

@Firma :: Integración y migración de aplicaciones con la Plataforma @firma 6

*Dirección General de Política Digital
Consejería de Hacienda y Administración Pública*

Sevilla, 20 de diciembre de 2016

Integración de aplicaciones con @firma 6

- 
1. Novedades de @firma v6
 2. Servicios web de @firma v6
 3. Entornos de prueba y producción
 4. Kit de Integración de @firma
 5. Escenarios de migración:
 - Autenticación (fachada de tickets)
 - Firma de usuario
 - Firma de servidor
 6. Proceso de migración

Novedades de @firma v6

Tecnológicas

- Servidor de aplicaciones JBOSS AS 7.1.1
- Basado en Java 6
- Compatible con bases de datos Oracle 11g
- Rediseño de la capa de persistencia (Hibernate 4)
- Nueva caché de nivel 2 (Infinispan)
- Configuración basada en base de datos, con base de datos de emergencia
- Auditoría basada en Business Intelligence (Suite BI Pentaho)
- Actualización de Axis 1.3 a 1.4 para solventar problemas de memoria.
- Permite configurar la Base de Datos de @firma v6 y otra instancia de la base de datos de custodia de @firma v5 para el acceso a las firmas y documentos custodiados por esta versión de la plataforma.

Novedades de @firma v6

Módulo de firma y validación

- Validación de certificados mediante TLS europeas conforme al estándar ETSI TS-102231
- Nuevo sistema de cacheo de CRLs.
- Compatibilidad con la política de firma AGE 1.9
- Se adaptan las firmas generadas a las especificaciones RFC 3852
- Se permite actualizar firmas PAdES básicas al formato longevo PadES-LTV, así como su validación.
- Generación y validación de firmas XAdES 1.4.1
- Generación y validación de firmas CAdES 1.7.4
- Mejoras y corrección de errores en los procesos de actualizaciones a formatos longevos de XAdES y CAdES.
- Corrección de errores al actualizar firmas CAdES básicas a otros perfiles cuando la firma incorpora varios firmantes.

Novedades de @firma v6

Módulo de auditoría y de alarmas

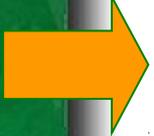
- Nuevo sistema de auditoría basado en Pentaho BI
- Nuevos informes y estadísticas más configurables.
- Mejoras en la gestión de alarmas
- Eliminación, modificación y adición de nuevas alarmas.

Novedades de @firma v6

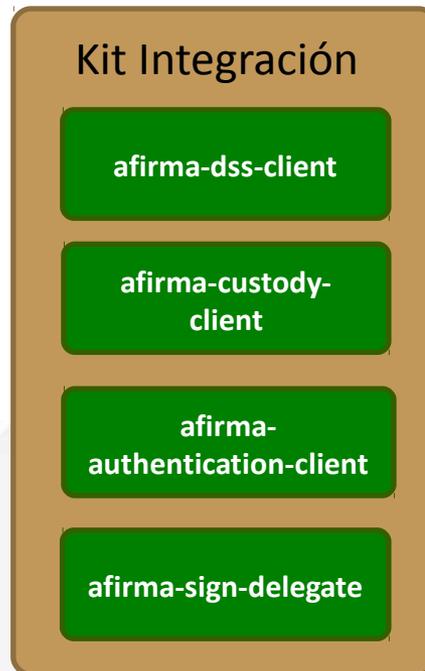
Módulo de administración

- Nuevo módulo de administración integrado con la plataforma, sin uso del Plugin de java para su ejecución.
- Se permite indicar el acceso de cada aplicación a cada servicio web de @firma.
- Mejoras en los procesos de mapeo
- Nuevo discriminador “pseudocódigo” para realizar mapeos más complejos.
- Nuevo tipo de mapeo lógico para realiar mapeos de atributos booleanos.
- Posibilidad de prueba “en caliente” de los métodos de validación de certificado, indicando el certificado a usar en la prueba.
- Inclusión de las ARLs para validación de los certificados de las CAs.

Integración de aplicaciones con @firma 6

- 
1. Novedades de @firma v6
 2. Servicios web de @firma v6
 3. Entornos de prueba y producción
 4. Kit de Integración de @firma
 5. Escenarios de migración:
 - Autenticación (fachada de tickets)
 - Firma de usuario
 - Firma de servidor
 6. Proceso de migración

Componentes de @firma



- Componentes MINHAP
- Componentes JA
- Componentes modificados

Servicios en @firma 6

ActualizarReferencia	ObtenerBloqueFirmas	ValidarCertificado
AdminDelegada	ObtenerContenidoDocumento	ValidarFirma
AlmacenarDocumento	ObtenerContenidoDocumentoOld	ValidarFirmaBloquesCompleto
EliminarContenidoDocumento	ObtenerFirmaTransaccion	ValidarFirmaBloquesDocumento
FirmaServidor	ObtenerIdDocumento	
FirmaServidorCoSign	ObtenerIdDocumentosBloqueFirmas	
FirmaServidorCounterSign	ObtenerIdDocumentosBloqueFirmasBackwards	DSSAfirmaArchiveRetrieval
FirmaUsuario2FasesF2	ObtenerInfoCertificado	DSSAfirmaArchiveSubmit
FirmaUsuario3FasesF1	ObtenerInfoCompletaBloqueFirmas	DSSAfirmaSign
FirmaUsuario3FasesF1CoSign	ObtenerInformacionBloqueFirmas	DSSAfirmaVerify
FirmaUsuario3FasesF1CounterSign	ObtenerInformacionBloqueFirmasBackwards	DSSAfirmaVerifyCertificate
FirmaUsuario3FasesF3	ObtenerTransacciones	DSSAsyncRequestStatus
FirmaUsuarioBloquesF1	ObtenerTransaccionesPorFecha	DSSBatchVerifyCertificate
FirmaUsuarioBloquesF3	ObtenerTransaccionesReferencia	DSSBatchVerifySignature



Servicios actuales



Servicios obsoletos



Servicios inexistentes

OASIS-DSS (Servicios DSS)

¿Qué es OASIS-DSS?

Digital Signature Services, define la interfaz para peticiones de servicios web que producen y/o verifican firmas digitales sobre unos datos dados.

Se basa en un par de mensajes XML petición/respuesta. A través de estos servicios un cliente puede enviar un mensaje solicitando la firma del servidor y recibiendo un XML que incluye la firma de los datos solicitados, o puede enviar una firma junto a los datos firmados y solicitar que se verifique, recibiendo una respuesta sobre si la firma es válida y corresponde con los datos enviados.

OASIS-DSS (Servicios DSS)

Servicios DSS (perfil Afirma)

Servicios DSS del perfil Afirma

- **DSSAfirmaSign**. Firma y multifirma de servidor.
- **DSSAfirmaVerify**. Verificación de firma y obtención de información sobre la misma. Además permite la realización de un upgrade o actualización sobre la firma a un formato más avanzado (por ejemplo la inclusión de un sello de tiempo).
- **DSSAfirmaArchiveSubmit**. Servicio de registro de firmas (hasta @firma 5.5).
- **DSSAfirmaArchiveRetrieval**. Servicio de obtención de firmas registradas.
- **DSSAfirmaVerifyCertificate**. Servicio de validación de certificado (a partir de @firma 5.5).
- **DSSBatchVerifyCertificate**. Servicio de validación masiva de certificados (a partir de @firma 5.5).
- **DSSBatchVerifySignature**. Servicio de validación masiva de firmas (a partir de @firma 5.5).
- **DSSAsyncRequestStatus**. Servicio de consulta del estado de peticiones asíncronas (a partir de @firma 5.5).

Integración de aplicaciones con @firma 6

- 
1. Novedades de @firma v6
 2. Servicios web de @firma v6
 3. Entornos de prueba y producción
 4. Kit de Integración de @firma
 5. Escenarios de migración:
 - Autenticación (fachada de tickets)
 - Firma de usuario
 - Firma de servidor
 6. Proceso de migración

Entornos de Prueba y Producción

Entorno de pruebas y producción

Núcleo de la Plataforma

Pruebas: <https://afirma6pru.juntadeandalucia.es>

Producción: <https://ws234.juntadeandalucia.es>

Fachada de tickets

Pruebas: <https://ws159.juntadeandalucia.es>

Producción: <https://ws235.juntadeandalucia.es>

La plataforma de @firma v6 es centralizada, con una única implantación para la Administración de la Junta de Andalucía, alojada en el Centro de Proceso de Datos de la Consejería de Hacienda y Administración Pública.

