciamos la firma enviando al usuario a Viafirma indicando la uri de
retorno.
viafirmaClient.solicitarFirma(idFirma, request,
response,"/viafirmaClientResponseServlet");
```

Firma en servidor (XAdES):

```
//Creamos Policy: el TypeSign debe ser 'DETACHED'
Policy policy = new Policy();
policy.setTypeFormatSign(TypeFormatSign.XADES_EPES_ENVELOPED);
policy.setTypeSign(TypeSign.DETACHED);

//Creamos el Documento: el TypeFile debe ser 'hash'
Documento documento = new Documento("hashSigned", hashOriginal.getBytes(),
TypeFile.hash, TypeFormatSign.XADES_EPES_ENVELOPED);

// Iniciamos la firma enviando alias y pass
String
idFirma=ViafirmaClientFactory.getInstance().signByServerWithPolicy(policy,
documento, alias, pass);
```

4.4. Otros Métodos de Utilidad

4.4.1. QR-Code y Barcode:

Al firmar un documento, viafirma platform permite generar un justificante con códigos gráficos que facilitan el seguimiento y trazabilidad de la operación.

Para generarlo, utilizamos el método **buildInfoQRBarCode** que recibe el código de firma y devuelve un byte array con una imagen en formato PNG que representa los siguientes datos:

- datos del firmante firmante
- código de firma
- fecha y la hora de la firma
- permalink

```
// Recuperamos el codigo de la firma realizada
byte[] resguardo = viafirmaClient.buildInfoQRBarCode(codFirma);
```

Se puede encontrar una muestra funcional en la aplicación de ejemplo en [AplicacionDeEjemplo]/examples/utils/generarQRBarCode.jsp

4.4.1.1. Generación de PDF-417

A partir de la versión 3.6.1 de viafirma platform se incluye la opción de generar firmas en PDF con resguardos según el formato PDF417. En la aplicación de ejemplo entregada junto al SDK existen ejemplos de uso: **/examples/policy/firmarSelloPDF417.jsp**

4.4.2. Métodos de verificación

4.4.2.1. Verificación de un documento firmado

Con **checkDocumentSigned** se comprueba la validez de un documento. El método recibe un byte array del documento firmado y un String con el código de la firma. Comprueba si el documento custodiado es el mismo y si la firma es válida. Finalmente devuelve un objeto de tipo **FirmaInfoViafirma** incluido en el API con toda la información asociada a la firma.

```
//Enviamos el documento original y el código de firma
FirmaInfoViafirma
info=viafirmaClient.checkDocumentSigned(datosFirmados,codFirma);
```

Se puede encontrar una muestra funcional en la aplicación de ejemplo en [AplicacionDeEjemplo]/examples/utils/verificarDocumentoFirmado.jsp]

4.4.2.2. Verificación entre documento original y firmado

Para comprobar si el documento firmado corresponde con el original y no ha sido modificado utilizamos el método **checkOrignalDocumentSigned**, que recibe el byte array del documento original y el código de firma.

Devuelve un objeto de tipo **FirmaInfoViafirma** con información relativa a la firma, utilizando en esta ocasión el atributo **valid** para indicar si el documento original y el firmado son el mismo.

```
// Enviamos documento original y el identificador de firma
FirmaInfoViafirma
info=viafirmaClient.checkOrignalDocumentSigned(datosOriginalAComprobar,idFirma);
```

Se puede encontrar una muestra funcional en la aplicación de ejemplo en [AplicacionDeEjemplo]/examples/utils/validarFirma.jsp]

4.4.2.3. Comprobación de validez de firma en un documento

También tenemos la opción de comprobar si una firma es válida pasándole al método **checkSignedDocumentValidity** el byte array del documento. Éste devuelve un boolean.

```
// Enviamos el documento firmado para comprobar la firma
boolean valido = viafirmaClient.checkSignedDocumentValidity(datosFirmados);
```

Se puede encontrar una muestra funcional en la aplicación de ejemplo en [AplicacionDeEjemplo]/examples/utils/validarFirma.jsp]

4.4.2.4. Validez de un HASH a partir del documento firmado

Métod **checkSignedHashDocumentValidity** y devuelve un **FirmaInfoViafirma**

```
FirmaInfoViafirma info =
ViafirmaClientFactory.getInstance().checkSignedHashDocumentValidity(signedDoc,
hashOriginal, TypeFormatSign.XADES_EPES_ENVELOPED);
```

Disponible a partir del cliente 2.9.68.

4.4.2.5. Validez de un HASH a partir de un id de firma

Método **checkSignedHashDocumentValidityById** y devuelve un **FirmaInfoViafirma**

```
FirmaInfoViafirma info =
ViafirmaClientFactory.getInstance().checkSignedHashDocumentValidityById(idFirma,
hashOriginal, TypeFormatSign.XADES_EPES_ENVELOPED);
```

Disponible a partir del cliente 2.9.68.

4.4.3. Recuperación de un documento custodiado

Para recuperar un documento que ya ha sido firmado sólo debemos invocar al método **getDocumentoCustodiado** pasándole como argumento un String con el identificador de la firma que queremos recuperar para que nos devuelva el documento en un byte array.

