Firma por protocolo

Índice

1 Idazki Desktop – Firma por protocolo

1.1 Descripción de la solución

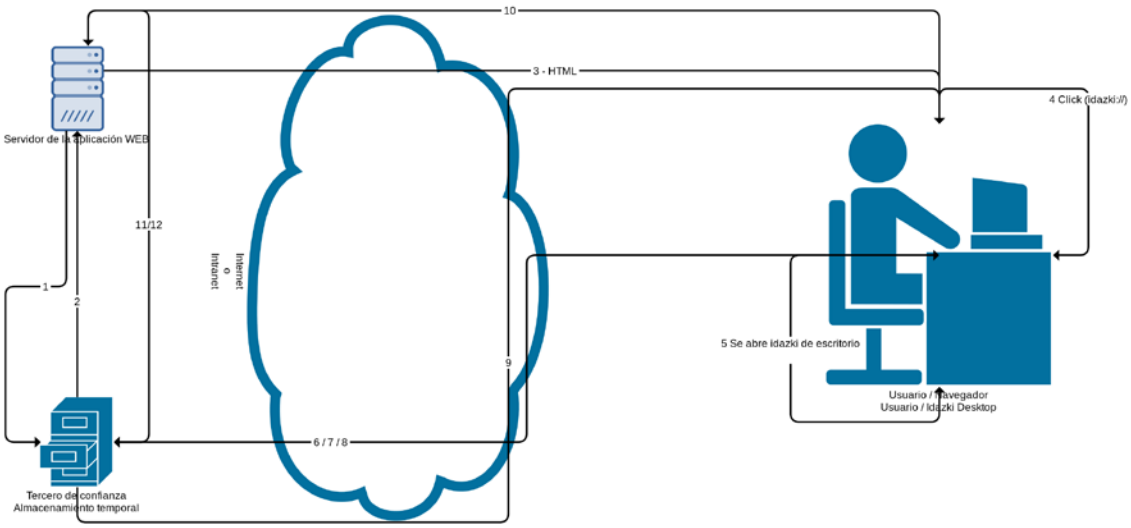
Debido al fin de soporte a los applet por parte de Chrome se ha implementado un sistema de firma basado en lo que se conoce como firma por protocolo. La firma por protocolo consiste en registrar en el sistema operativo del usuario un nuevo protocolo (en este caso idazki://). e instruir al sistema operativo para que las URI-s que tengan este protocolo sean abiertas por un programa concreto previamente instalado (en este caso Idazki Desktop). Con esta forma de hacer se persigue eliminar los applets y evitar así el uso de java dentro del navegador.

Esta solución viene a abordar el problema de comunicación entre el dispositivo local de firma (pkcs11 y pkcs12) y el navegador. Hasta ahora siempre se había abordado este problema desde el punto de vista de los applets. Esto es así porque el problema a solventar es la comunicación entre el dispositivo y el navegador. El navegador no puede acceder al dispositivo y un programa ejecutado fuera del contexto del navegador no puede acceder a los datos del navegador. Como se ha comentado, históricamente esta unión se ha realizado con applets que permiten acceder a ambos elementos y hacer de puente entre ellos.

Por tanto la idea es cambiar el applet por lo descrito anteriormente, que permite coordinar, usando esta arquitectura, el servicio de terceros (a partir de ahora denominado Tercero), como puente entre navegador y dispositivo criptográfico.

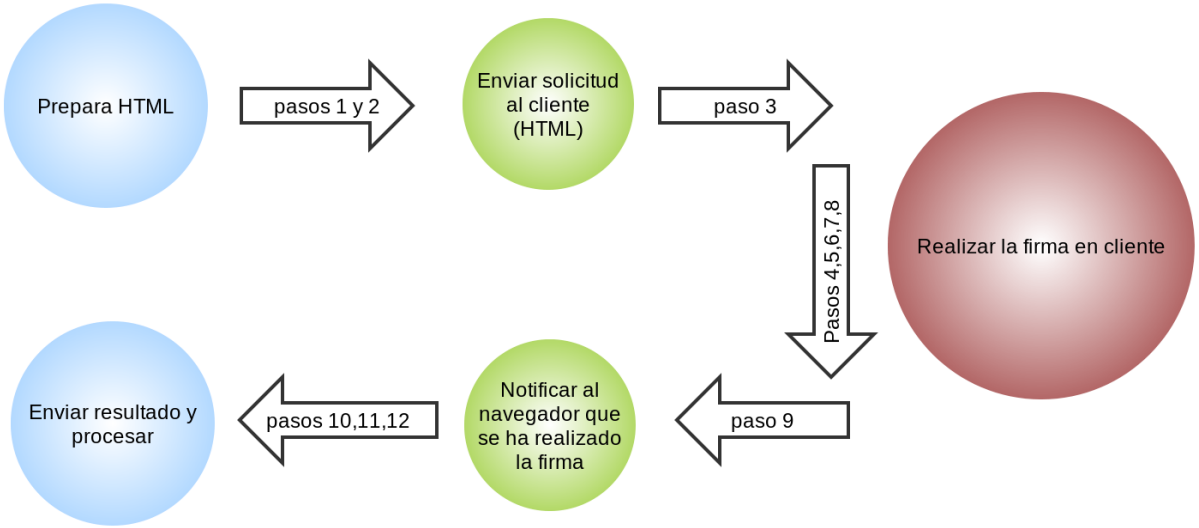
1.1.1 Arquitectura del sistema y flujo de información

En la siguiente gráfica se representa el flujo de información de una solicitud que conlleva una firma mediante el nuevo sistema.



A modo de resumen los 12 pasos representados en el gráfico pueden ser agrupados en los siguientes bloques:

- Preparar el HTML para la firma.
- Enviar la solicitud al cliente (HTML).
- El cliente realiza la firma.
- Se notifica al navegador que la firma se ha realizado.
- Se envía el resultado y se procesa.



A continuación, se describe cada uno de los pasos con mayor detalle:

Paso	Descripción	Scope
1	Se prepara la firma que se va a enviar al navegador, token de seguridad y tipo de firma.	Servidor → Tercero
2	El tercero devuelve un token que se usará para la seguridad y para identificar las características de las transacción iniciada.	Tercero → Servidor
3	El servidor devuelve el html con el javasript el token de la transacción etc...	Servidor → Navegador
4	El usuario hace click en el link por protocolo que está en la página servida en el html.	Navegador → Navegador Sistema operativo cliente
5	El sistema operativo del cliente identifica el protocolo y abre la aplicación de escritorio (fuera del contexto del navegador). Se pasa en esta apertura el token de las transacción.	Navegador → Idazki de escritorio

6-7	Idazki de escritorio obtiene la info del tipo de firma y resto de información necesaria para realizar la firma (este es el punto de unión).	Idazki de escritorio → Tercero
8	Se completa la firma y se envía al tercero.	Idazki de escritorio → Tercero
9	Se notifica el id de la firma al navegador (polling) establecido en el paso 3	Tercero → navegador o navegador a → tercero
10	Se envía la info del formulario y la referencia de la firma al servidor	Navegador → Servidor
11-12	Se procesa la información en el servidor y se recupera el documento firmado del tercero. El proceso finaliza.	Servidor → Tercero

1.1.2 Módulos de la solución

Por tanto, la solución final se compone de los siguientes módulos:

- **Librería Javascript (idazki.js):** Se trata de una librería javascript que permite integrar la firma digital en las aplicaciones Web que así lo requieran. Es la que provee la URI de firma al navegador (con el nuevo protocolo registrado).
- **Idazki Desktop:** Aplicación de escritorio que se encarga de registrar el protocolo (en el proceso de instalación), atender las URI con dicho protocolo, y realizar la firma en el equipo del usuario.
- **Tercero de confianza:** Se trata del servicio que actúa de Bridge entre la aplicación Web (librería javascript) e Idazki Desktop.