Se trata de una implantación nueva, que coexiste con la actual implantación de @firma 5.5 corporativa.

Integración de aplicaciones con @firma 6

1. Novedades de @firma v6
2. Servicios web de @firma v6
3. Entornos de prueba y producción
4. Kit de Integración de @firma
5. Escenarios de migración:
 - Autenticación (fachada de tickets)
 - Firma de usuario
 - Firma de servidor
6. Proceso de migración

Introducción

¿Qué es el kit de desarrollo de @firma?

- El kit de desarrollo de @firma es un **conjunto de librerías java** que permite a los integradores interactuar fácilmente con los servicios DSS, de autenticación y custodia de @firma desde la version 5.3.1.
- Estas librerías sustituyen a la librería utilizada hasta ahora: `afirma5ServiceInvoker.jar`.
- El kit de desarrollo está compuesto por cuatro módulos:
 - Cliente DSS (`afirma-dss-client`)
 - Cliente de autenticación (`afirma-authentication-client`)
 - Cliente de custodia (`afirma-custody-client`)
 - Cliente de firma delegada (`afirma-signer-delegate`)
- Cada uno de los módulo constituye una librería java y pueden utilizarse de manera independiente.

Introducción

Requisitos mínimos

El kit de desarrollo de @firma requiere lo siguiente para su ejecución:

- Tener instalada la **JDK 1.5. o superior.**
- **Incluir las dependencias** (ficheros jar) que se facilitan con el paquete entregable en el classpath de la aplicación cliente.
- **Visibilidad de los servicios** de @firma requeridos.
- Fichero de propiedades ***afirma.properties*** correctamente configurado e incluido en el classpath (las propiedades aquí definidas pueden ser establecidas en caliente).

Introducción

Limitaciones

El kit de desarrollo de @firma presenta las siguientes limitaciones:

- Cliente DSS:

- @firma 5.3.1 no implementa el servicio DSSVerifyCertificate (para este caso se hace uso del servicio nativo) ni los servicios asíncronos.
- Los servicios asíncronos de @firma 5.5. presentan deficiencias en su funcionamiento.
- El servicio DSSAfirmaVerify no devuelve ordenados los certificados de los firmantes en las multifirmas.

- Cliente de custodia:

- No permite custodiar documentos, sólo se acepta la consulta de los mismos.

Cliente DSS (afirma-dss-client)

Funcionalidad y servicios

¿Qué hace?

1. Implementa la lógica de mensajería:

- Realiza la composición de la petición XML
- Procesamiento de la respuesta XML

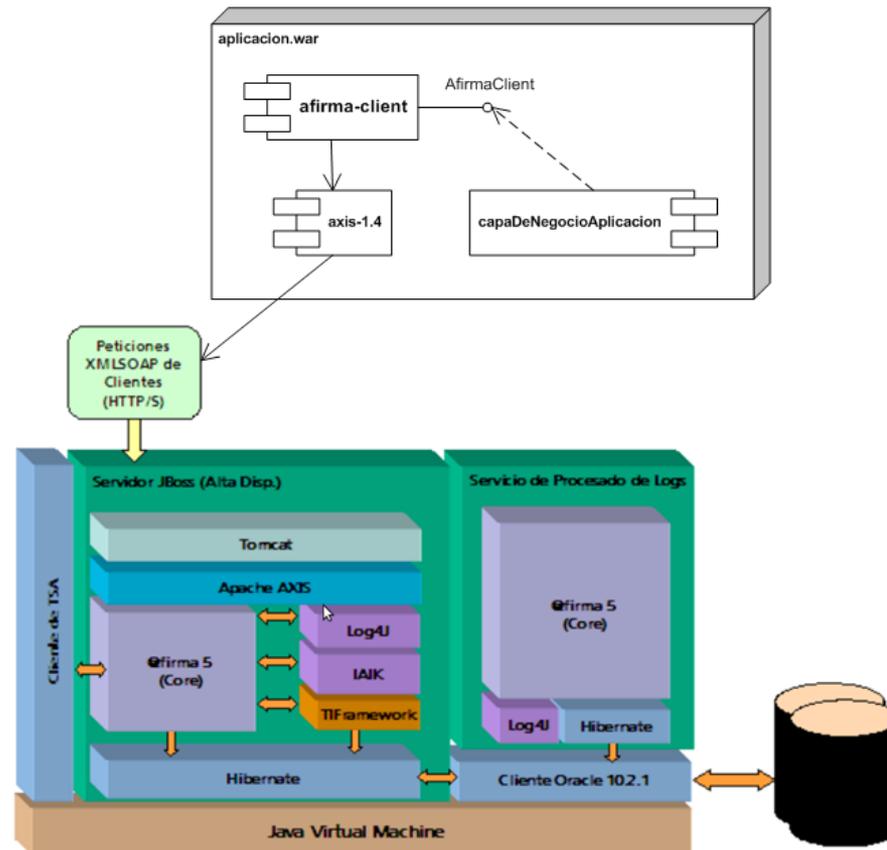
2. Implementa el envío y recepción del mensaje SOAP:

- Transparente para integrador
- Utiliza librerías de código abierto: Axis, WS Security...

Cliente DSS (afirma-dss-client)

Introducción

- El cliente DSS (afirma-dss-client) es un componente software que tiene como objetivo fundamental facilitar a los integradores la interacción con los servicios DSS disponibles en la plataforma @firma. Se trata de un componente que implementa la lógica necesaria relacionada con la mensajería (basada en los estándares DSS de OASIS) y las comunicaciones (SOAP), ofreciendo a los integradores una interfaz sencilla.



Cliente DSS (afirma-dss-client)

Servicios DSS

¿Qué servicios DSS se utilizan?

A partir de @firma 5.3.1:

- **DSSAfirmaSign**: Firma de servidor.
- **DSSAfirmaVerify**: Validación de firmas.
- **DSSAfirmaArchiveRetrieval**: Obtención de firmas custodiadas.

A partir de @firma 5.5:

- **DSSAfirmaVerifyCertificate**: Validación de certificados.
- **DSSBatchVerifyCertificate**: Validación masiva de certificados.
- **DSSBatchVerifySignature**: Validación masiva de firmas.
- **DSSAsyncRequestStatus**: Consulta de estado de peticiones asíncronas.

Cliente DSS (afirma-dss-client)

Interfaz del componente (I)

All Classes

Packages

- [es.juntadeandalucia.afirma.client](#)
- [es.juntadeandalucia.afirma.client.be](#)
- [es.juntadeandalucia.afirma.client.be](#)
- [es.juntadeandalucia.afirma.client.be](#)
- [es.juntadeandalucia.afirma.client.be](#)
- [es.juntadeandalucia.afirma.client.be](#)

All Classes

- [AbstractRequest](#)
- [AdditionalDocumentInfo](#)
- [AdditionalReportOption](#)
- [AdditionalSignatureInfo](#)
- [AfirmaArchiveProfileSchemaNS](#)
- [AfirmaClient](#)**
- [Afirmament.SignatureForm](#)
- [AfirmaClient.SignatureType](#)
- [AfirmaClient.XmlSignatureMode](#)
- [AfirmaClientBuilder](#)
- [AfirmaClientImpl](#)
- [AfirmaConfiguration](#)
- [AfirmaException](#)
- [AfirmaXSSProfileSchemaNS](#)
- [ArchiveIdentifier](#)
- [ArchiveRetrievalRequest](#)
- [ArchiveRetrievalRequestFactory](#)
- [ArchiveRetrievalResponse](#)
- [ArchiveSubmitRequest](#)
- [ArchiveSubmitRequestFactory](#)
- [ArchiveSubmitResponse](#)
- [AsyncRequestFactory](#)
- [AsyncResponse](#)
- [Base64Data](#)
- [Base64Signature](#)
- [BatchRequest](#)