```
// Recuperamos el documento firmado del servidor.
byte [] datos = viafirmaClient.getDocumentoCustodiado(idFirma);
```

Se puede encontrar una muestra funcional en la aplicación de ejemplo en [AplicacionDeEjemplo]/examples/utils/recuperarDocumento.jsp]

4.4.4. Recuperación del original de un documento firmado

Con **getOriginalDocument** recuperamos el documento original de una firma. El método recibe un String con el identificador de la firma del documento original en que estamos interesados, este devuelve un byte array con dicho documento.

```
// Recuperamos el documento firmado del servidor.
byte [] datos = viafirmaClient.getDocumentoCustodiado(idFirma);
```

Se puede encontrar una muestra funcional en la aplicación de ejemplo en [AplicacionDeEjemplo]/examples/utils/recuperarDocumentoOriginal.jsp]

4.4.5. Recuperación del listado de firmas de un lote

Cuando realizamos un lote y lo firmamos, los documentos firmados se consideran una unidad y no pueden ser accedidos independientemente, pero sí podemos recuperar los originales de cada uno. Para esto necesitamos el método **getOriginalDocumentsIds**, que a partir del String de la firma del lote devuelve un List<String> con los identificadores dentro del lote de los originales.

```
// Recuperamos el listado de identificadores de documento.
List<String> listIdentifiersDocuments = null;
listIdentifiersDocuments=viafirmaClient.getOriginalDocumentsIds(idFinalFirmaLote);
```

Después recorreremos el listado e iremos llamando al método **getOriginalDocument** para cada uno de los ids, recibiendo los originales.

```
// Comenzamos la recuperación de los documentos originales.
for (String identifierDocument : listIdentifiersDocuments) {
        Documento docu = viafirmaClient.getOriginalDocument(
        identifierDocument);
        }
```

Se puede encontrar una muestra funcional en la aplicación de ejemplo en [AplicacionDeEjemplo]/examples/utils/recuperarDocumentosOriginales.jsp]

4.4.6. Recupera información de una firma

Si queremos recuperar los datos de una firma concreta, contamos con el método **getSignInfo** que recibe el identificador de la firma y devuelve un objeto de tipo **FirmaInfoViafirma** con la información de firma.

```
// Recuperamos la información del la firma
FirmaInfoViafirma info=viafirmaClient.getSignInfo(idFirma);
```

Se puede encontrar una muestra funcional en la aplicación de ejemplo en [AplicacionDeEjemplo]/examples/utils/recuperarInfoFirma.jsp]

4.4.7. Validación de un Certificado

A partir de la v3.6 de viafirma platform se incorpora la validación del certificado de forma independiente. El método **checkCertificate** se incorpora en los clientes v2.9.70 y v3.0.41.

Clientes compatibles	**A partir de la v2.9.70 y v3.0.41**
Viafirma platform compatible	A partir de la v3.6.1

Método checkCertificate

Devuelve un objeto ValidationInfo, que contiene:

CertificateInfo, con las siguientes propiedades:

```
String  subject;
String  issuer;
Date  notafter;
Date  notbefore;
String signAlgName;
String signAlgOID;
String type;
String version;
String serialNumber;
KeyUsage keyUsage;
Map<String,String> propertiesOID;
List<CertificateInfo> trustedChain;
```

ValidatonResponse, con las siguientes propiedades:

```
static final long serialVersionUID = 1L;
boolean isValidated;
boolean isExpired;
String method;
OcspResponse ocspResponse;
CrlResponse crlResponse;
```

Consulta la Javadoc correspondiente a la versión 2.9.70 y 3.0.41 para más información.

ANEXO I : OTROS RECURSOS DE AYUDA

1. PERSONALIZACIÓN CAPA CLIENTE

Viafirma platform posee una capacidad de abstracción de cara a la integración por parte de terceros, permitiendo al integrador modificar la capa cliente según sus necesidades.

Esta operación será realizada mediante hojas de estilo (CSS) tal y como se describe a continuación.

1.1. Hoja de Estilos (CSS)

Copiar "viafirmaStyle.css" en el directorio raíz de la aplicación que vaya a integrar con viafirma-platform. La página en la que se cargará el applet de viafirma-platform mostrará el aspecto que haya sido definido en esta CSS. A continuación un ejemplo de hoja de estilo:

Ejemplo de CSS

