Firma por lotes predefinidos con el MiniApplet Cliente @firma

Contenidos

[Introducción 3](#_Toc433901765)

[Creación de los lotes 3](#_Toc433901766)

[Respuesta a una ejecución de un lote 11](#_Toc433901767)

[Descripción del modo transaccional de ejecución de los lotes 12](#_Toc433901768)

# Introducción

Mediante la nueva funcionalidad (aun en periodo de pruebas) es posible definir externamente un lote de firma mediante un fichero XML, ejecutándose este de forma desatendida por el cliente, con la concurrencia de los módulos servidores, y devolviéndose el resultado global de la ejecución del lote.

# Creación de los lotes

Los integradores que deseen usar esta funcionalidad deben crear (externamente, sin usar el MiniApplet) el XML de definición del lote de firma, que debe seguir este esquema:

<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified" xmlns:xs="http://www.w3.org/2001/XMLSchema">

<xs:element name="signbatch">

<xs:complexType>

<xs:sequence>

<xs:element name="singlesign" maxOccurs="unbounded" minOccurs="1">

<xs:complexType>

<xs:sequence>

<xs:element type="xs:string" name="datasource"/>

<xs:element name="format">

<xs:simpleType>

<xs:restriction base="xs:string">

<xs:enumeration value="XAdES"/>

<xs:enumeration value="CAdES"/>

<xs:enumeration value="PAdES"/>

</xs:restriction>

</xs:simpleType>

</xs:element>

<xs:element name="suboperation">

<xs:simpleType>

<xs:restriction base="xs:string">

<xs:enumeration value="sign"/>

<xs:enumeration value="cosign"/>

</xs:restriction>

</xs:simpleType>

</xs:element>

<xs:element name="extraparams">

<xs:simpleType>

<xs:restriction base="xs:base64Binary" />

</xs:simpleType>

</xs:element>

<xs:element name="signsaver">

<xs:complexType>

<xs:sequence>

<xs:element type="xs:string" name="class"/>

<xs:element name="config">

<xs:simpleType>

<xs:restriction base="xs:base64Binary" />

</xs:simpleType>

</xs:element>

</xs:sequence>

</xs:complexType>

</xs:element>

</xs:sequence>

<xs:attribute type="xs:string" name="Id" use="required"/>

</xs:complexType>

</xs:element>

</xs:sequence>

<xs:attribute type="xs:integer" name="concurrenttimeout" use="optional"/>

<xs:attribute name="stoponerror" use="optional">

<xs:simpleType>

<xs:restriction base="xs:string">

<xs:enumeration value="true"/>

<xs:enumeration value="false"/>

</xs:restriction>

</xs:simpleType>

</xs:attribute>

<xs:attribute name="algorithm" use="required">

<xs:simpleType>

<xs:restriction base="xs:string">

<xs:enumeration value="SHA1withRSA"/>

<xs:enumeration value="SHA256withRSA"/>

<xs:enumeration value="SHA384withRSA"/>

<xs:enumeration value="SHA512withRSA"/>

</xs:restriction>

</xs:simpleType>

</xs:attribute>

</xs:complexType>

</xs:element>

</xs:schema>

Un posible ejemplo de XML creado siguiendo este esquema podría ser el siguiente:

<?xml version="1.0" encoding="UTF-8" ?>

<signbatch stoponerror="false" algorithm="SHA256withRSA" concurrenttimeout="9223372036854775807">

<singlesign Id="7725374e-728d-4a33-9db9-3a4efea4cead">

<datasource>http://google.com</datasource>

<format>XAdES</format>

<suboperation>sign</suboperation>

<extraparams>

Iw0KI1N1biBBdWcgMjMgMTM6NDU6NDAgQ0VTVCAyMDE1DQpTaWduYXR1cmVJZD03NzI1Mzc0ZS03MjhkLTRhMzMtOWRiOS0zYTRlZmVhNGNlYWQNCg==