1.2 Características generales

1.2.1 Primitivas criptográficas implementadas

La solución dispone de las siguientes primitivas criptográficas:

- Firma electrónica con los siguientes formatos:
  - XMLDSig / XAdES (BES, EPES, T, C[1], XL) enveloping/enveloped/detached. Soporte multifirma en paralelo (para firmas enveloping) y contrafirma (para firmas enveloped)[2].
  - Facturae (EPES-BES)
  - CMS / CAdES (BES, EPES, T, C) attached/detached. Soporte multifirma en paralelo y contrafirma (no está soportado contrafirmas en multifirma en paralelo) en attached. Multifirma CMS dettached.
  - XMLDSig enveloping/enveloped/detached. Soporte multifirma.
  - PDF[3] (con y sin sello de tiempo)/ PAdES (BES, EPES con y sin sello de tiempo)
- Soporte para PKCS#11 y almacén criptográfico de Windows.
- Calculo de hash SHA-1 y SHA-256.
- Validación de certificados contra el OCSP de IZENPE[4].
- Sellado de tiempo contra la TSA de IZENPE.
- Búsqueda de certificados en los almacenes criptográficos.

1.2.2 Sistemas operativos / Navegadores soportados

Se consideran soportados los siguientes navegadores:

Navegador	Versión
	3.3.2
IE7	
IE8	
IE9+	
Firefox 3+	
Chrome 42+	
Safari 7+	

1 Para XAdES-C y CAdES-C solo se añadirá la respuesta OCSP de Izenpe.

2 Antes de añadir la firma a un documento se comprobará la validez criptográfica de las firmas existentes.

3 No se soportan PDFs protegidos, ni PDF con formularios XFA

4 Siempre que se proceda a firmar, antes se comprobará que el certificado no esté revocado ni suspendido.

Se consideran soportados los siguientes sistemas operativos:

Sistema operativo	Versión
	3.3.2
Windows XP	
Windows 7	
Windows 8	
Windows 10	
Ubuntu 14.04	
OSX 10.12	

1.2.3 Formatos de firma soportados

En la siguiente tabla se muestra la relación de firmas soportadas:

Tipo de firma	Versión
	3.3.2
XAdES-BES	
XAdES-EPES	
XAdES-T	
XAdES-C	
XAdES-X	
XAdES-XL	
XAdES-A	
XAdES-Enveloped	
XAdES-Enveloping	
XAdES-Detached	
Facturae	
CAdES-BES	
CAdES-EPES	
CAdES-T	
CAdES-C	
CAdES-X	
CAdES-XL	
CAdES-A	
CAdES-Atached	
CAdES-Detached	
PDF	
PDF con timestamp	
PadES-BES	
PadES-EPES	
PadES-T	

1.2.4 Dispositivos soportados

Windows

Tarjeta	Tipo	Observaciones
CAPI	CAPI	Por defecto se utiliza este sistema, de modo que se soportan todas las tarjetas que se integren con CAPI.
Dnie	pkcs11	c:\\windows\\system32\\UsrPkcs11.dll
Izenpe	pkcs11	c:\\windows\\system32\\aetpkss1.dll
Izenpe	pkcs11	c:\\windows\\system32\\bit4ipki.dll
FNMT	pkcs11	c:\\windows\\system32\\FNMT_P11.dll c:\\windows\\system32\\FNMT_P11_x64.dll

Linux

Tarjeta	Tipo	Observaciones
DNIe (driver de izenpe)	pkcs11	/usr/lib/dniepkcs11.so /usr/lib64/dniepkcs11.so
Izenpe	pkcs11	/usr/lib/libbit4ipki.so /usr/lib64/libbit4ipki.so
Opensc	pkcs11	/usr/lib/opensc-pkcs11.so o /usr/lib/x86_64-linux-gnu/opensc-pkcs11.so
FNMT	Pkcs11	/usr/lib/libpkcs11-fnmtdnie.so /usr/lib64/libpkcs11-fnmtdnie.so

Mac

Tarjeta	Tipo	Observaciones
Izenpe	pkcs11	/System/Library/Izenpe/pkcs11/libbit4ipki.dylib
Opensc	pkcs11	/Library/OpenSC/lib/opensc-pkcs11.so
FNMT	pkcs11	/Library/Libpkcs11-fnmtdnie/lib/libpkcs11-fnmtdnie.so

1.2.5 Encodings soportados

La firma por protocolo Idazki permite firmar ficheros, contenidos binarios y/o textos en claro. Al configurar estas entradas a firmar puede darse el caso que los parámetros de configuración contengan caracteres no soportados en UNICODE (como tildes o eñes). Dado que la codificación de estos caracteres especiales depende del encoding utilizado se puede corromper la firma si todos los elementos del proceso de firma no utilizan el mismo encoding.

En el caso de Idazki, el encoding establecido es el UTF-8. Para poder asegurar un correcto funcionamiento en firmas cuyos parámetros contenga caracteres especiales es necesario que todos los participantes implicados en el proceso de firma usen UTF-8 como encoding.

1.3 Librería Javascript – idazki.js

Como se ha adelantado anteriormente esta librería es la encargada de integrar el proceso de firma en las aplicaciones Web. Para el diseño de la librería se ha tratado de mantener el nombre de las funciones del API del applet Idazki, de forma que la migración del código de las integraciones del applet pueda realizarse de la forma más sencilla posible. La librería javascript consta de los siguientes métodos:

- **setOption**(key, val)
- **signSetAdESLevel**(val)
- **setCryptoStoreAuto**()
- **setCryptoStorePkcs12(path,password)**
- **addInput**(intype, incontent, outtype, outcontent)
- **sign**(signtype, successCallback)
- **getOutputContent**(idx, isbinary, callback)
- **getOutputCount**()
- **signXAdESEnvelopedSelectNode**(xmlDocumentBase64, nodeNameToSign, nodeNameToInsertSignature, c14method, xmlDocumentEncoding, successCallback)
- **signXAdESEnvelopedSelectNodeBase64**(xmlDocumentBase64, nodeNameToSign, nodeNameToInsertSignature, c14method, xmlDocumentEncoding, successCallbac)
- **hash**(successCallback)
- **setErrorCallback**(callback)
- **endTransaction**()
- **getToken**()
- **sayHello(**onSuccess,onError, self**)**
- **setSignTimeOut(**timeout**)**

Como se puede observar las funciones son muy similares a las que tenía el Applet Idazki, y su funcionamiento es también básicamente el mismo. Simplemente en este caso al tratarse de una librería javascript el flujo es asíncrono y se gestiona mediante el uso de callbacks.

1.3.1 API Javascript

A continuación se describe con mayor detalle las características de cada uno de los métodos del API.

setErrorCallback

Define un callback para gestionar los errores producidos durante la firma.

FORMATO DE LA LLAMADA

setErrorCallback(callback)

PARAMETROS

- **callback:** Función a ejecutar en caso de error, con el siguiente esquema **function(errorCode)**. Donde **errorCode** es el nemónico del último error ocurrido.

Nemónico de error	
ERR_INCORRECTPPASSWORD	PIN incorrecto. Al de tres intentos consecutivos la tarjeta se bloqueará.
ERR_PIN_LOCKED	La tarjeta se encuentra bloqueada.
ERR_BADPASSWORDORCORRUPTFILE	La contraseña es incorrecta o el archivo está corrupto
ERR_CANNOTOPENFILE_FILENAME	No se pudo abrir un archivo
ERR_CANCELEDBYUSER	El usuario canceló la operación
ERR_CANNOTUSEPKCS12_FILENAME	No se pudo abrir un PKCS#12
ERR_CANNOTUSEPKCS11_FILENAME	No se pudo usar un PKCS#11
ERR_CANNOTUSESYSTEMCRYPTOSTORE	No se pudo usar el almacén criptográfico de Windows.
ERR_INVALIDPARAMETERS_PARAMETERS	Se pasaron parámetros inválidos a una llamada.
ERR_INVALIDADESLEVEL_PARAMETERS	Se especificó mal un nivel de AdES
ERR_INVALIDSIGNATUREOPERATION_PARAMETERS	Se pasaron parámetros inválidos a una llamada de firma.
ERR_CANNOTCREATESIGNATURE	No se pudo crear la firma.