```
body {
  background: #fff; margin: 0; padding: 0;
} #global {
  background: #fff; padding-top: 0; width: 520px;
} #cabecera {
  background: none; height: 87px;
} #cabecera h1 a {
  background: url(./images/layout/cabecera2.png) no-repeat 100% 0; height: 87px; width:
475px;
} #contenido {
  background: none;
}
#cuerpo { background: none;
} #ayuda {
  background: none; padding-left: 8px;
  padding-left: 120px;
} #ayuda .ayuda h2 {
  left: 15px;
} * html #ayuda .ayuda h2 {
} #pie {
  background: none;
}
```

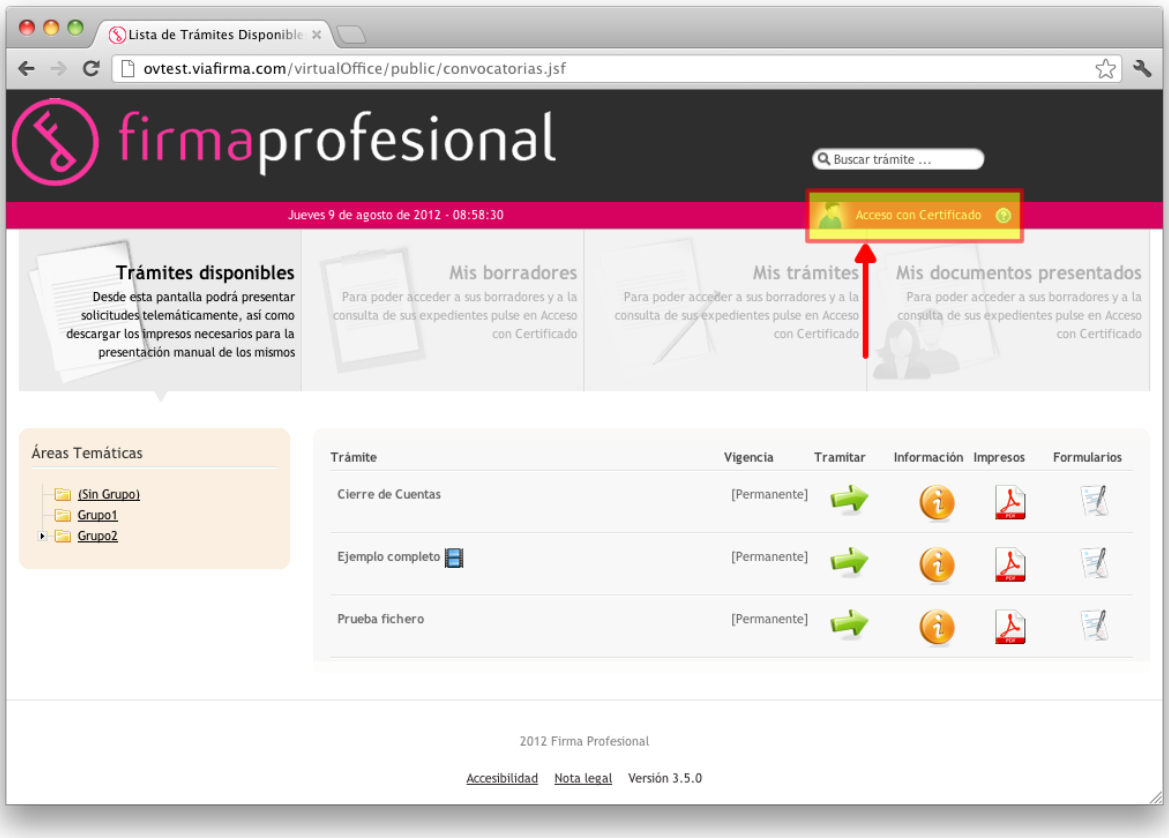
1.2. Ejemplo de integraciones

A continuación se muestran algunos ejemplos de integración con viafirma-platform.

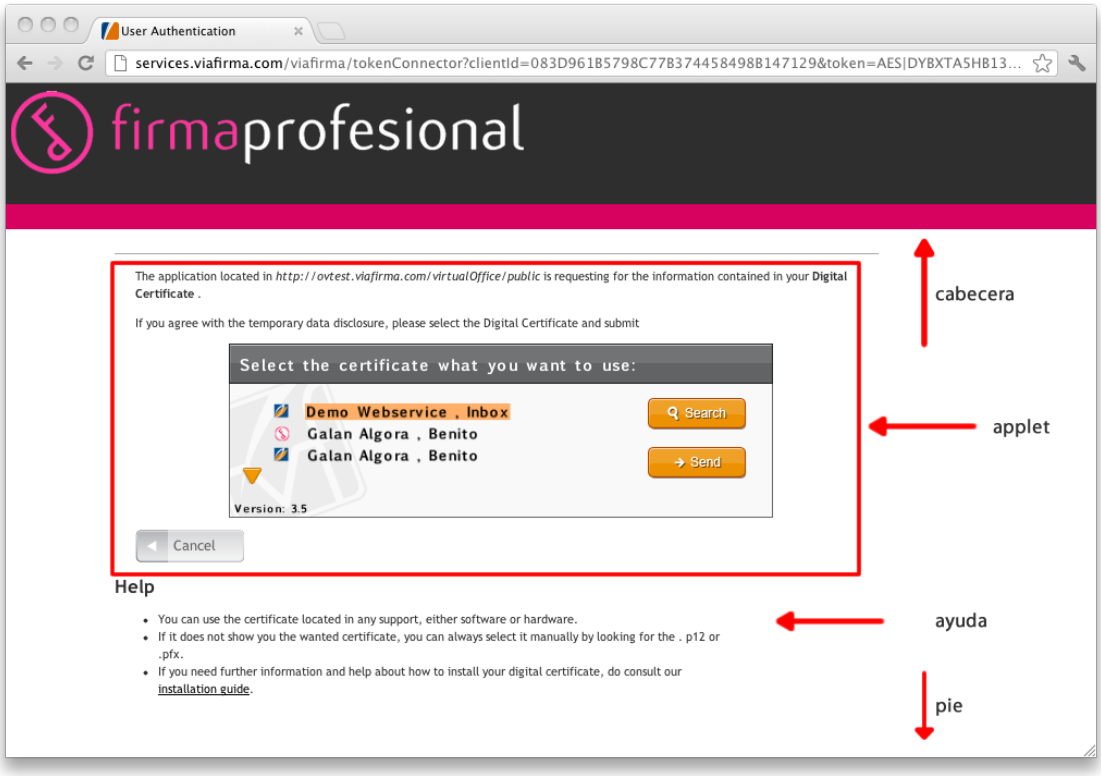
1.2.1. Virtual-Office

http://ovtest.viafirma.com/virtualOffice/public/convocatorias.jsf