</extraparams>

<signsaver>

<class>es.gob.afirma.signers.batch.SignSaverFile</class>

<config>

Iw0KI1N1biBBdWcgMjMgMTM6NDU6NDAgQ0VTVCAyMDE1DQpGaWxlTmFtZT1DXDpcXFVzZXJzXFx0b21hc1xcQXBwRGF0YVxcTG9jYWxcXFRlbXBcXEZJUk1BMS54bWwNCg==

</config>

</signsaver>

</singlesign>

<singlesign Id="93d1531c-cd32-4c8e-8cc8-1f1cfe66f64a">

<datasource>SG9sYSBNdW5kbw==</datasource>

<format>CAdES</format>

<suboperation>sign</suboperation>

<extraparams>

Iw0KI1N1biBBdWcgMjMgMTM6NDU6NDAgQ0VTVCAyMDE1DQpTaWduYXR1cmVJZD05M2QxNTMxYy1jZDMyLTRjOGUtOGNjOC0xZjFjZmU2NmY2NGENCg==

</extraparams>

<signsaver>

<class>es.gob.afirma.signers.batch.SignSaverFile</class>

<config>

Iw0KI1N1biBBdWcgMjMgMTM6NDU6NDAgQ0VTVCAyMDE1DQpGaWxlTmFtZT1DXDpcXFVzZXJzXFx0b21hc1xcQXBwRGF0YVxcTG9jYWxcXFRlbXBcXEZJUk1BMi54bWwNCg==

</config>

</signsaver>

</singlesign>

</signbatch>

En este se distinguen los siguientes elementos:

Tamaño máximo del XML de definición de lote

El XML de definición de lote no puede exceder en tamaño de 1.024KB. Si necesitase componer trabajos mayores, estos deberán fraccionarse en varios lotes.

Cabecera de definición de lote

En el ejemplo, es la línea “<signbatch stoponerror="false" algorithm="SHA256withRSA">”. Contiene dos atributos configurables por el integrador:

1. “stoponerror”
   1. Cuando se establece a “false” se indica que el proceso debe continuar incluso si alguna de las firmas del lote no puede completarse, y cuando se establece a “true” el proceso se para en el momento en el que se produce el primer error.
2. “algorithm”
   1. Indica el algoritmo de firma a usar para todo el lote, y puede tener los siguientes valores:
      1. SHA1withRSA
         1. No recomendado por obsoleto.
      2. SHA256withRSA
      3. SHA384withRSA
      4. SHA512withRSA

Definición de cada firma dentro del lote

Dentro del elemento de definición de lote debemos incluir uno o varios elementos de tipo “singlesign”, que es obligatorio cuenten en origen con un identificados único (en el ejemplo se observa la primera (el ejemplo contiene dos firmas dentro del lote) cabecera de definición de firma “<singlesign id="7725374e-728d-4a33-9db9-3a4efea4cead">”, que indica que es una firma dentro del lote identificada por la cadena “7725374e-728d-4a33-9db9-3a4efea4cead”.

Configuración de cada firma dentro del lote

Cada una de las firmas dentro del lote puede ser configurada individualmente con los siguientes parámetros:

Origen de los datos a firmar

El origen de los datos debe indicarse dentro del elemento “datasource” del XML, por ejemplo: “<datasource>http://google.com</datasource>”

El origen de los datos a firmar puede indicarse:

1. Con una URL. En este caso el servidor (nunca el cliente) descargará directamente los datos a firmar.
   1. Se admite HTTP y HTTPS.
      1. Si se usa SSL con certificado cliente es necesario que el certificado cliente esté instalado en el almacén de certificados personales del Entorno de Ejecución de Java (JRE).
   2. En este caso, la URL debe devolver los datos finales a firmar, sin ninguna codificación intermedia (ni Base64 ni nada).
      1. Así, por ejemplo, para firmar un PDF que esté disponible en una URL debemos indicar directamente la URL que apunte a ese PDF, como:
         1. https://atos.net/content/dam/global/documents/investor-presentations/atos-q1-2015-presentation.pdf
   3. Si la URL apunta a una huella digital (para los modos de firma que lo soporten), esta debe estar igualmente en binario, como:
      1. https://miweb.com/firmas/hash001.bin
   4. En el ejemplo, puede observarse que en la primera firma se indica que el origen de los datos es la URL “http://google.com”.
2. Indicando directamente los datos a firmar codificados en Base64. En este caso el servidor descodificará el Base64 para obtener los datos a firmar.
   1. Puede usarse también para indicar directamente una huella (en los modos de firma que lo soporten), teniéndose en este caso que indicar en Base64.
   2. En el ejemplo, puede observarse que en la primera firma se indica que el origen de los datos es el Base64 “SG9sYSBNdW5kbw==”, que descodificado equivale a la cadena de texto “*Hola Mundo*”.