ERR_CANNOTCIPHERDOCUMENT	No se pudo cifrar un documento.
ERR_CANNOTDECIPHERDOCUMENT	No se pudo descifrar un documento.
ERR_NOCERTIFICATESFORSIGN	No se encontraron certificados para firmar.
ERR_SHOULDINSTALLJREPOLICYFILES	No están instaladas las Sun JCE.
ERR_PASSWORDSDOESNOTMATCH	La comprobación de contraseña falló.
ERR_BROWSERNOTFOUNDGOTOPAGE_URL	No se pudo abrir un navegador web.
ERR_BADPASSWORDORCORRUPTFILE	No se pudo descifrar un archivo porque la contraseña era incorrecta o el archivo estaba corrompido.
ERR_CANNOTRETRIEVETIESPUBLIC	No se pudo recuperar el código TIES
ERR_INTERNALERROR	Ocurrió un error interno.
ERR_SELECTEDCERTIFICATEISREVOKED	El certificado seleccionado está revocado.
ERR_CANNOTCREATETIMETSAMP	No se pudo crear un sello de tiempo.
ERR_CANNOTSIGNPROTECTEDPDF	No se puede firmar un PDF protegido.
ERR_CANNOTLOADCONFIG	No se pudo leer la configuración.
ERR_CANNOTFINDTIESCARD	No se pudo recuperar el código TIES
ERR_INVALIDVERIFYOPERATION_PARAMETERS	Parámetros inválidos para la operación de verificación de firma
ERR_CANNOTVERIFYDOCUMENT	No se pudo verificar la firma de un documento
ERR_NOFILESSELECTED	No se seleccionaron archivos.
ERR_INVALIDVERIFYOPERATION_PARAMETERS	Error en los parámetros de llamada de verificación de firma.
ERR_JS_CONNECTION	Error al establecer la conexión con el tercero.
ERR_WS_INTERNAL	Error interno en el tercero.
ERR_WS_BAD_REQUEST	Formato o parámetros incorrectos en la llamada REST al tercero.
ERR_APP_INTERNAL	Error interno en Idazki Desktop.
ERR_APP_CLIENTINCOMPATIBLE	Versión de Idazki Desktop incompatible con la versión del tercero / API javascript.
ERR_UPDATED	Se ha iniciado un proceso de actualización de

Idazki

EJEMPLO

```
idazki.setErrorCallback(function(errorCode) {
        document.form.texto_salida.value = errorCode;
        idazki.endTransaction();
});
```

setOption

Establece una opción genérica del comportamiento del applet.

FORMATO DE LA LLAMADA

setOption(CadenaTexto key, CadenaTexto value)

PARAMETROS

- **key**: Clave de la propiedad.
- **value**: Valor de la propiedad.

Las diferentes opciones que se permiten son:

Clave	Funcionalidad
crypto-winusepkcs11	Especifica si Windows debe probar primero usar PKCS#11 en lugar del CSP de Windows.
crypto-pkcs11libs	Especifica la lista de librerías PKCS#11, separadas por punto y coma, que se intentan cargar. La lista por defecto contempla los drivers para – DNIe – FNMT – Izenpe – OpenSC (linux) Se debe de indicar el path completo de la librería. Por ejemplo para usar el DNIe en un sistema Windows 64 bits hay que poner: *idazki.setOption("crypto-pkcs11libs","C:/Windows/SysWOW64/DNIe_P11.dll");*
timestamp-url	URL del servidor de timestamp
timestamp-policyoid	OID de política requerida del servidor de timestamp
timestamp-hashalgo	Algoritmo de hash del sello de tiempo (SHA-1 o SHA-256)
signature-hashalgo	Algoritmo de hash de la firma (SHA-1 o SHA-256)
signature-signerrole	Rol del firmante en firmas CAdES/XAdES

ades-add-policy[1]	"1" si hay que añadir la política a firmas de tipo CAdES/XAdES/PAdES.
xades-signature-policy-id-identifier	Identificador de política XAdES
xades-signature-policy-id-description	Descripción de la política XAdES
xades-signature-policy-hash	Hash de la política para firmas XAdES
xades-signature-policy-hashalgo	Algoritmo de hash de política para firmas XAdES
xades-signature-policy-qualifier-uri	URL de la política EPES para firmas XAdES
xades-node-to-sign-name	Id del nodo a firmar. Únicamente válido para las firmas enveloping. Para las firmas enveloped hay un método especifico con el mismo propósito (signXAdESEnvelopedSelectNode).
xades-node-to-sign-root	'true' para firmar el propio nodo raíz del documento en las firmas enveloping, de lo contrario se firma un nodo contenedor (ds:Object). Para utilizar esta opción es necesario que el nodo raíz del documento contenga un atributo **Id**. Si se utiliza la propiedad 'xades-node-to-sign-name' esta propiedad no se tiene en cuenta. Únicamente válido para las firmas enveloping. El valor por defecto de esta propiedad es 'false'.
cades-signature-policy-oid	OID de política para firmas CAdES
cades-signature-policy-hash	Hash de la política EPES para firmas CAdES
cades-signature-policy-hashalgo	Algoritmo de hash de política para firmas CadES
pades-signature-policy-oid	OID de política para firmas PAdES
pades-signature-policy-hash	Hash de la política EPES para firmas PAdES

1 Si esta opción está a "1" debe podrán los parámetros xades-signature* para tipos XAdES, cades-signature* para tipos CAdES y pades-signature* para tipos PAdES. En los tres tipos de firmas, Izenpe asignará unos valores por defecto.

pades-signature-policy-hashalgo	Algoritmo de hash de política para firmas PAdES
pades-signature-policy-qualifier-uri	URL de la política EPES para firmas PAdES
pdf-signature-llx	Posición X_0 de visualización de firma en PDF
pdf-signature-lly	Posición Y_0 de visualización de firma en PDF
pdf-signature-urx	Posición X_1 de visualización de firma en PDF
pdf-signature-ury	Posición Y_1 de visualización de firma en PDF
pdf-signature-page	Página de la firma PDF, ... -2=Antepenúltima -1=Penúltima, 0=Última, 1=Primera, 2= Segunda...
pdf-signature-reason	Razón de la firma PDF
pdf-signature-location	Lugar de la firma
pdf-signature-contact	Contacto de la firma
pdf-signature-image	Si está definido, especifica la imagen gráfica de fondo para la firma en formatos PDF. Se puede especificar la ubicación de un archivo (p.e. c:/temp/firma.gif) o bien una URL (http:// o https://)
pdf-signature-image-bytes	Si está definido, especifica la imagen gráfica de fondo para la firma en formatos PDF. Se espera que en este parámetro se proporcionen los bytes de la imagen en Base64. *NOTA: Si la opción 'pdf-signature-image' está definida tiene preferencia sobre esta propiedad.*
pdf-signature-image-foreground	Si está definido, especifica la imagen gráfica para la firma en formatos PDF. Se puede especificar la ubicación de un archivo (p.e. c:/temp/firma.gif) o bien una URL (http:// o https://)
pdf-signature-image-foreground-bytes	Si está definido, especifica la imagen gráfica para la firma en formatos PDF. Se espera que en este parámetro se proporcionen los bytes de la imagen en Base64. *NOTA: Si la opción 'pdf-signature-image-foreground' está definida tiene preferencia sobre esta propiedad.*

pdf-signature-text	Si está definido, especifica el texto que se muestra en la firma en formatos PDF. Esta propiedad acepta varias líneas usando un token como separador de líneas. El token es '**~#~**' (sin las comillas). Dentro de esta clave, se pueden definir las siguientes claves que se traducirán por su valor correspondiente en el momento de realizar la firma. Estas variables son: ${pdf-signature-subject}: Muestra el Subject del certificado del firmante. ${pdf-signature-name}: Muestra el CN del certificado del firmante. ${pdf-signature-issuer}: Muestra el Issuer del certificado del firmante. ${pdf-signature-serial}: Muestra el número de serie del certificado del firmante. ${pdf-signature-date}: Muestra la fecha y hora a la que fue firmado el documento. ${pdf-signature-contact}: Muestra la información de contacto si se ha definido en el parámetro pdf-signature-contact arriba indicado. ${pdf-signature-location}: Muestra la información de localización si se ha definido en el parámetro pdf-signature-location arriba indicado. ${pdf-signature-reason}: Muestra la información de razón de la firma si se ha definido en el parámetro pdf-signature-reason arriba indicado.
pdf-signature-text-font-name	Nombre del tipo de fuente que se va a usar. Afecta a todo el texto fijado en pdf-signature-text. Los valores posibles son: Courier, Helvetica, Times-Roman, Symbol, ZapfDingbats
pdf-signature-text-font-color	Color en formato RGB de la fuente del texto. Los valores se deberán pasar separados por comas. Por ejemplo