Method Summary

ArchiveRetrievalResponse	dssAfirmaArchiveRetrieval (String transactionId) Servicio para la obtención de firmas electrónicas custodiadas en la plataforma.
ArchiveSubmitResponse	dssAfirmaArchiveSubmit (String sign, String certificate) Servicio para el registro o custodia de firmas electrónicas ya generadas.
ArchiveSubmitResponse	dssAfirmaArchiveSubmit (String sign, String certificate, AfirmaClient.SignatureType signatureType, AfirmaClient.XmlSignatureMode xmlSignatureMode) Servicio para el registro o custodia de firmas electrónicas ya generadas.
AsyncResponse	dssAfirmaAsyncRequestStatus (String asyncResponseId) Servicio para la consulta del estado de peticiones asíncronas.
BatchResponse	dssAfirmaBatchVerifyCertificate (String requestId, List<String> certificateList) Servicio de Validación Masiva de Certificados.
BatchResponse	dssAfirmaBatchVerifySignature (String requestId, List<VerifySignatureObject> verifySignatureObjectList, AfirmaClient.SignatureType signatureType, AfirmaClient.XmlSignatureMode xmlSignatureMode) Servicio de Validación Masiva de Firmas.
SignResponse	dssAfirmaCoSign (String documentArchiveId) Servicio de Firma Delegada que permite la realización de operaciones de cofirma de servidor en los formatos soportados en la plataforma.
SignResponse	dssAfirmaCoSign (String documentArchiveId, String aliasCertificadoFirma) Servicio de Firma Delegada que permite la realización de operaciones de cofirma de servidor en los formatos soportados en la plataforma.
SignResponse	dssAfirmaCounterSign (String documentArchiveId) Servicio de Firma Delegada que permite la realización de operaciones de contrafirma de servidor en los formatos soportados en la plataforma.
SignResponse	dssAfirmaCounterSign (String documentArchiveId, String aliasCertificadoFirma) Servicio de Firma Delegada que permite la realización de operaciones de contrafirma de servidor en los formatos soportados en la plataforma.
SignResponse	dssAfirmaSign (String document) Servicio de Firma Delegada que permite la realización de operaciones de firma de servidor en los formatos soportados en la plataforma.

Cliente DSS (afirma-dss-client)

Interfaz del componente (II)

Servicio @firma / operación	Métodos de la interfaz
DSSAfirmaArchiveRetrieval / archiveRetrieval	dssAfirmaArchiveRetrieval
DSSAfirmaSign / sign	dssAfirmaSign, dssAfirmaCoSign, dssAfirmaCounterSign
DSSAfirmaVerify / verify	dssAfirmaVerify, dssAfirmaUpgrade
DSSAfirmaVerifyCertificate/ verify	dssAfirmaVerifyCertificate
DSSBatchVerifySignature / verifySignatures	dssAfirmaBatchVerifySignature
DSSBatchVerifyCertificate / verifyCertificates	dssAfirmaBatchVerifyCertificate
DSSAsyncRequestStatus / getProcessResponse	dssAfirmaAsyncRequestStatus

Cliente DSS (afirma-dss-client)

Integración - ¿cómo se integra en mi aplicación? (I)

- **Aplicaciones desarrolladas con Maven**

El componente afirma-dss-client puede ser incluido en aplicaciones desarrolladas en maven añadiendo la siguiente dependencia al pom.xml de la aplicación:

```
<dependency>
  <groupId>es.juntadeandalucia.afirma</groupId>
  <artifactId>afirma-dss-client</artifactId>
  <version>1.x.x</version>
</dependency>
```

Del mismo modo debe incluirse la referencia al repositorio de software donde se encuentra el componente:

```
<repositories>
  <repository>
    <id>ArtifactoryRepo</id>
    <url>http://ws024.juntadeandalucia.es/maven/repository/internal</url>
  </repository>
</repositories>
```

Cliente DSS (afirma-dss-client)

Integración - ¿Cómo se integra en mi aplicación? (II)

- **Aplicaciones no Maven**

El componente afirma-dss-client está implementado en su totalidad en la librería **afirma-dss-client-x.x.x.jar**. Tan sólo hay que incluir las siguientes dependencias:

- commons-logging:commons-logging:jar:1.1
- log4j:log4j:jar:1.2.12
- logkit:logkit:jar:1.0.1
- avalon-framework:avalon-framework:jar:4.1.3
- javax.servlet:servlet-api:jar:2.3
- commons-lang:commons-lang:jar:2.1
- org.apache.ws.commons.util:ws-commons-util:jar:1.0.2
- xml-apis:xml-apis:jar:1.0.b2
- xerces:xercesImpl:jar:2.4.0
- axis:axis:jar:1.4
- org.apache.axis:axis-jaxrpc:jar:1.4
- org.apache.axis:axis-saaj:jar:1.4
- axis:axis-wsdl4j:jar:1.5.1
- commons-discovery:commons-discovery:jar:0.2
- org.apache.ws.security:wss4j:jar:1.6.7
- org.apache.santuario:xmlsec:jar:1.5.2
- org.opensaml:opensaml:jar:2.5.1-1
- org.opensaml:openws:jar:1.4.2-1
- org.opensaml:xmltooling:jar:1.3.2-1
- org.slf4j:slf4j-api:jar:1.6.1
- joda-time:joda-time:jar:1.6.2

Cliente DSS (afirma-dss-client)

Integración - Configuración

Para configurar el cliente, tan sólo hay que configurar un fichero de propiedades denominado **afirma.properties** en el CLASSPATH de la aplicación que lo utiliza.

```
#####  
# Información de conexión  
#####  
afirma.idapp = IDAPP  
afirma.host = ws028.juntadeandalucia.es  
afirma.truststore = almacenconfianza.jks  
afirma.truststorePassword = pass  
  
#####  
# Información de autenticación  
#####  
# BinarySecurityToken  
#afirma.authorization.ks.path = PATH_PKCS12  
#afirma.authorization.ks.type = PKCS12  
#afirma.authorization.ks.password = usr  
#afirma.authorization.ks.cert.alias = pas  
  
# UsernameToken  
afirma.user = user  
afirma.password = pass  
  
#####  
# Información sobre formatos de firma  
#####  
afirma.signaturetype = XAdESv1.3.2  
afirma.xmlsignaturemode = ENVELOPING  
afirma.signatureform = BES  
afirma.signaturePolicy = urn:oid:2.16.724.1.3.1.1.2.1.8  
afirma.keyname = default  
afirma.version=6
```

Cliente DSS (afirma-dss-client)

Integración - Ejemplos (I)

Ejemplo de firma delegada (firma servidor)

```
// Se obtiene el documento a firmar
String documento = Base64.encode( "<texto atributo=\"valor\">Documento
ejemplo</texto>".getBytes() );

// Se crea una instancia del componente AfirmaClient
AfirmaClient afirmaClient = AfirmaClientBuilder.getAfirmaClient();

// Se invoca la operación dssAfirmaSign con los parámetros requeridos.
SignResponse signResponse =
afirmaClient.dssAfirmaSign( documento, AfirmaClient.SignatureType.XAdES_v132,
AfirmaClient.XmlSignatureMode.ENVELOPING, AfirmaClient.SignatureForm.T );

// Se imprime la respuesta
System.out.println( signResponse );
```

Cliente DSS (afirma-dss-client)

Integración - Ejemplos (II)

Ejemplo de actualización de firma

```
// Se obtiene la firma a actualizar
String signBase64 = FileUtils.loadFileFromClasspathToString( "xades_enveloping.xsig" );

// Se crea una instancia del componente AfirmaClient
AfirmaClient afirmaClient = AfirmaClientBuilder.getAfirmaClient();

// Se invoca la operación dssAfirmaUpgrade incluyéndose como parámetro la firma en base64, el tipo
de firma, el modo de firma XML
VerifySignatureResponse verifyResponse =
afirmaClient.dssAfirmaUpgrade( signBase64, AfirmaClient.SignatureType.XAdES_v132,
AfirmaClient.XmlSignatureMode.ENVELOPING, AfirmaClient.SignatureForm.T );

// Se imprime la respuesta
System.out.println( signResponse );
```

Cliente Autenticación (afirma-authentication-client)

Introducción

- **Objetivo.** Facilitar a los integradores la interacción con los servicios de autenticación disponibles en la plataforma @firma a través de la fachada de tickets.
- **¿Qué es?** Componente software que implementa la lógica necesaria relacionada con la mensajería y las comunicaciones (SOAP), ofreciendo a los integradores una interfaz sencilla basada en servlets. El entregable lo conforman:
 - Librería core y dependencias (ficheros “jar”)
 - Manual de integrador
 - Javadoc
 - Código fuente del componente y clases de test JUnit
- **¿Qué tecnología?** Está desarrollado en java y está construido con Maven siguiendo las recomendaciones de MADEJA.