Existe un caso adicional, y es que cuando se indica un texto que no puede interpretarse como una URL ni como Base64, se firma este directamente sin hacer ningún tipo de transformación. En general, debe evitarse evitar este tercer modo, ya que, por una parte, puede ser difícil diferenciar entre Base64 y texto normal, y por otra, pueden encontrarse errores de interpretación por la codificación del texto (ASCII, UTF-8, ANSI, etc.).

Formato de firma

El formato de firma a utilizar debe indicarse dentro del elemento “format” del XML, por ejemplo “<format>XAdES</format>”.

Se admiten los siguientes formatos:

1. XAdES
   1. “*XAdES*”
2. CAdES
   1. “CAdES”
3. PAdES
   1. “*PAdES*”
   2. Con este formato solo se pueden firmar documentos PDF.

Algoritmo de firma

El algoritmo de firma a utilizar debe indicarse dentro del elemento “format” del XML, por ejemplo “<format>XAdES</format>”.

Se admiten los siguientes algoritmos:

1. SHA1withRSA
   1. “*SHA1withRSA*”
   2. No se recomienda por obsoleto.
2. SHA256withRSA
   1. “*SHA256withRSA*”
3. SHA384withRSA
   1. “*SHA384withRSA*”
4. SHA512withRSA
   1. “*SHA512withRSA*”

Operación de firma

La operación concreta de firma a realizar debe indicarse dentro del elemento “suboperation” del XML, por ejemplo <suboperation>sign</suboperation>”.

Se admiten las siguientes operaciones:

1. Firma
   1. “sign”
2. Cofirma
   1. “cosign”

Parámetros adicionales para la firma

Los parámetros adicionales para el formato y la operación concreta de firma (tal y como se describen en el manual del MiniApplet) deben indicarse dentro del elemento “extraparams” del XML, por ejemplo “<extraparams>bW9kZT1pbXBsaWNpdA0Kc2lnbmF0dXJlUHJvZHVjdGlvbkNpdHk9TWFkcmlk</extraparams>”.

Estos parámetros adicionales deben indicarse codificando su representación textual como Base64. Así, las siguientes propiedades (indicando cada parámetro en una línea de texto con el formato *nombre\_parámetro=valor*):

mode=implicit

signatureProductionCity=Madrid

Quedarían codificadas en Base64 como:

bW9kZT1pbXBsaWNpdA0Kc2lnbmF0dXJlUHJvZHVjdGlvbkNpdHk9TWFkcmlk

Si no se desea establecer parámetros adicionales debe dejarse el nodo vacío.

Configuración del guardado de la firma

El guardado de la firma una vez esta se completa es una tarea que realiza igualmente el servidor, utilizando para ello clases especiales de guardado que el integrador debe codificar según sus necesidades.

Las clases deben situarse en el CLASSPATH de Java de la aplicación Web en el lado servidor (normalmente dentro de un fichero JAR). Consulte la documentación de su servidor de aplicaciones para realizar esta tarea.

Una forma muy cómoda de hacerlo es editando el WAR (es un fichero ZIP con la extensión cambiada) antes de desplegarlo, y añadiendo el JAR con las nuevas clases en el directorio WEB-INF/lib. Hay que tener cuidado de no alterar el resto de contenidos ni la estructura de carpetas antes de volver a comprimir el WAR (como ZIP).