	255,255,255
pdf-signature-text-font-size	Entero que fija el tamaño de la fuente.
pdf-signature-text-font-style	Estilo de letra ("normal", "bold", "italic") con el que se escriben los detalles de la firma que se muestran como texto en su apariencia.
pdf-signature-text-date-format	Formato de la fecha que se va a incluir en la firma. Por defecto se usa yyyy.MM.dd HH:mm:ss XXX
estimated-signature-size	Tamaño reservado para el tamaño de la firma PDF. Por defecto se reservan 65000 bytes.
va-universal-ocsp-url	URL del validador OCSP para todos los tipos de certificados
lang	Idioma ('0' para castellano '1' para euskara)
dlgcertsel-certfilter	Especifica el filtro para la lista de certificados que se podrán seleccionar desde el cuadro de dialogo de selección de certificados. Puede especificarse mediante la lista de políticas admitidas mediante la cadena "policy=oid,oid,..."[1] o bien los certificados en concreto mediante su fingerprint sha-1 mediante la cadena "sha1thumb=thumb1,thumb2,..."[2]
dlgcertsel-selectfirst	Si está a "1", en el caso que solo haya un certificado para firmar lo seleccionará automáticamente sin mostrar la ventana de selección de certificado.
dlgcertsel-checkocsp	Omite la advertencia de imposibilidad de comprobación del estado de revocación del certificado seleccionado para el proceso de firma.
dlgcertsel-enableocsp	Permite deshabilitar la comprobación del estado de revocación del certificado firmante. Si está a '0' no se realiza la llamada OCSP, mientras que a '1' (valor por defecto) sí se realiza.

1 p.e. "policy=1.3.6.1.4.1.14777.104.3, 1.3.6.1.4.1.14777.104.4"

2 p.e. "sha1thumb=782f84c6270ab21166c3d797140d495b7245f0a8"

dlgsign-show	Si no está a “1” no muestra información sobre ni el certificado ni los elementos a firmar.
proxy	Especifica si se desea usar proxy. Especifique <server>:<port> o si <server>:<port>:<usr>:<passwd> desea usar autenticación. Una cadena vacía específica que no se desea utilizar proxy.
signature-dettached-ref-uri	Especifica la URI del documento a firmar en las firmas xades dettached.
signature-dettached-ref-type	Especifica el tipo de documento a firmar en las firmas xades dettached.
hash-canonizexmlwithcomments	Funciona igual que el parámetro “signature-canonizexmlwithcomments” pero en este caso hace referencia al método de canonización utilizado en el cálculo del hash externo en las firmas xades dettached... Se debe usar en firmas con entrada que no sean XML.

EJEMPLO

```
idazki.setOption("dlgcertsel-enableocsp", "0");
```

setCryptoStoreAuto

Establece la siguiente lógica en la selección de certificados:

- En sistemas Windows:
    1. Si está definido el parámetro crypto-winusepkcs11 intenta leer en todos los dispositivos criptográficos que tienen interfaz PKCS#11
    2. Además, busca los certificados instalados en el almacén criptográfico de Windows
- En sistemas no-Windows:
    1. Intenta leer en todos los dispositivos criptográficos que tienen interfaz PKCS#11
    2. Si no encuentra ninguno, pregunta por si se desea utilizar un archivo de claves PKCS#12 y en caso de respuesta afirmativa se saca un navegador de directorios para su selección.

Muestra la lista de certificados encontrados al usuario para su selección.

Nota: consulte los parámetros dlgcertsel* en setOption() para personalizar el comportamiento del cuadro de dialogo.

FORMATO DE LA LLAMADA

setCryptoStoreAuto()

EJEMPLO

```
idazki.setCryptoStoreAuto();
```

addInput

Configuración de los elementos de entrada y salida. Se utiliza con los métodos: sign () y hash ().

FORMATO DE LA LLAMADA

addInput(CadenaTexto intype, CadenaTexto in, CadenaTexto outtype, CadenaTexto out)

PARAMETROS

- **intype**: tipo de entrada.
- **in**: entrada.
- **outtype**: tipo de salida.
- **out**: salida.

Los tipos aceptados de entrada (intype) son los siguientes:

Tipo	
file	El parámetro in debe contener el nombre de un archivo.
filechooser	El parámetro in debe ser null.
inline-hash	El parámetro in debe contener la codificación en base64 de un hash.
inline-binary	El parámetro in debe contener la codificación en base64 de un contenido binario.
inline-text	El parámetro in debe contener un contenido de texto.
Folder	El parámetro out debe contener el nombre de un directorio. Si se pasa como parámetro '*gui' se seleccionará el directorio con un filechooser

Los tipos aceptados de salida (outtype) son los siguientes:

Tipo	
file	El parámetro out debe contener el nombre de un archivo.
folder	El parámetro out debe contener el nombre de un directorio. Si se pasa como parámetro '*gui' se seleccionará el directorio con un filechooser
inline	El resultado de la operación será accesible con getOutputContent().
filechooser	El parámetro out debe ser null.

EJEMPLOS

Para seleccionar como entrada un archivo y salida un archivo.

```
addInput(“file”,"document.pdf",”file”,”newdoc.pdf”);
```

Para seleccionar como entrada un directorio y salida un directorio.

```
addInput(“folder”,"in",”folder”,”out”);
```

Para seleccionar como entrada un hash y salida un archivo.

```
addInput(“hash”,"UE+aTxBNj4…",”file”,”detached.xades”);
```

Para seleccionar como entrada un texto y salida mediante llamadas al applet.

```
addInput(“inline-text”,"helloworld!",”inline”,null);
```

Para indicar a Idazki Desktop que solicite en el momento de la firma el archivo de entrada y de salida.

```
addInput(“filechooser”,null, ”filechooser”, null);
```

Nota: No todos los tipos de input se pueden combinar con todos los tipos de output. La siguente tabla tiene los tipos de inputs en filas y los tipos de outputs en columnas. Una celda verde significa que se puede combinar el tipo de input de la fila de esa celda con el tipo de output de esa columna. Cualquier otro caso es una combinación no valida.

	file	filechooser	inline	folder
file				
filechooser				
inline-hash				
inline-binary				
inline-text				
folder				

addInputExtended

Configuración de los elementos de entrada y salida. Se utiliza con los metodos: sign() y hash().

FORMATO DE LA LLAMADA

addInputExtended(CadenaTexto intype, CadenaTexto in, CadenaTexto outtype, CadenaTexto out, CadenaTexto inputConfig)

PARAMETROS

- **intype**: El tipo de entrada que se proporciona.
- **in**: La entrada proporcionada.
- **outtype**: El tipo de salida que se proporciona.
- **out**: La salida proporcionada.
- **InputConfig**: Un mapa de propiedades en formato JSON que sobrescribe las opciones por establecidas a través del método setOptión.

Nota: En esta versión las propiedades sobreescritas por el inputConfig solo afectan a las firmas PDF y PADES. Es decir, las propiedades que empiezan por 'pdf-signature'.