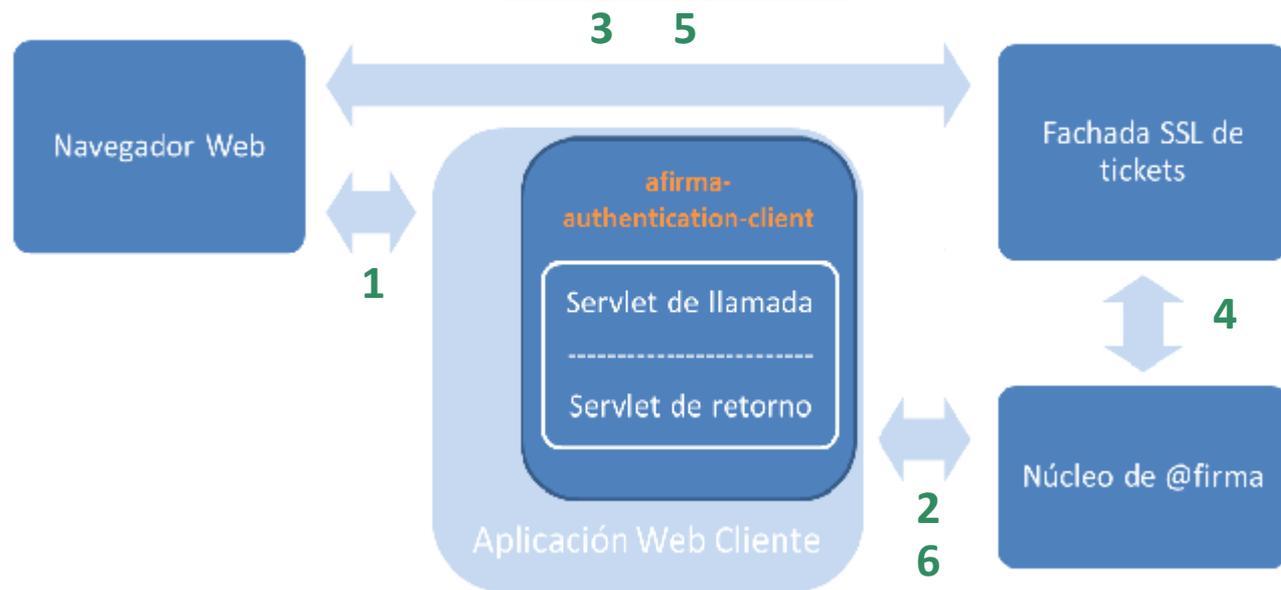
Cliente Autenticación (afirma-authentication-client)

Servicios

¿Qué servicios de autenticación utiliza?

- Fachada SSL de autenticación por tickets
- Servicios web de autenticación por tickets

Arquitectura lógica



Cliente Autenticación (afirma-authentication-client)

Integración e interfaz

- **CallAuthenticationServlet:** Servlet que se encarga de recopilar la información necesaria del navegador web (identificador de sesión), solicitar el ticket invocando el servicio **GenerarTicket** y redirigir el flujo de la aplicación a la fachada SSL de autenticación por ticket. Los parámetros que se facilitan a la fachada SSL son los siguientes:
 - identificador del ticket
 - identificador de la aplicación en @firma
 - identificador de sesión
 - URL del servlet de retorno

- **ReturnAuthenticationServlet:** Servlet que verifica la corrección de los datos facilitados por la fachada e invoca al servicio **ObtenerInfoValidacionTicket**, el cual provee los datos de validación del ticket y del certificado. Una vez finalizado el proceso, el componente de autenticación incluye los siguientes datos en sesión como resultado del proceso de validación del ticket:
 - resultado del proceso (éxito o error).
 - descripción del resultado del proceso.
 - datos del certificado validado.

La información incluida en sesión debe ser recuperada por la aplicación web cliente para finalizar el proceso de autorización de acceso a la misma.

Cliente Autenticación (afirma-authentication-client)

Integración paso a paso

1. Incluir las dependencias java en el classpath de la aplicación.
2. Establecer la configuración adecuada en el fichero `afirma.properties` e incluirlo en el classpath de la aplicación.
3. Definir los servlets de llamada y retorno en el fichero `web.xml`.
4. Invocar el servlet de llamada (**CallAuthenticationServlet**) desde algún punto de la aplicación.
5. Recuperar los datos devueltos por el servlet de retorno (**ReturnAuthenticationServlet**) que se incluyen en el atributo de sesión "`afirma_authentication_client_response`".

Cliente Autenticación (afirma-authentication-client)

Integración - Dependencias

- **Aplicaciones desarrolladas con Maven**

El componente afirma-dss-client puede ser incluido en aplicaciones desarrolladas en maven añadiendo la siguiente dependencia al pom.xml de la aplicación:

```
<dependency>
  <groupId>es.juntadeandalucia.afirma</groupId>
  <artifactId>afirma-authentication-client</artifactId>
  <version>1.x.x</version>
</dependency>
```

Del mismo modo debe incluirse la referencia al repositorio de software donde se encuentra el componente:

```
<repositories>
  <repository>
    <id>ArtifactoryRepo</id>
    <url>http://ws024.juntadeandalucia.es/maven/repository/internal</url>
  </repository>
</repositories>
```

Cliente Autenticación (afirma-authentication-client)

Integración - Dependencias

- **Aplicaciones no Maven**

El componente afirma-authentication-client está implementado en su totalidad en la librería **afirma-authentication-client-x.x.x.jar**. Tan sólo hay que incluir las siguientes dependencias:

- commons-logging:commons-logging:jar:1.1
- log4j:log4j:jar:1.2.12
- logkit:logkit:jar:1.0.1
- avalon-framework:avalon-framework:jar:4.1.3
- commons-lang:commons-lang:jar:2.1
- org.apache.ws.commons.util:ws-commons-util:jar:1.0.2
- xml-apis:xml-apis:jar:1.0.b2
- xerces:xercesImpl:jar:2.4.0
- axis:axis:jar:1.4
- org.apache.axis:axis-jaxrpc:jar:1.4
- org.apache.axis:axis-saaj:jar:1.4
- axis:axis-wsdl4j:jar:1.5.1
- commons-discovery:commons-discovery:jar:0.2
- org.apache.ws.security:wss4j:jar:1.6.7
- org.apache.santuario:xmlsec:jar:1.5.2
- org.opensaml:opensaml:jar:2.5.1-1
- org.opensaml:openws:jar:1.4.2-1
- org.opensaml:xmltooling:jar:1.3.2-1
- org.slf4j:slf4j-api:jar:1.6.1
- joda-time:joda-time:jar:1.6.2

Cliente Autenticación (afirma-authentication-client)

Integración - Configuración (I)

- La configuración se establece en el fichero `afirma.properties`, que debe incluirse en el classpath de la aplicación.
- Los parámetros de configuración son:
 - Identificador de aplicación (obligatorio)
 - Datos de autenticación:
 - `UsernameToken`
 - `BinarySecurityToken`
 - Datos del servidor (Obligatorio)
 - Almacén de certificados de confianza (SSL)
 - Host del servidor SSL de autenticación
 - URL del servlet de retorno definido en `web.xml`
 - URL de vuelta de la aplicación cliente tras finalizar el proceso

Cliente Autenticación (afirma-authentication-client)