En este ejemplo se realiza una integración de terceros, virtualOffice, alojada en un servidor, e integra con viafirma-platform alojada en otro servidor, en este caso, en nuestro servicio público de services.viafirma.com/viafirma/



Tras invocar la autenticación o firma con viafirma-platform, se hace push del applet desde servidor, y la página que lo renderiza se carga con las CSS contenidas en el fichero viafirmaStyles.css de la aplicación que la invocó.



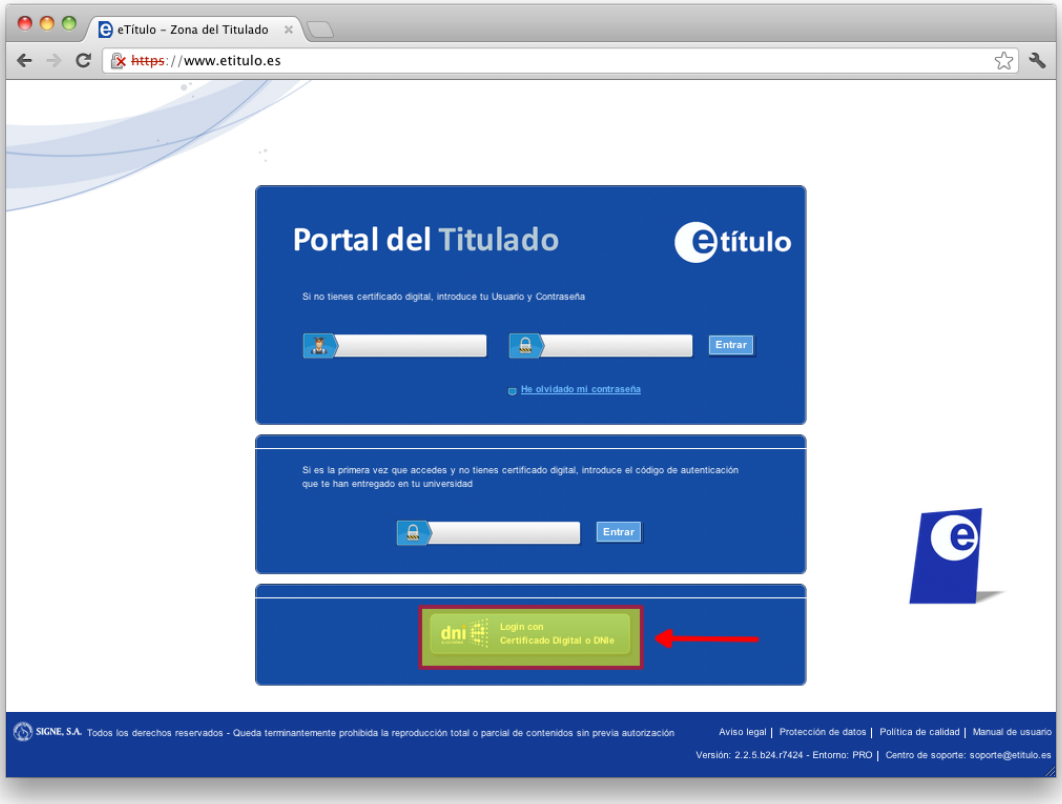
Como se puede observar en la imagen, la cabecera mantiene el mismo aspecto que la página principal de la aplicación desde la que se invocó a viafirma-platform.

Otros bloques configurables en la hoja de estilo son las secciones de ayuda y el pie.

1.2.2. E-título: portal del titulado