Para facilitar la generación de los parámetros inputConfig se ha creado una clase de utilidad JavaScript (InputConfig). Los ejemplos se basan en esta clase.

EJEMPLOS

Nota: Dado que los cuatro primeros parámetros de este método funcionan igual que los del método addInput y, que en la sección del método addInput, ya hay ejemplos para el uso de estos cuatro parámetros, en esta sección solo se da ejemplos para el uso del parámetro *inputConfig*.

Para seleccionar como entrada un archivo y salida un archivo.

```
// Creamos el objeto Idazki
var idazki = new Idazki('<<tercero seguro>>');

// Fijamos todas las opciones de idazki
idazki.setOption("pdf-signature-llx","50");
idazki.setOption("pdf-signature-lly","50");
```

```
idazki.setOption("pdf-signature-urx","150");
idazki.setOption("pdf-signature-ury","150");
idazki.setOption("pdf-signature-page","1");
idazki.setOption("pdf-signature-reason","outter reason");
idazki.setOption("pdf-signature-location","outter location");
idazki.setOption("pdf-signature-image-bytes",<<imagen en B64>>);
idazki.setOption("pdf-signature-text","linea1~#~linea2");
idazki.setOption("option-waitprogressdialog","0");
idazki.setOption("dlgsign-show","1");
idazki.setOption("pdf-signature-page","0");
idazki.setCryptoStoreAuto();

var inputConfig = new InputConfig();
inputConfig.addValue("pdf-signature-llx","50");
inputConfig.addValue("pdf-signature-lly","50");
inputConfig.addValue("pdf-signature-urx","350");
inputConfig.addValue("pdf-signature-ury","120");
inputConfig.addValue("pdf-signature-page","0");
inputConfig.addValue("pdf-signature-reason","inner reason");
inputConfig.addValue("pdf-signature-location","inner location");
inputConfig.addValue("pdf-signature-image-bytes",<<imagen en B64>>);
inputConfig.addValue("pdf-signature-text","otralinea1~#~otralinea2");

// Fijamos los elementos a firmar
idazki.addInput("filechooser", null, "filechooser", null);
idazki.addInputExtended(
        "filechooser", null,
        "filechooser", null,
        inputConfig.create());
```

En el ejemplo anterior el primer elemento se firmará con las opciones establecidas mediante la invocación al método *setOptions*, mientras que para el segundo las opciones establecidas mediante el objeto *inputConfig* tienen preferencia sobre las del método *setOption*.

getOutputContent

Recupera el contenido de una operación, cuya entrada ha sido especificada mediante un outtype de tipo “inline”.

FORMATO DE LA LLAMADA

getOutputContent(Entero index, Booleano isBinary, callback)

PARAMETROS

- **index**: el índice del elemento que se quiere recuperar.
- **isBinary**: indica si se quiere recuperar el contenido en base64.
- **callback**: Función a ejecutar una vez el resultado de la firma esté disponible. Cumple el esquema **function(result)**, donde **result** es el resultado de la firma.

EJEMPLO

```
idazki.getOutputContent(0, true, function(result) {

        document.form.texto_salida.value = result;
        idazki.endTransaction();
});
```

sign

Aplica la firma electrónica a los elementos añadidos mediante el método addInput().

FORMATO DE LA LLAMADA

sign(CadenaTexto type, successCallback)

PARAMETROS

- **type**: tipo de firma.
- **successCallback**: Función a ejecutar una vez la firma se haya realizado satisfactoriamente. Cumple el esquema **function(outputData)**, donde **outputData** es “true” en caso de que la firma se realice correctamente. El resultado se obtiene mediante el uso de la función *getOutputContent*.

Type	
cades-sign-attached	Firma CMS/CAdES attached
cades-cosign-attached	Añadir firma paralela a un CMS/CAdES attached
cades-sign-dettached	Firma CMS/CAdES dettached
cades-cosign-dettached	Añadir firma paralela a una firma CAdES dettached
xades-sign-dettached	Firma XMLDSig/XAdES dettached
xades-sign-enveloping	Firma XMLDSig/XAdES enveloping
xades-cosign-enveloped[1]	Crear/Añadir firma paralela a un XMLDSig/XAdES de forma enveloped
pdf	Firma PDF
pdf-timestamped	Firma PDF con sello de tiempo
pades	PAdES-B
pades-timestamped	PAdES-T
facturae	Firma de XMLs tipo Facturae.
cms-cosign-dettached	OBSOLETO, use cades-cosign-dettached
xades-countersign-enveloping	OBSOLETO

EJEMPLO

```
idazki.sign("xades-sign-enveloping", function(outputData) {

        idazki.getOutputContent(0, false, function(result) {
```

1 *Si firma un <Manifest>, el sistema creará una firma de tipo http://www.w3.org/2000/09/xmldsig#Manifest*

```
                document.form.texto_salida.value = result;
                idazki.endTransaction();
        });
});
```

signSetAdESLevel

Establece el nivel de firma para CAdES / XAdES.

FORMATO DE LA LLAMADA

signSetAdESLevel(CadenaTexto level)

PARAMETROS

- **level:** nivel de firma

Type	
none	CMS/XMLDSig
bes	CAdES-BES/XAdES-BES
t	CAdES-T/XAdES-T
c	CAdES-C/XAdES-C
xl	XAdES-XL

EJEMPLO

```
idazki.signSetAdESLevel("bes");
```

hash

Calcula el hash de los elementos añadidos mediante el método addInput()

FORMATO DE LA LLAMADA

hash(successCallback)

PARAMETROS

- **successCallback:** Función a ejecutar una vez el hash se haya calculado satisfactoriamente. Cumple el esquema **function(outputData)**, donde **outputData** es “true” en caso de que la firma se realice correctamente. El resultado se obtiene mediante el uso de la función *getOutputContent*.

EJEMPLO

```

```

```
idazki.hash(function() {

        idazki.getOutputContent(0, true, function(result) {

                document.form.texto_salida.value = result;
                idazki.endTransaction();
        });
});
```

signXAdESEnvelopedzelectNode

Permite firmar, con formato XAdES. Este tipo de llamada cae fuera de la lógica addInput/Sign() ya que no muestra ninguna ventana para la validación de la firma a efectuar. Se permite especificar los nodos a firmar del xml de entrada y la posición donde será ubicada la firma.

FORMATO DE LA LLAMADA

signXAdESEnvelopedSelectNode(*CadenaTexto* xmlEntrada, *CadenaTexto* nodoAFirmar, *CadenaTexto* nodoPosiciónFirma, *CadenaTexto* mecanismoCannonicalización, String xmlEntradaEncoding, successCallback)

PARAMETROS

- **xmlEntrada**: El documento XML que se desea firmar.
- **nodoAFirmar**: El nodo que se desea firmar, puede especificarse:
  - o bien la referencia al ID del documento: p.e. #id
  - o bien un xpath[1] identificando el nodo: p.e. //node. En el caso que dicho nodo no tenga un identificador, este será añadido de manera automática.
- **nodoPosiciónFirma**: El nodo donde se desea insertar la firma, puede especificarse:
  - o bien la referencia al ID del documento: p.e. #id
  - o bien un xpath[2] identificando el nodo: p.e. //node
- **mecanismoCannonicalización**: Especifica el mecanismo que se desea utilizar para la transformación de canonicalización[3]. Debe especificar uno de los siguientes valores:

1 Consulte http://www.w3.org/TR/xpath/

2 Consulte http://www.w3.org/TR/xpath/

3 Consulte http://www.w3.org/TR/xmldsig-core/

- TRANSFORM_C14N_OMIT_COMMENTS
- TRANSFORM_C14N_WITH_COMMENTS
- TRANSFORM_C14N_EXCL_OMIT_COMMENTS
- TRANSFORM_C14N_EXCL_WITH_COMMENTS

- **xmlEntradaEncoding**: Especifica el encoding del documento xml de entrada.
- Debe especificar uno de los siguientes valores:
  - ISO-8859-1
  - UTF-8
- **successCallback**: Función a ejecutar una vez la firma se haya realizado satisfactoriamente. Cumple el esquema **function(outputData)**, donde **outputData** es el resultado de la firma.