Integración - Configuración (II)

```
#####
# Configuración del componente afirma-authentication-client
#####

#####
# Credenciales de la aplicación (obligatorio)
#
# Si se establecen las propiedades de autenticación mediante BinarySecurityToken (certificado) este tipo de autenticación prevalece respecto a la autenticación
# mediante UsernameToken (usuario / contraseña).
#####
afirma.idapp = STERIA_TEST
#####

#####
# BinarySecurityToken
#####
afirma.authorization.ks.path = cert.p12
afirma.authorization.ks.type = PKCS12
afirma.authorization.ks.password = pass
afirma.authorization.ks.cert.alias = alias

#####
# UsernameToken
#####
afirma.user = usuario
afirma.password = password
#####

#####
# Datos del servidor (obligatorio)
#
# El componente siempre realizará las peticiones al siguiente endpoint: https://<afirma_host>/afirmaws/services/<nombre_del_servicio>
#####
afirma.host = ws028.juntadeandalucia.es
#####

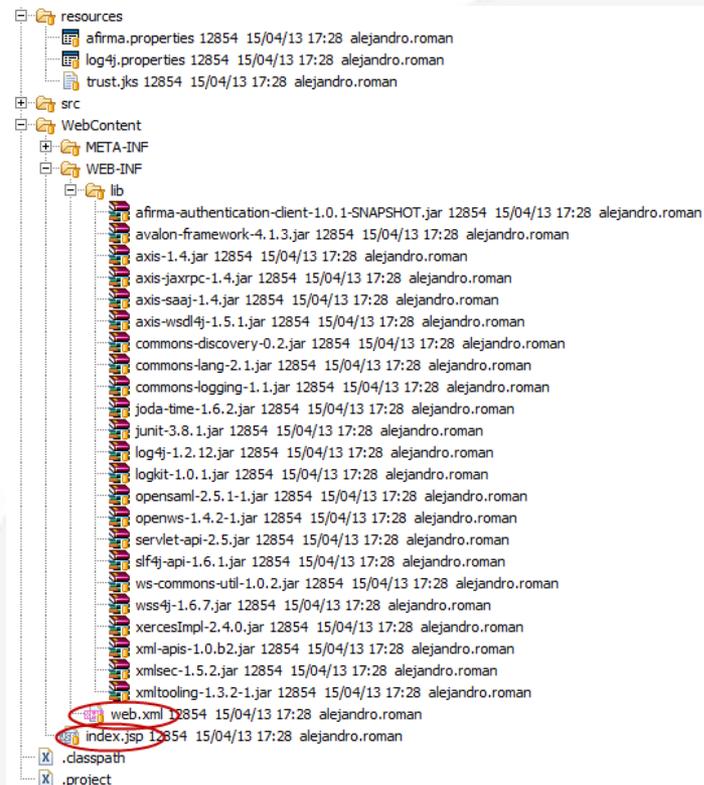
#####
# Almacén de certificados de confianza (opcional)
# El componente presupone que el almacén de certificados de confianza está incluido en el CLASSPATH de la aplicación, por lo que en 'afirma.truststore'
# debe indicarse el nombre del fichero JKS (o bien, la ruta relativa al mismo dentro del CLASSPATH).
# Por ejemplo, una aplicación web (empaquetada en un WAR) cuyo almacén está situado en el directorio WEB-INF/classes/almacenconfianza.jks, deberá
# establecer la propiedad afirma.truststore de la siguiente manera:
# afirma.truststore = almacenconfianza.jks
# También se admite referenciar al almacén de confianza mediante su ruta absoluta.
#####
afirma.truststore = trust.jks
afirma.truststorePassword = testdes
#####

#####
# Autenticación Fachada de Tickets
# Es necesario incluir los siguientes datos si se va a hacer uso de la autenticación mediante la fachada de tickets: servidor de autenticación y url de retorno
#####
# Host del servidor de la fachada SSL de autenticación por tickets
afirma.tickets.auth.host = ws116.juntadeandalucia.es
# URL del servlet de retorno definido en el fichero web.xml
afirma.tickets.url.servlet = https://[HOST_APP]/autFachadaTicketComponente/ReturnAuthenticationServlet
# URL de vuelta de la aplicación cliente tras finalizar el proceso de autenticación
afirma.tickets.url.app = https://[HOST_APP]/autFachadaTicketComponente/index.jsp
```

Cliente Autenticación (afirma-authentication-client)

Integración - Ejemplos (I)

- Creamos la estructura de una aplicación web de ejemplo para implementar la autenticación:



Cliente Autenticación (afirma-authentication-client)

Integración - Ejemplos (II)

- Definimos los servlets de llamada y retorno del componente en el fichero web.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app id="WebApp_ID" version="2.4"
  xmlns="http://java.sun.com/xml/ns/j2ee" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">
  <display-name>AuthenticationApp</display-name>
  <welcome-file-list>
    <welcome-file>index.jsp</welcome-file>
  </welcome-file-list>

  <servlet>
    <servlet-name>CallAuthenticationServlet</servlet-name>
    <servlet-class>
      es.juntadeandalucia.afirma.servlet.CallAuthenticationServlet</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>CallAuthenticationServlet</servlet-name>
    <url-pattern>/CallAuthenticationServlet</url-pattern>
  </servlet-mapping>

  <servlet>
    <servlet-name>ReturnAuthenticationServlet</servlet-name>
    <servlet-class>
      es.juntadeandalucia.afirma.servlet.ReturnAuthenticationServlet</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>ReturnAuthenticationServlet</servlet-name>
    <url-pattern>/ReturnAuthenticationServlet</url-pattern>
  </servlet-mapping>
</web-app>
```

Cliente Autenticación (afirma-authentication-client)

Integración - Ejemplos (III)

- Invocamos al servlet de llamada y recuperamos los datos resultantes de la validación del objeto "session":

```
<%@page import="es.juntadeandalucia.afirma.authentication.beans.CertificateInfo"%>
<%@page import="es.juntadeandalucia.afirma.authentication.beans.ResultAuthenticationBean"%>
<%@page import="java.util.Iterator"%>
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Ejemplo Autenticación Web.</title>
</head>
<body>

    <div align="center">

        <a href="CallAuthenticationServlet" target="_self">Autenticaci&oacute;n
            mediante certificados digitales basado en Tickets</a>

        <%
            ResultAuthenticationBean result = new ResultAuthenticationBean();
            result = (ResultAuthenticationBean) session.getAttribute( "afirma_authentication_client_response" );
        %>
        <%
            if(result != null){
                out.println( "\nTicket v&acutilde;o: " + result.isValidTicket( ) );
                out.println( "\nDescripci&oacute;n resultado autenticaci&oacute;n: " + result.getResultDescription( ) );

                if(result.getCertificateData( ) != null){
                    out.println( "\nDATOS DEL CERTIFICADO:" );

                    Iterator<CertificateInfo> iterador = result.getCertificateData( ).iterator( );
                    while(iterador.hasNext( )){
                        CertificateInfo cer = (CertificateInfo) iterador.next( );

                        out.print( cer.getFieldIdentity( ) + " - " );
                        out.print( cer.getFieldValue( ) );
                        out.println( "\n" );
                    }
                }
            }
        %>
    </div>
</body>
</html>
```

Cliente Custodia (afirma-custody-client)

Introducción

- **Objetivo.** Facilitar a los integradores la interacción con los servicios básicos de consulta del módulo de Custodia disponibles en la plataforma @firma.
- **¿Qué es?** Componente software que implementa la lógica necesaria relacionada con la mensajería y las comunicaciones (SOAP), ofreciendo a los integradores una interfaz sencilla. El entregable lo conforman:
 - Librería core y dependencias (ficheros jar)
 - Manual de integrador
 - Javadoc
 - Código fuente del componente y clases de test JUnit
- **¿Qué tecnología?** Está desarrollado en java y está construido con Maven siguiendo las recomendaciones de MADEJA.

Cliente Custodia (afirma-custody-client)

Funcionalidad y servicios (I)

¿Qué hace?

1. Implementa la lógica de mensajería:

- Realiza la composición de la petición XML
- Procesamiento de la respuesta XML

2. Implementa el envío y recepción del mensaje SOAP:

- Transparente para integrador
- Utiliza librerías de código abierto: Axis, WS Security...

Cliente Custodia (afirma-custody-client)

Funcionalidad y servicios (II)

¿Qué servicios de custodia utiliza?

- **GetDocumentContent:** Obtiene el contenido de un documento almacenado en el módulo de Custodia, a partir del identificador de transacción de la firma del documento.
- **GetDocumentContentByDocId:** Obtiene el contenido de un documento almacenado en el módulo de Custodia, a partir de su identificador de documento.
- **GetESignature:** Obtiene la firma electrónica almacenada en el módulo de Custodia a partir del identificador de transacción.

Cliente Custodia (afirma-custody-client)

Integración - Interfaz (I)

- **getDocumentContent**

- Parámetros de entrada:

- transactionId**: Identificador de la transacción de la firma del documento.

- Respuesta:

- Objeto de tipo **GetDocumentContentResponse** con los datos de la respuesta.

- Cabecera del método:

- public GetDocumentContentResponse getDocumentContent(String transactionId) throws CustodyException;**

- **getDocumentContentByDocId**

- Parámetros de entrada:

- docId**: Identificador del documento custodiado.

- Respuesta:

- Objeto de tipo **GetDocumentContentResponse** con los datos de la respuesta.

- Cabecera del método:

- public GetDocumentContentResponse getDocumentContentByDocId(String docId) throws CustodyException;**

Cliente Custodia (afirma-custody-client)

Integración - Interfaz (II)

- **getESignature**

- Parámetros de entrada:

- transactionId**: Identificador de transacción de una firma custodiada en @Firma.

- Respuesta:

- Objeto de tipo **GesESignatureResponse** con los datos de la respuesta.

- Cabecera del método:

- public GesESignatureResponse getESignature(String transactionId) throws CustodyException;**

Cliente Custodia (afirma-custody-client)

Integración - Dependencias

- **Aplicaciones desarrolladas con Maven**

El componente afirma-custody-client puede ser incluido en aplicaciones desarrolladas en maven añadiendo la siguiente dependencia al pom.xml de la aplicación:

```
<dependency>
  <groupId>es.juntadeandalucia.afirma</groupId>
  <artifactId>afirma-custody-client</artifactId>
  <version>1.x.x</version>
</dependency>
```

Del mismo modo debe incluirse la referencia al repositorio de software donde se encuentra el componente:

```
<repositories>
  <repository>
    <id>ArtifactoryRepo</id>
    <url>http://ws024.juntadeandalucia.es/maven/repository/internal</url>
  </repository>
</repositories>
```

Cliente Custodia (afirma-custody-client)

Integración - Dependencias

- **Aplicaciones no Maven**

El componente afirma-custody-client está implementado en su totalidad en la librería **afirma-custody-client-x.x.x.jar**. Tan sólo hay que incluir las siguientes dependencias:

- commons-logging:commons-logging:jar:1.1
- log4j:log4j:jar:1.2.12
- logkit:logkit:jar:1.0.1
- avalon-framework:avalon-framework:jar:4.1.3
- commons-lang:commons-lang:jar:2.1
- org.apache.ws.commons.util:ws-commons-util:jar:1.0.2
- xml-apis:xml-apis:jar:1.0.b2
- commons-codec:commons-codec:jar:1.6
- xerces:xercesImpl:jar:2.4.0
- axis:axis:jar:1.4
- org.apache.axis:axis-jaxrpc:jar:1.4
- org.apache.axis:axis-saaj:jar:1.4
- axis:axis-wsdl4j:jar:1.5.1
- commons-discovery:commons-discovery:jar:0.2
- org.apache.ws.security:wss4j:jar:1.6.7
- org.apache.santuario:xmlsec:jar:1.5.2
- org.opensaml:opensaml:jar:2.5.1-1
- org.opensaml:openws:jar:1.4.2-1
- org.opensaml:xmltooling:jar:1.3.2-1
- org.slf4j:slf4j-api:jar:1.6.1
- joda-time:joda-time:jar:1.6.2

Cliente Custodia (afirma-custody-client)

Integración - Configuración (I)

- La configuración se establece en el fichero `afirma.properties`, que debe incluirse en el classpath de la aplicación.
- Los parámetros de configuración son:
 - Identificador de aplicación (obligatorio)
 - Datos de autenticación:
 - UsernameToken
 - BinarySecurityToken
 - Datos del servidor (Obligatorio)
 - Almacén de certificados de confianza (SSL)

Cliente Custodia (afirma-custody-client)

Integración - Configuración (II)

```
#####
# Configuración del componente afirma-custody-client
#####

#####
# Credenciales de la aplicación (obligatorio)
# Si se establecen las propiedades de autenticación mediante BinarySecurityToken (certificado) este tipo de autenticación prevalece
# respecto a la autenticación mediante UsernameToken (usuario / contraseña).
#####
afirma.idapp = STERIA_TEST
#####

#####
# BinarySecurityToken
#####
afirma.authorization.ks.path = cert.p12
afirma.authorization.ks.type = PKCS12
afirma.authorization.ks.password = pass
afirma.authorization.ks.cert.alias = alias
#####

#####
# UsernameToken
#####
afirma.user = usuario
afirma.password = password
#####

#####
# Datos del servidor (obligatorio)
# El componente siempre realizará las peticiones al siguiente endpoint: https://<afirma_host>/afirmaws/services/<nombre_del_servicio>
#####
afirma.host = ws028.juntadeandalucia.es
#####

#####
# Almacén de certificados de confianza (opcional)
# El componente presupone que el almacén de certificados de confianza está incluido en el CLASSPATH de la aplicación, por lo que en 'afirma.truststore'
# debe indicarse el nombre del fichero JKS (o bien, la ruta relativa al mismo dentro del CLASSPATH).
# Por ejemplo, una aplicación web (empaquetada en un WAR) cuyo almacén está situado en el directorio WEB-INF/classes/almacenconfianza.jks,
# deberá establecer la propiedad afirma.truststore de la siguiente manera:
# afirma.truststore = almacenconfianza.jks
# También se admite referenciar al almacén de confianza mediante su ruta absoluta.
#####
afirma.truststore = trust.jks
afirma.truststorePassword = testdes
#####
```

Cliente Custodia (afirma-custody-client)

Integración - Ejemplos (I)

```
public class GetDocumentContentRequestTest extends TestCase
{
    public GetDocumentContentRequestTest( String testName )
    {
        super( testName );
    }

    public static Test suite()
    {
        return new TestSuite( GetDocumentContentRequestTest.class );
    }

    public void testGetDocumentContentRequest() throws Exception
    {
        // Creo una instancia del componente
        CustodyClient custodyClient = CustodyClientBuilder.getCustodyClient();

        // Invoco el servicio GetDocumentContent
        GetDocumentContentResponse response = custodyClient.getDocumentContent( "1365674096609848" );

        assertTrue( "true".equals( response.getStatus() ) );
    }

    public void testGetDocumentContentByDocIdRequest() throws Exception
    {
        // Documento a firmar
        String document = Base64.encode( "Hola mundo!".getBytes() );
        String documentName = "holaMundo.txt";
        String documentType = "txt";

        // Creo una instancia del componente
        CustodyClient custodyClient = CustodyClientBuilder.getCustodyClient();

        // Invoco el servicio StoreDocument
        StoreDocumentResponse response1 = custodyClient.storeDocument( document, documentName, documentType );

        System.out.println( response1 );

        assertTrue( "true".equals( response1.getStatus() ) );

        // Invoco el servicio GetDocumentContent
        GetDocumentContentResponse response2 = custodyClient.getDocumentContentByDocId( response1.getDocumentId() );

        assertTrue( "true".equals( response2.getStatus() ) );
    }
}
```

Cliente Custodia (afirma-custody-client)

Integración - Ejemplos (II)

```
public class GetESignatureRequestTest extends TestCase
{
    public GetESignatureRequestTest( String testName )
    {
        super( testName );
    }

    public static Test suite()
    {
        return new TestSuite( GetESignatureRequestTest.class );
    }

    public void testGetESignatureRequest() throws Exception
    {
        // Creo una instancia del componente
        CustodyClient custodyClient = CustodyClientBuilder.getCustodyClient();

        // Invoco el servicio GetESignature
        GetESignatureResponse response = custodyClient.getESignature( "1365674096609848" );

        System.out.println( response );

        assertTrue( "true".equals( response.getStatus() ) );
    }
}
```

Cliente Firma Delegada (afirma-signer-delegate)

Introducción

- **Objetivo.** Facilitar a las aplicaciones la generación de firmas de servidor evitando el envío de los documentos a @firma.
- **¿Qué es?** Componente software que implementa la lógica necesaria relacionada con la generación de firmas electrónicas. El entregable lo conforman:
 - Librería core, cliente y dependencias (ficheros jar)
 - Manual de integrador
 - Javadoc
 - Código fuente del componente y clases de test JUnit
- **¿Qué tecnología?** Está desarrollado en java y está construido con Maven siguiendo las recomendaciones de MADEJA.
- **¿Qué hace?** Realiza firmas electrónicas básicas y EPES: **firmas, co-firmas y contrafirmas.**

Cliente Firma Delegada (afirma-signer-delegate)

Modos de integración

El componente de firma delegada está compuesto por dos elementos software:

- **afirma-signer-delegate-server:** Librería java que puede funcionar como API de firma delegada, o bien, como servicio RMI (Remote Method Invocation).
- **afirma-signer-delegate-client:** Librería java muy ligera que consume el servicio RMI cuando el componente afirma-signer-delegate-server se ejecuta como servicio RMI.

MODOS DE INTEGRACIÓN

- **Integración con afirma-signer-delegate-server:** La aplicación hace uso del API y no necesita servicio adicional.
- **Integración con afirma-signer-delegate-client:** El elemento afirma-signer-delegate-server se ejecuta como servicio RMI aislado y recibe las peticiones del elemento cliente. **RECOMENDADO**

Cliente Firma Delegada (afirma-signer-delegate)

Integración - Interfaz

- **sign** (data, hash)
- **sign** (data, signatureType, signatureMode, hash)
- **sign** (data, signatureType, signatureMode, alias, hash)
- **sign** (data, signatureType, signatureMode, alias, signaturePolicyBean, hash)

- **coSign** (sign, data, hash)
- **coSign** (sign, data, signatureType, signatureMode, hash)
- **coSign** (sign, data, signatureType, signatureMode, alias, hash)
- **coSign** (sign, data, signatureType, signatureMode, alias, signaturePolicyBean, hash)

- **counterSign** (sign)
- **counterSign** (sign, signatureType, signatureMode)
- **counterSign** (sign, signatureType, signatureMode, alias)
- **counterSign** (sign, signatureType, signatureMode, alias, signaturePolicyBean)

Cliente Firma Delegada (afirma-signer-delegate)

Integración - Dependencias

- **Aplicaciones desarrolladas con Maven**

El componente afirma-signer-delegate puede ser incluido en aplicaciones desarrolladas en maven añadiendo la siguiente dependencia al pom.xml de la aplicación:

```
<dependency>
  <groupId>es.juntadeandalucia.afirma</groupId>
  <artifactId>afirma-signer-delegate-[server o client]</artifactId>
  <version>1.x.x</version>
</dependency>
```

Del mismo modo debe incluirse la referencia al repositorio de software donde se encuentra el componente:

```
<repositories>
  <repository>
    <id>ArtifactoryRepo</id>
    <url>http://ws024.juntadeandalucia.es/maven/repository/internal</url>
  </repository>
</repositories>
```

Cliente Firma Delegada (afirma-signer-delegate)

Integración - Dependencias

- **Aplicaciones no Maven**

El componente `afirma-signer-delegate-server` está implementado en su totalidad en la librería **`afirma-signer-delegate-server-x.x.x.jar`**. Tan sólo hay que incluir las siguientes dependencias:

- `es.juntadeandalucia.afirma:afirma-signer-delegate-common:jar:1.x.x`
- `es.gob.afirma:miniapplet-full:jar:1.1_4`
- `commons-logging:commons-logging:jar:1.1`
- `log4j:log4j:jar:1.2.12`
- `logkit:logkit:jar:1.0.1`
- `avalon-framework:avalon-framework:jar:4.1.3`
- `commons-lang:commons-lang:jar:2.1`
- `xerces:xercesImpl:jar:2.9.0`
- `xml-apis:xml-apis:jar:1.3.04`

El componente `afirma-signer-delegate-client` está implementado en su totalidad en la librería **`afirma-signer-delegate-client-x.x.x.jar`**. Tan sólo hay que incluir las siguientes dependencias:

- `es.juntadeandalucia.afirma:afirma-signer-delegate-common:jar:1.x.x`
- `commons-logging:commons-logging:jar:1.1`
- `commons-lang:commons-lang:jar:2.1`
- `log4j:log4j:jar:1.2.12`

Cliente Firma Delegada (afirma-signer-delegate)

Integración - Configuración (I)

- La configuración se establece en el fichero **signer.properties**, que debe incluirse en el classpath de la aplicación.
- Los parámetros de configuración son:
 - Almacén de certificados para firma electrónica
 - Formato de la firma (opcional)
 - Algoritmo de HASH (opcional)
 - Política de firma (opcional)

Cliente Firma Delegada (afirma-signer-delegate)

Integración - Configuración (II)

```
#####
# Configuración del componente afirma-signer-delegate
#####

#####
# Configuración del almacén de certificados
#####
signer.ks.path = anf_steria_fp.pfx
signer.ks.type = PKCS12
signer.ks.password = 12341234
signer.ks.cert.alias = anf_usuario activo
#####

#####
# Tipo de firma utilizado (opcional)
#
# Los tipos permitidos son: CADES, XAdES Enveloping, XAdES Enveloped,
# XAdES Detached, PAdES
#
# Si no se indica un tipo, se toma por defecto CADES
#####
signer.signaturetype = XAdES Enveloping
#####

#####
# Modalidad de firma (opcional)
#
# Los modos permitidos son: IMPLICIT, EXPLICIT
#
# Si no se indica una modalidad, se toma por defecto EXPLICIT
#####
signer.signaturemode = EXPLICIT

#####
# Algoritmo de HASH (opcional)
#####
signer.hashalgorithm = SHA1

#####
# Política de firma (opcional)
#####
signer.signaturepolicy.identifier = 2.16.724.1.3.1.1.2.1.8
signer.signaturepolicy.qualifier = https://ws024.juntadeandalucia.es/politicafirma/politica_firma_CHAP_v1_8.pdf
signer.signaturepolicy.identifierHash = 7SxX3erFuH31TvAw9LZ70N7p1vA=
signer.signaturepolicy.identifierHashAlgorithm = http://www.w3.org/2000/09/xmldsig#sha1
```

Cliente Firma Delegada (afirma-signer-delegate)

Integración - Ejemplos (I)

```
public void testCadesSignerDelegate() throws Exception
{
    log.info( "#testCadesSignerDelegate" );

    // Creo una instancia del componente
    SignerDelegate signerDelegate = SignerDelegateBuilder.getSignerDelegate();

    byte[] documento = "Hola mundo!".getBytes();
    byte[] sign = signerDelegate.sign( documento, SignatureType.CAdES, SignatureMode.EXPLICIT, false );

    AfirmaClient afirmaClient = AfirmaClientBuilder.getAfirmaClient();

    VerifySignatureResponse response =
        afirmaClient.dssAfirmaVerify( Base64.encode( sign ), Base64.encode( documento ), null,
                                     AfirmaClient.SignatureType.CAdES, null );
    assertTrue( response.getResult().indexOf( "ValidSignature" ) != -1 );

    log.info( "Firma: " + Base64.encode( sign ) );
}
```

Cliente Firma Delegada (afirma-signer-delegate)

Integración - Ejemplos (II)

```
public void testCadesCoSignerDelegate() throws Exception
{
    log.info( "#testCadesCoSignerDelegate" );

    // Creo una instancia del componente
    SignerDelegate signerDelegate = SignerDelegateBuilder.getSignerDelegate();

    byte[] sign = signerDelegate.sign( "Hola mundo!".getBytes(), SignatureType.CAdES, SignatureMode.IMPLICIT, false );

    byte[] cosign = signerDelegate.coSign( sign, "Hola mundo!".getBytes(), SignatureType.CAdES,
        SignatureMode.IMPLICIT, "firmaprofesional", false );

    AfirmaClient afirmaClient = AfirmaClientBuilder.getAfirmaClient();

    VerifySignatureResponse response =
        afirmaClient.dssAfirmaVerify( Base64.encode( cosign ), AfirmaClient.SignatureType.CAdES, null );
    assertTrue( response.getResult().indexOf( "ValidSignature" ) != -1 );

    log.info( "Firma: " + Base64.encode( sign ) );
}
```

Cliente Firma Delegada (afirma-signer-delegate)

Integración - Ejemplos (III)

```
public void testCadesSignerDelegate() throws Exception
{
    log.info( "#testCadesSignerDelegate" );

    // Creo una instancia del componente
    SignerDelegate signerDelegate = SignerDelegateBuilder.getSignerDelegate();

    byte[] documento = "Hola mundo!".getBytes();

    MessageDigest digest = MessageDigest.getInstance( "SHA-1" );
    byte[] hash = digest.digest( documento );

    // Firma del documento
    byte[] sign1 = signerDelegate.sign( documento, SignatureType.CAdES, SignatureMode.EXPLICIT, false );

    // Firma del HASH
    byte[] sign2 = signerDelegate.sign( hash, SignatureType.CAdES, SignatureMode.EXPLICIT, true );

    AfirmaClient afirmaClient = AfirmaClientBuilder.getAfirmaClient();

    // Valido la primera firma explícita con el documento original
    VerifySignatureResponse response =
        afirmaClient.dssAfirmaVerify( Base64.encode( sign1 ), Base64.encode( documento ), null,
            AfirmaClient.SignatureType.CAdES, null );

    assertTrue( response.getResult().indexOf( "ValidSignature" ) != -1 );

    // Valido la segunda firma explícita con el documento original
    VerifySignatureResponse response2 =
        afirmaClient.dssAfirmaVerify( Base64.encode( sign2 ), Base64.encode( documento ), null,
            AfirmaClient.SignatureType.CAdES, null );

    assertTrue( response2.getResult().indexOf( "ValidSignature" ) != -1 );