Estas clases deben implementar el interfaz Java *SignSaver*, que tiene la siguiente forma:

**package** es.gob.afirma.signers.batch;

**import** java.io.IOException;

**import** java.util.Properties;

/\*\* Interfaz para el guardado, almacenaje o envío de firmas una vez realizadas.

\* **@author** Tomás García-Merás. \*/

**public** **interface** SignSaver {

/\*\* Guarda una firma electrónica.

\* **@param** sign Definición de la firma que se hizo.

\* **@param** dataToSave Datos a guardar, resultado de la firma electrónica.

\* **@throws** IOException Si hay problemas durante el proceso. \*/

**void** saveSign(**final** SingleSign sign, **final** **byte**[] dataToSave) **throws** IOException;

/\*\* Deshace un guardado previo (para los modos transaccionales).

\* **@param** sign Identificador de la firma a deshacer. \*/

**void** rollback(**final** SingleSign sign);

/\*\* Configura cómo ha de guardarse la firma electrónica.

\* cada implementación requerirá unas propiedades distintas dentro del

\* objeto de propiedades.

\* **@param** config Propiedades de configuración. \*/

**void** init(**final** Properties config);

/\*\* Obtiene las propiedades de configuración.

\* **@return** Propiedades de configuración. \*/

Properties getConfig();

}

Como se puede observar, el objeto de guardado recibe sus parámetros de funcionamiento y configuración en un fichero de propiedades de Java en el método “init”, que es invocado siempre inmediatamente después de ser instanciado.

La forma de indicar qué clase de guardado a usar y con qué configuración es mediante el nodo “signsaver”, que contiene a su vez dos nodos:

* “class”, con el nombre cualificado de la clase a usar.
* “config”, con las propiedades de configuración codificadas en Base64.

Así, en el ejemplo tenemos el siguiente nodo:

<signsaver>

<class>es.gob.afirma.signers.batch.SignSaverFile</class>

<config>

Iw0KI1RodSBBdWcgMjAgMTI6MTM6NDEgQ0VTVCAyMDE1DQpGaWxlTmFtZT1DXDpcXFVzZXJzXFx0b21hc1xcQXBwRGF0YVxcTG9jYWxcXFRlbXBcXEZJUk1BMi54bWwNCg==

</config>

</signsaver>

Este indica que debe usarse la clase de guardado “es.gob.afirma.signers.batch.SignSaverFile” con la configuración “Iw0KI1RodSBBdWcgMjAgMTI6MTM6NDEgQ0VTVCAyMDE1DQpGaWxlTmFtZT1DXDpcXFVzZXJzXFx0b21hc1xcQXBwRGF0YVxcTG9jYWxcXFRlbXBcXEZJUk1BMi54bWwNCg==”, que si la descodificamos vemos que contiene:

FileName=C\:\\Users\\tomas\\AppData\\Local\\Temp\\FIRMA2.xml

Que es la configuración que necesita para guardar la firma (esta clase simplemente guarda la firma en el fichero que se le indique en la configuración.

Se incluyen con la implementación dos ejemplos de clases de guardado:

* Guardado a fichero:
  + Clase:
    - es.gob.afirma.signers.batch.SignSaverFile
  + Parámetros de configuración:
    - FileName
      * Indica el fichero donde debe guardarse la firma.
  + Esta clase es un simple ejemplo, y no debe usarse directamente en producción, ya que presenta problemas de seguridad (podrían indicarse que se sobrescriba un fichero de sistema).
* Envío a una dirección HTTP POST:
  + Clase: es.gob.afirma.signers.batch.SignSaverHttpPost
  + Parámetros de configuración:
    - PostUrl
      * Puede ser tanto HTTP como HTTPS
    - PostParamName
      * Indica la URL a la que hacer el HTTP POST
      * Indica el parámetro de la petición POST donde deben adjuntarse los datos.
        + Estos se adjuntarán siempre en Base64 con alfabeto URL SAFE.
  + Esta clase realiza una llamada HTTP POST a la dirección indicada, enviando como datos del POST un parámetro con el nombre indicado cuyo valor será la codificación Base64 del resultado de la firma. Es necesario entonces que el integrador provea este servicio HTTP POST para recibir y tratar la firma enviada (habitualmente un servicio Web).
    - La URL indicada debe ser accesible desde el servidor (que es quien realiza el envío).