EJEMPLO

```
idazki.signXAdESEnvelopedSelectNodeBase64(document.form.texto_entrada.value, "#documento5", "#respuesta5",
"TRANSFORM_C14N_OMIT_COMMENTS", "utf-8", function(outputData) {
        document.form.texto_salida.value = outputData;
        idazki.endTransaction();
});
```

signXAdESEnvelopedSelectNodeBase64

Permite firmar, con formato XadES un documento xml codificado en base 64 obteniendo la firma también en base64. Este tipo de llamada cae fuera de la lógica addInput/Sign() ya que no muestra ninguna ventana para la validación de la firma a efectuar. Se permite especificar los nodos a firmar del xml de entrada y la posición donde será ubicada la firma.

FORMATO DE LA LLAMADA

signXAdESEnvelopedSelectNodeBase64(CadenaTexto xmlEntrada, CadenaTexto nodoAFirmar, CadenaTexto nodoPosiciónFirma, CadenaTexto mecanismoCannonicalización, String xmlEntradaEncoding, successCallback)

PARAMETROS

- **xmlEntrada**: El documento XML que se desea firmar codificado en base64.
- **nodoAFirmar**: El nodo que se desea firmar, puede especificarse:
  - o bien la referencia al ID del documento: p.e. #id

- o bien un xpath[1] identificando el nodo: p.e. //node. En el caso que dicho nodo no tenga un identificador, este será añadido de manera automática.

- **nodoPosiciónFirma**: El nodo donde se desea insertar la firma, puede especificarse:
  - o bien la referencia al ID del documento: p.e. #id
  - o bien un xpath[2] identificando el nodo: p.e. //node
- **mecanismoCannonicalización**: Especifica el mecanismo que se desea utilizar para la transformación de canonicalización[3]. Debe especificar uno de los siguientes valores:
  - TRANSFORM_C14N_OMIT_COMMENTS
  - TRANSFORM_C14N_WITH_COMMENTS
  - TRANSFORM_C14N_EXCL_OMIT_COMMENTS
  - TRANSFORM_C14N_EXCL_WITH_COMMENTS
- **xmlEntradaEncoding**: Especifica el encoding del documento xml de entrada. Debe especificar uno de los siguientes valores:
  - ISO-8859-1
  - UTF-8
- **successCallback**: Función a ejecutar una vez la firma se haya realizado satisfactoriamente. Cumple el esquema **function(outputData)**, donde **outputData** es el resultado de la firma codificado en base64.

EJEMPLO

```
idazki.signXAdESEnvelopedSelectNodeBase64(document.form.texto_entrada.value, "#documento5", "#respuesta5",
"TRANSFORM_C14N_OMIT_COMMENTS", "utf-8", function(outputData) {
        document.form.texto_salida.value = outputData;
        idazki.endTransaction();
});
```

1 Consulte http://www.w3.org/TR/xpath/

2 Consulte http://www.w3.org/TR/xpath/

3 Consulte http://www.w3.org/TR/xmldsig-core/

endTransaction

Finaliza la transacción en curso en el Tercero. Es importante que una vez el tramite completo de firma haya finalizado notifiquemos al tercero de confianza mediante esta función para que dé por finalizada la transacción.

FORMATO DE LA LLAMADA

endTransaction()

EJEMPLO

El cierre de la transacción hay que realizarlo tanto cuando la operación se ha realizado satisfactoriamente como cuando se produce un error en la misma. Por tanto, en caso de éxito habrá que cerrarla después de haber obtenido el resultado de la firma:

```
idazki.getOutputContent(0, false, function(result) {
        document.form.texto_salida.value = result;
        idazki.endTransaction();
});
```

Y en caso de error habrá que cerrarla una vez capturado el código del error:

```
idazki.setErrorCallback(function(errorCode) {
        document.form.texto_salida.value = errorCode;
        idazki.endTransaction();
});
```

getToken

Obtiene el token asociado a la translación. El token sirve para:

- Identificar la transacción y los recursos asociados a la misma.
- Como mecanismo de seguridad, asegurando que no se puede acceder a los recursos sin el token correspondiente.

FORMATO DE LA LLAMADA

getToken()

RETORNO

El token asociado a la translación.

EJEMPLO

```
var token = idazki.getToken();
```

sayHello

Realiza una validación del entorno de cliente para tratar de averiguar si el PC del cliente tiene instalado y registrado el idazki-protocol

FORMATO DE LA LLAMADA

sayHello(callbackFunction successCallBack, callbackFunction errorCallBack, Integer timeout, IdazkiJS self)

PARAMETROS

successCallback: Función a ejecutar una vez se haya realizado satisfactoriamente la validación del entorno de cliente. Este callback debe implementar la siguiente interfaz successCallback(success, idazkiObj).

errorCallback: Función a ejecutar si no se ha podido ejecutar la validacion del entorno de cliente. Este callback debe implementar la siguiente interfaz errorCallback(error, idazkiObj).

timeout: ms que se debe esperar antes de decidir que se ha producido un error en la validación.

idazkiJS: el propio objeto en sí para enviarlos como parte del callback, sirve también (dada la naturaleza del js) para enviar cualquier elemento y que entre en el scope de los callbacks.

RETORNO

void

EJEMPLO

```
idazki.sayHello(onHelloSuccess,onHelloError, 5000, idazki);

function onHelloSuccess(success, idazkiObj)
{
        console.log(success);
        idazkiObj.endTransaction();
        sign();
}

function onHelloError(error, idazkiObj)
{
        console.log(error);
        idazkiObj.endTransaction();
        alert("Parece que no tienes instalado el componente … bla bla bla");
}
```

setSignTimeOut

Establece un tiempo máximo de respuesta para la operación de firma (en milisegundos).

FORMATO DE LA LLAMADA

setSignTimeout(int timeout)

PARAMETROS

- **timeout**: timeout en milisegundos.

EJEMPLO

```
var idazki = new Idazki("http://localhost:8080/idazki-protocol-services");
idazki.setSignTimeOut(10000);
idazki.setOption("dlgcertsel-enableocsp", "0");
idazki.signSetAdESLevel("bes");
idazki.setCryptoStoreAuto();
...
```

1.3.2 Uso de la librería

En primera instancia debemos declarar la librería javascript. Para ello hacemos referencia al fichero **idazki.js** facilitado en el código fuente, así como a la librería jQuery (1.11.3) necesaria para el funcionamiento del componente.

```
<head>
...
        <script type="text/javascript" src="../js/jquery-1.11.3.min.js"></script>
        <script type="text/javascript" src="../js/idazki.js"></script>
...
</head>
```

Una vez hecho esto debemos implementar mediante el API descrito en el punto anterior la funcionalidad de firma deseada. A continuación, se adjunta un ejemplo comentado del uso del API para la realización de una firma XAdES Enveloping con perfil BES.

```
// 1-  inicializamos el componente indicando la url del tercero.
var idazki = new Idazki("http://localhost:8080/idazki-protocol-services");

// 2-  configuramos la opciones generales y el método de carga de certificados.
idazki.setOption("dlgcertsel-enableocsp", "0");
idazki.signSetAdESLevel("bes");
idazki.setCryptoStoreAuto();

// 3-  configuramos el comportamiento para el tratamiento de errores.
idazki.setErrorCallback(function(errorCode) {
        document.form.texto_salida.value = errorCode;
        idazki.endTransaction();
});

// 3- configuramos la entrada / salida pasando el texto a firmar.
```

```
idazki.addInput("inline-text", document.form.texto_entrada.value,                    "inline", null);

// 4- iniciamos en el proceso de firma indicando el tipo.
idazki.sign("xades-sign-enveloping", function(outputData) {

        // 5- obtenemos el resultado de la firma
        idazki.getOutputContent(0, false, function(result) {
                document.form.token.value = idazki.getToken();
                document.form.texto_salida.value = result;
                idazki.endTransaction();
        });
});
```

1.3.3 Consideraciones adicionales

Dependiendo de las características y de la configuración del entorno del usuario, desde que el usuario acciona el botón de firmar hasta que la aplicación **Idazki Desktop** se inicia pueden pasar varios segundos. Por tanto es recomendable para mejorar la usabilidad del servicio, incluir un mensaje de texto para indicar al usuario de tal circunstancia (o bien un indicador de espera). Esta tarea debe realizarse en la aplicación donde va a ser integrado **Idazki Desktop**, de modo que es responsabilidad del integrador incluir dicha funcionalidad (mediante html / javascript, por ejemplo).