En este caso la integración se realiza in-house. Tanto la aplicación de terceros, etitulo, como viafirma-platform, están instaladas en el mismo servidor del cliente.

https://www.etitulo.es/



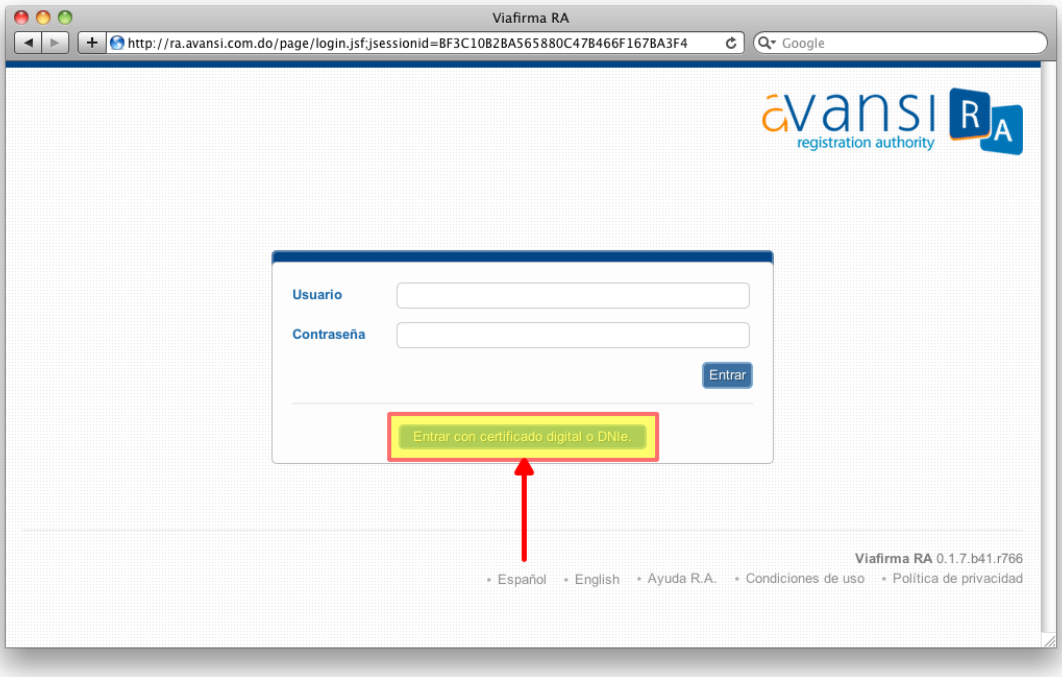
En este caso, la sección de ayuda la dejan sin contenidos.

1.2.3. RA-Avansi: integración pop-up

Existen otras integraciones en las que los integradores prefieren hacer uso de javascript que embeba el applet en un pop-up, dejando a la aplicación de terceros en un segundo plano.

El efecto es totalmente distinto a los ejemplos anteriores, tal y como se puede apreciar en los siguentes ejemplos.

https://ra.avansi.com.do/



La siguiente captura muestra un mecanismo distinto, en este caso, un pop-up en primer plano donde sólo se muestra el applet de viafirma-platform, quedando en segundo plano, deshabilitada, la aplicación de terceros que la invocó.