    // Valido la segunda firma explícita con el hash
    VerifySignatureResponse response3 =
        afirmaClient.dssAfirmaVerify( Base64.encode( sign2 ), null, Base64.encode( hash ),
            AfirmaClient.SignatureType.CAdES, null );

    assertTrue( response3.getResult().indexOf( "ValidSignature" ) != -1 );
}
```

Integración de aplicaciones con @firma 6

1. Novedades de @firma v6
2. Servicios web de @firma v6
3. Entornos de prueba y producción
4. Kit de Integración de @firma
5. Escenarios de migración:
 - Autenticación (fachada de tickets)
 - Firma de usuario
 - Firma de servidor
6. Proceso de migración

Escenarios de Migración

Requisitos generales

Una aplicación desarrollada en @firma v5 necesita cumplir con los siguientes requisitos mínimos para poder migrar al nuevo entorno de @firma v6, independientemente de los servicios que utilice:

- La aplicación debe tener visibilidad hacia los entornos de @firma 6.
- La autenticación con los servicios web de @firma se realiza mediante usuario y contraseña (username token) o certificado electrónico (binary token).
- La integración de la aplicación con @firma se realiza exclusivamente mediante el Kit de Integración de @firma (no se utilizan las librerías obsoletas de “afirma5invoker”).
- La aplicación es la responsable de la custodia de las firmas y documentos firmados, no utilizando la plataforma @firma para esta finalidad.
- Las operaciones de firma y validación utilizan preferentemente modalidades de firma explícitas y únicamente se envían a la plataforma @firma los hashes o resúmenes de los documentos originales.
- Si la aplicación está integrada con una instancia local de @firma, primero debe ser migrada a la Plataforma Corporativa de @firma en su versión 5.5

Escenarios de Migración

Autenticación mediante fachada de tickets

Una aplicación desarrollada en @firma v5 que utilice firmas de servidor deberá cumplir con los siguientes requisitos para su integración en @firma v6:

- La autenticación se realiza actualmente con la fachada de tickets corporativa (ws116 para el entorno de pruebas o ws117 para el entorno de producción).
- La autenticación se apoya en las librerías del kit de integración “afirma-authentication-client” (última versión 1.0.7).

Escenarios de Migración

Firma de usuario

Una aplicación desarrollada en @firma v5 que utilice firmas de usuario de cualquier tipo deberá cumplir con los siguientes requisitos para su integración en @firma v6:

- Todas las firmas electrónicas de usuario deben utilizar preferentemente el componente miniapplet en sus últimas versiones o, el cliente de firma, al menos en la versión 3.3.1.
- La integración con el núcleo de afirma se apoya únicamente en las librerías del kit de integración “afirma-dss-client” (última versión 1.1.6).
- En caso de realizarse firmas de más de un firmante (cofirmas o contrafirmas) con sello de tiempo, es necesario el uso de la última versión del componente “afirma-dss-client” 1.1.6 correctamente configurado, indicando la versión “afirma.version” .
- La aplicación es la responsable de custodiar los documentos y firmas que se realizan. No se realiza custodia documental en @firma.
- En caso de requerir acceso al módulo de custodia de afirma, éste se apoyará únicamente en las librerías del kit de integración “afirma-custody-client” (última versión 1.0.3).
- Las firmas en cualquier caso utilizarán formatos CAdES, XAdES o PAdES. Queda excluído el uso de cualquier otro formato de firma.
- Las firmas de documentos binarios, incluidos PDF, se realizarán mediante firma CAdES explícita, enviando para su actualización o firma al núcleo de la plataforma únicamente el hash o resumen del documento.

Escenarios de Migración

Firma de servidor

Una aplicación desarrollada en @firma v5 que utilice firmas de servidor de cualquier tipo deberá cumplir con los siguientes requisitos para su integración en @firma v6:

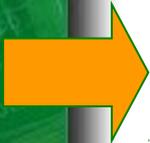
- La integración con el núcleo de afirma se apoya únicamente en las librerías del kit de integración “afirma-dss-client” (última versión 1.1.6).
- En caso de realizar firmas de servidor mediante la librería anterior, indicar que aunque actualmente es compatible este modo de firma, se ha marcado como obsoleto, de modo que no es recomendable su uso.
- **Se recomienda el uso de las librerías del kit de integración “afirma-signer-delegate” (última versión 1.0.7) para la realización de las firmas de servidor.**
- La aplicación es la responsable de custodiar los documentos y firmas que se realizan. No se realiza custodia documental en @firma.

Las firmas en cualquier caso utilizarán formatos CAdES, XAdES o PAdES. Queda excluido el uso de cualquier otro formato de firma.

- Las firmas de documentos binarios, incluidos PDF, se realizarán mediante firma CAdES explícita, enviando para su actualización o firma al núcleo de la plataforma únicamente el hash o resumen del documento.

Integración de aplicaciones con @firma 6

1. Novedades de @firma v6
2. Servicios web de @firma v6
3. Entornos de prueba y producción
4. Kit de Integración de @firma
5. Escenarios de migración:
 - Autenticación (fachada de tickets)
 - Firma de usuario
 - Firma de servidor
6. Proceso de migración



Proceso de migración

Aplicaciones actuales

Una aplicación desarrollada en @firma v5 puede ser migrada al entorno de @firma v6 si cumple los requisitos técnicos indicados anteriormente. El procedimiento de migración es el siguiente:

1. La aplicación debe estar dada de alta en el entorno corporativo de @firma v5.
2. Primero se migrará el entorno de pruebas, mediante petición en itracker, indicando la siguiente información:
 - Identificador de la aplicación.
 - Nombre de la aplicación.
 - Servicios que se utilizan: validación de firmas, validación de certificados, actualización de firmas, firma de servidor.
 - En caso de considerarlo necesario, podrá modificar el identificador de aplicación y/o método de autenticación.
3. Una vez validado el funcionamiento de la aplicación en el entorno de pruebas, se procederá a migrar el entorno de producción, mediante petición en itracker del mismo modo que se hizo para el entorno de producción.
4. Una vez validados los entornos se procederá a su baja en el entorno de @firma v5 corporativo.

Proceso de migración

Nuevas aplicaciones

Las nuevas peticiones de alta de aplicaciones que se reciban en la Dirección General de Política Digital ya serán dadas de alta en la infraestructura de la versión 6 de la Plataforma. Las nuevas aplicaciones por tanto deberán cumplir con los siguientes requisitos técnicos:

- La integración con @firma deberá realizarse obligatoriamente mediante las librerías del kit de integración, en su última versión disponible.
- La aplicación es la responsable de custodiar los documentos y firmas que se realizan. No se realiza custodia documental en @firma.
- Para los procesos de firma de servidor únicamente se aceptará la firma delegada mediante las librerías “afirma-signer-delegate” del kit de integración.
- Para los procesos de firma de usuario, se deberá utilizar el componente miniapplet en su última versión, compatible con “autofirma” (esto implica la adaptación de los procesos de firma masiva).
- En caso de sólo requerir servicios de autenticación, se usará la fachada de tickets como medio de autenticación mediante el componente “afirma-authentication-client” del kit de integración.
- Sólo se permitirán dos tipos de autenticación en la plataforma: usuario y contraseña o certificado. En este último caso, el certificado aportado podrá ser autogenerado para el entorno de pruebas y de “sello de entidad” para el entorno de producción.

Proceso de migración

Aplicaciones corporativas

Las aplicaciones corporativas cuyo desarrollo es competencia del Servicio de Coordinación de Administración Electrónica son compatibles con @firma v6:

Notific@, Compulsa, VEA, PEG, PEIT, HCV, Web de Soporte

Estas aplicaciones ya se encuentran actualmente integradas con @firma v6 en todos sus entornos.

Itracker, SCSP, eCO, Plataforma de Tramitación

Estas aplicaciones se encuentran en proceso de migración.

Portafirmas

Se liberará próximamente una nueva versión, la 3.0.3, que será compatible con @firma v6.

La próxima versión de Portafirmas, la v4 sólo será compatible con @firma v6.

Proceso de migración

Planificación temporal

El plazo límite para la migración de aplicaciones de @firma 5 a @firma 6 finalizará el **30/06/2017**. En base a esta fecha se establece el siguiente calendario de actuaciones:

A partir del **21/12/2016**, las nuevas solicitudes relativas al entorno de pruebas se darán de alta en el entorno de @firma 6. No se podrán dar de alta aplicaciones en el entorno de pruebas de @firma 5.

A partir del **31/03/2017** se producirá una baja masiva de aplicaciones en la implantación de pruebas de @firma 5. No se podrán dar de alta aplicaciones en el entorno de producción de @firma 5.

A partir del **01/01/2018**, se dejará de dar soporte a las aplicaciones en la implantación de producción de @firma 5.

Bajo demanda se podrán realizar sesiones técnicas adicionales de formación a personal TIC de las Consejerías y entidades para la incorporación de sus aplicaciones a la nueva versión de la plataforma y, en su caso, su adaptación.

**@Firma :: Integración y migración
de aplicaciones con la Plataforma
@firma 6**

*Dirección General de Política Digital
Consejería de Hacienda y Administración Pública*