1.3.4 Ejemplos de uso del API javascript

XAdES Enveloping BES (sin ocsp) - Text

Firma XAdES BES Enveloping con la validación OCSP deshabilitada.

```
var idazki = new Idazki("http://localhost:8080/idazki-protocol-services");

idazki.setOption("dlgcertsel-enableocsp", "0");
idazki.signSetAdESLevel("bes");
idazki.setCryptoStoreAuto();
```

```
idazki.setErrorCallback(function(errorCode) {
        document.form.texto_salida.value = errorCode;
        idazki.endTransaction();
});

idazki.addInput("inline-text", document.form.texto_entrada.value, "inline", null);

idazki.sign("xades-sign-enveloping", function(outputData) {

        idazki.getOutputContent(0, false, function(result) {
                document.form.token.value = idazki.getToken();
                document.form.texto_salida.value = result;
                idazki.endTransaction();
        });
});
```

XAdES Enveloping XL - Con política - File

Firma XAdES XL Enveloping con política de firma.

```
var idazki = new Idazki("http://localhost:8080/idazki-protocol-services");

idazki.signSetAdESLevel("xl");
idazki.setOption("ades-add-policy", "1");
idazki.setCryptoStoreAuto();

idazki.setErrorCallback(function(errorCode) {
        document.form.texto_salida.value = errorCode;
        idazki.endTransaction();
});

idazki.addInput("filechooser", null, "filechooser", null);

idazki.sign("xades-sign-enveloping", function(outputData) {
```

```
        idazki.getOutputContent(0, false, function(result) {
                document.form.token.value = idazki.getToken();
                idazki.endTransaction();
        });
});
```

XAdES Enveloped Select Node - Id - Text

Firma XAdES Enveloped especificando el nodo a firmar.

```
var idazki = new Idazki("http://localhost:8080/idazki-protocol-services");

idazki.setOption("dlgcertsel-enableocsp", "0");
idazki.signSetAdESLevel("bes");
idazki.setCryptoStoreAuto();

idazki.setErrorCallback(function(errorCode) {
        document.form.texto_salida.value = errorCode;
        idazki.endTransaction();
});

idazki.signXAdESEnvelopedSelectNode(document.form.texto_entrada.value, "#documento5", "#respuesta5",
"TRANSFORM_C14N_OMIT_COMMENTS", "utf-8", function(outputData) {
        document.form.token.value = idazki.getToken();
        document.form.texto_salida.value = outputData;
        idazki.endTransaction();
});
```

XAdES Enveloped Select Node - Id - Base64

Firma XAdES Enveloped especificando el nodo a firmar.

```
var idazki = new Idazki("http://localhost:8080/idazki-protocol-services");

idazki.setOption("dlgcertsel-enableocsp", "0");
idazki.signSetAdESLevel("bes");
idazki.setCryptoStoreAuto();

idazki.setErrorCallback(function(errorCode) {
```

```
        document.form.texto_salida.value = errorCode;
        idazki.endTransaction();
});

idazki.signXAdESEnvelopedSelectNodeBase64(document.form.texto_entrada.value, "#documento5", "#respuesta5",
"TRANSFORM_C14N_OMIT_COMMENTS", "utf-8", function(outputData) {
        document.form.token.value = idazki.getToken();
        document.form.texto_salida.value = outputData;
        idazki.endTransaction();
});
```

XAdES Detached T - InlineHash

Firma XAdES Detached indicando el Hash del documento precalculado de forma externa.

```
var idazki = new Idazki("http://localhost:8080/idazki-protocol-services");

idazki.setOption("signature-hashalgo", "SHA-1");
idazki.setOption("signature-dettached-ref-type", "hash");
idazki.signSetAdESLevel("t");
idazki.setCryptoStoreAuto();

idazki.setErrorCallback(function(errorCode) {
        document.form.texto_salida.value = errorCode;
        idazki.endTransaction();
});

idazki.addInput("inline-hash", document.form.texto_entrada.value, "inline", null);

idazki.sign("xades-sign-dettached", function(outputData) {

        idazki.getOutputContent(0, false, function(result) {
                document.form.token.value = idazki.getToken();
                document.form.texto_salida.value = result;
                idazki.endTransaction();
```

```
        });
});
```

Firma Facturae en versión 3.2.1.

```
var idazki = new Idazki("http://localhost:8080/idazki-protocol-services");

idazki.setOption("dlgcertsel-enableocsp", "0");
idazki.setCryptoStoreAuto();

idazki.setErrorCallback(function(errorCode) {
        document.form.texto_salida.value = errorCode;
        idazki.endTransaction();
});

idazki.addInput("filechooser", null, "filechooser", null);

idazki.sign("facturae", function() {

        document.form.token.value = idazki.getToken();

        idazki.getOutputContent(0, true, function(result) {

                idazki.endTransaction();
        });
});
```

PDF - ISO 32000 - File

Firma de un documento PDF.

```
var idazki = new Idazki("http://localhost:8080/idazki-protocol-services");

idazki.setOption("dlgcertsel-enableocsp", "0");
idazki.setCryptoStoreAuto();
```

```
idazki.setErrorCallback(function(errorCode) {
        document.form.texto_salida.value = errorCode;
        idazki.endTransaction();
});

idazki.addInput("filechooser", null, "filechooser", null);

idazki.sign("pdf", function() {

        document.form.token.value = idazki.getToken();

        idazki.getOutputContent(0, true, function(result) {

                idazki.endTransaction();
        });
});
```

PDF - PAdES T - File

Firma PAdES de un documento PDF.

```
var idazki = new Idazki("http://localhost:8080/idazki-protocol-services");

idazki.setCryptoStoreAuto();

idazki.setErrorCallback(function(errorCode) {
        document.form.texto_salida.value = errorCode;
        idazki.endTransaction();
});

idazki.addInput("filechooser", null, "filechooser", null);

idazki.sign("pades-timestamped", function() {
```

```
        document.form.token.value = idazki.getToken();

        idazki.getOutputContent(0, true, function(result) {

                idazki.endTransaction();
        });
});
```

CAdES - C - File

Firma CAdES.

```
var idazki = new Idazki("http://localhost:8080/idazki-protocol-services");

idazki.signSetAdESLevel("c");
idazki.setCryptoStoreAuto();

idazki.setErrorCallback(function(errorCode) {
        document.form.texto_salida.value = errorCode;
        idazki.endTransaction();
});

idazki.addInput("filechooser", null, "filechooser", null);

idazki.sign("cades-sign-attached", function() {

        document.form.token.value = idazki.getToken();

        idazki.getOutputContent(0, true, function(result) {

                idazki.endTransaction();
        });
});
```

Hash Action – Text

Calcula el hash del texto introducido.

```
var idazki = new Idazki("http://localhost:8080/idazki-protocol-services");

idazki.setErrorCallback(function(errorCode) {
        document.form.texto_salida.value = errorCode;
        idazki.endTransaction();
});

idazki.addInput("inline-text", document.form.texto_entrada.value, "inline", null);

idazki.hash(function() {

        document.form.token.value = idazki.getToken();

        idazki.getOutputContent(0, true, function(result) {

                document.form.texto_salida.value = result;
                idazki.endTransaction();
        });
});
```