##### **Notas sobre el ejemplo SignSaverFile**

La clase es.gob.afirma.signers.batch.SignSaverFile, es, como se ha comentado anteriormente, únicamente un ejemplo de implementación el del interfaz es.gob.afirma.signers.batch.SignSaver que puede resultar de utilidad para depuración, pero que no debe ser utilizada nunca en entornos reales.

Para evitar su exposición accidental, se distribuye con la escritura a disco deshabilitada mediante una variable final:

/\*\* El guardado real está deshabilitado por defecto, habilitar para usar esta clase

\* para depuración. No debe usarse para entornos reales, ya que no hay comprobaciones de

\* qué ficheros pueden sobrescribirse. \*/

**private** **static** **final** **boolean** ***DISABLED*** = **true**;

Si desea usarla para pruebas con escrituras reales en disco, debe habilitarla cambiando el valor de dicha variable:

**private** **static** **final** **boolean** ***DISABLED*** = **false**;

# Respuesta a una ejecución de un lote

Cuando se termina de procesar un lote de firma, el cliente recibe como respuesta un XML que describe como ha resultado el proceso.

Este XML es acorde al siguiente esquema:

<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified" xmlns:xs="http://www.w3.org/2001/XMLSchema">

<xs:element name="signs">

<xs:complexType>

<xs:sequence>

<xs:element name="signresult" maxOccurs="unbounded" minOccurs="0">

<xs:complexType>

<xs:simpleContent>

<xs:extension base="xs:string">

<xs:attribute type="xs:string" name="id" use="required"/>

<xs:attribute type="xs:string" name="result" use="required"/>

<xs:attribute type="xs:string" name="description" use="optional"/>

</xs:extension>

</xs:simpleContent>

</xs:complexType>

</xs:element>

</xs:sequence>

</xs:complexType>

</xs:element>

</xs:schema>

Un ejemplo de XML devuelto podría ser el siguiente:

<?xml version="1.0" encoding="UTF-8" ?>

<signs>

<signresult id="001-XAdES" result="DONE\_AND\_SAVED" description=""/>

<signresult id="002-CAdES" result="DONE\_AND\_SAVED" description=""/>

<signresult id="003-CAdES" result="DONE\_AND\_SAVED" description=""/>

<signresult id="004-CAdES" result="DONE\_AND\_SAVED" description=""/>

</signs>

En él distinguimos un nodo “signresult” por cada una de las firmas del lote, con su correspondiente identificador.

Este puede tener tres atributos:

1. “id”
   1. Identificador de la firma.
2. “result”
   1. Resultado del proceso.
3. “description” (opcional)
   1. Descripción del resultado del proceso.

# Descripción del modo transaccional de ejecución de los lotes

Cuando se indica que un lote debe pararse en caso de error (con el atributo “stoponerror="true"” en la cabecera del XML de definición), se activa un modo transaccional, que sigue el siguiente proceso:

1. Las firmas son generadas íntegramente en servidor, pero no se guardan hasta que no se realizan todas las del lote. SI una generación fallase se interrumpe todo el proceso y se da por perdido.
2. Una vez están generadas, se comienza el proceso de guardado de firmas (el orden del lote no es relevante, el programa puede guardarlas en un orden distinto). Si un guardado falla, se deshacen los guardados que sí se hubiesen completado adecuadamente, llamado para ello al método “rollback(**final** SingleSign sign)” de la clase de guardado (*SignSaver*) definida en el lote para cada firma.
   1. Es responsabilidad del integrador implementar adecuadamente este método “rollback(**final** SingleSign sign)” en su clase de guardado.
      1. En los ejemplos provistos de implementaciones de *SignSaver*, si se usa salvado en un archivo del sistema de ficheros, el “rollback” simplemente borra el fichero creado, pero si se usa el ejemplo de HTTP POST, el “rollback” no deshace la operación (realmente no hace nada, ya que no hay forma de deshacer una llamada HTTP).