Full Server - Firma PDF - Gestión de ficheros en Servidor - Paso 0

Este método permite la firma de documentos de más de 4MB (PDFs por ejemplo). Este caso de uso, es distinto a los anteriores, ya que tanto el documento a firmar, como la firma resultante, se gestionan en el lado del servidor. Este ejemplo se compone de 2 pasos:

Paso 1. En primer lugar, iniciamos la transacción en el servidor, y especificamos el documento a firmar, mediante la función *"inputfile-upload/".* Es importante mencionar que el tipo de "*intype"* y de "*outtype* " debe ser "*url".*

ServerPdfFileServlet.java

```
// - comenzamos la transacion, obteniendo el token
HttpPost post = new HttpPost(RestServicesUtil.getServerUrl(https) + "/" +
        RestServicesUtil.TransactionServerUrl + "/"
                                    + RestServicesUtil.Services.StartTransaction.getServiceName());
HttpClient client = HttpClientBuilder.create().build();
HttpResponse httpResp = client.execute(post);

String token = "";
if (httpResp.getStatusLine().getStatusCode() == HttpStatus.SC_OK) {
        InputStream is = httpResp.getEntity().getContent();
        token = IOUtils.toString(is);
        log.trace("start-transaction: " + token);
        is.close();
} else {
        log.error(httpResp.getStatusLine().getStatusCode());
}



String idx = "0";
HttpEntity entity = MultipartEntityBuilder
                                .create()
                                .addTextBody("intype", "url")
                                .addTextBody("in", "null")
                                .addTextBody("outtype", "url")
                                .addTextBody("out_folder", "null")
                                .addBinaryBody("file", new File(filePath), ContentType.create("application/octet-
stream"), "filename")
                                .build();

post = new HttpPost(RestServicesUtil.getServerUrl(https) + "/" + RestServicesUtil.TransactionServerUrl + "/inputfile-
upload/" + token + "/" + idx + "/add");
client = HttpClientBuilder.create().build();
post.setEntity(entity);
httpResp = client.execute(post);

if (httpResp.getStatusLine().getStatusCode() == HttpStatus.SC_OK) {
        log.trace("upload file ok");
} else {
        log.error(httpResp.getStatusLine().getStatusCode());
```

```
}
```

Paso 1.1. Una vez hecho esto, en esa misma petición, debemos utilizar la librería *idazki.js* para iniciar el proceso de firma de Idazki Desktop. En este caso, no se debe utilizar el método *addInput* como en el resto de ejemplos, ya que, ya hemos especificado previamente en el servlet el documento a firmar. De modo que, únicamente debemos especificar, el token y la configuración de la firma deseada.

server_pdf_file.jsp

```
var idazki = new Idazki("<%= RestServicesUtil.getServerUrl(https) %>", "<%= token %>");
idazki.setOption("dlgcertsel-enableocsp", "0");
idazki.setCryptoStoreAuto();

idazki.setErrorCallback(function(errorCode) {
        document.form.texto_salida.value = errorCode;
        // en este test no se finaliza transacción en cliente, si no en servidor.
});

idazki.sign("pdf", function() {

        document.form.token.value = idazki.getToken();

        idazki.getOutputContent(0, true, function(result) {

                document.form.upload.removeAttribute("disabled");
        });
        // en este test no se finaliza transacción en cliente, si no en servidor.
});
```

Paso 2. Una vez que el usuario haya realizado la firma mediante Idazki Desktop, mediante una segunda petición, podremos obtener la firma en el servidor, y cerrar la transacción.

ServerPdfFileResultServlet.java

```
String token = (String) req.getParameter("token");
log.trace("token: " + token);
```

```
// consultamos el documento firmado
String idx = "0";
HttpGet get = new HttpGet(RestServicesUtil.getServerUrl(https) + "/" + RestServicesUtil.TransactionServerUrl + "/" +
token + "/signaturefile/" + idx + "/get");
HttpClient client = HttpClientBuilder.create().build();
HttpResponse response = client.execute(get);
String serverResponse = "";

if (response.getStatusLine().getStatusCode() == HttpStatus.SC_OK) {
        InputStream is = response.getEntity().getContent();
        // Y lo copiamos a filesystem
        IOUtils.copy(is, new FileOutputStream(filePath));

        log.trace("get-signature-file ok");

} else {
        log.error(response.getStatusLine().getStatusCode());
}

// consultamos config firma
HttpGet get2 = new HttpGet(RestServicesUtil.getServerUrl(https) + "/" + RestServicesUtil.TransactionServerUrl + '/' +
token + '/' + RestServicesUtil.Services.GetConfiguration.getServiceName());
HttpClient client2 = HttpClientBuilder.create().build();
HttpResponse response2 = client2.execute(get2);
String serverResponse2 = "";
if (response2.getStatusLine().getStatusCode() == HttpStatus.SC_OK){
        InputStream is2 = response2.getEntity().getContent();
        serverResponse2 = IOUtils.toString(is2);
        log.trace("get-configuration: " + serverResponse2);
        is2.close();
} else {
        log.error(response2.getStatusLine().getStatusCode());
}

// terminamos transacción
HttpPost post = new HttpPost(RestServicesUtil.getServerUrl(https) + "/" + RestServicesUtil.TransactionServerUrl + '/' +
token + '/' + RestServicesUtil.Services.RemoveTransaction.getServiceName());
HttpClient client3 = HttpClientBuilder.create().build();
HttpResponse response3 = client3.execute(post);
if (response3.getStatusLine().getStatusCode() == HttpStatus.SC_OK){
        InputStream is3 = response3.getEntity().getContent();
```

```
        String serverResponse3 = IOUtils.toString(is3);
        log.trace("remove-transaction: " + serverResponse3);
        is3.close();
} else {
        log.error(response3.getStatusLine().getStatusCode());
}
```

Nota: El código del servidor depende del entorno utilizado (servidor de aplicaciones, librerías...), de modo que, se incluye a modo de referencia.

1.4 Idazki Desktop – Aplicación de escritorio

Idazki Desktop es la aplicación que se encarga de registrar el protocolo (en el proceso de instalación), atender las URI con dicho protocolo, y realizar la firma en el equipo del usuario.

En la mayoría de los casos la aplicación no requiere ningún tipo de configuración. Aunque sí dispone de un fichero de propiedades en el cual es posible configurar algunos aspectos como los siguientes:

- Proxy
- Tiempos de conexión para ficheros grandes
- Habilitar el cacheo de PIN

El fichero de propiedades se encuentra en la siguiente ruta:

{IDAZKI_INSTALL_DIR}/izenpesigner-desktop.properties

1.4.1 Proxy

En el caso del applet Idazki al ejecutarse bajo el contexto del navegador la gestión del proxy se realizaba a través del mismo. En **Idazki Desktop** se ha tenido que implementar un nuevo módulo para la gestión del proxy. Este nuevo módulo auto detecta la configuración del proxy del equipo, de modo que en la mayoría de los casos no debería ser necesaria configuración alguna.

izenpesigner-desktop.properties

```
proxyAuto=true
proxyAutoUser=
proxyAutoPassword=
proxy=
```

Como se puede apreciar también es posible especificar un usuario y un password. En caso de dejarlos en blanco (tal y como vienen en la configuración por defecto) se mostrará un dialogo para pedir dichas credenciales en el momento de la firma.

1.4.2 Tiempos de conexión para ficheros grandes

En el caso de tener que firmar ficheros grandes, es posible que Idazki Desktop pueda cortar las conexiones a la hora de transferir los ficheros, y por lo tanto dar un error. Para evitar esta situación, se pueden definir los tiempos de conexión y de lectura, en el fichero de configuración. Lo valores vienen dados en mili-segundos

Estos valores dependerán de la velocidad de conexión de la red, y del tamaño del fichero a firmar. Por defecto se fijan en 60 segundos.

izenpesigner-desktop.properties

```
connectTimeOut=60000
readTimeOut=60000
```

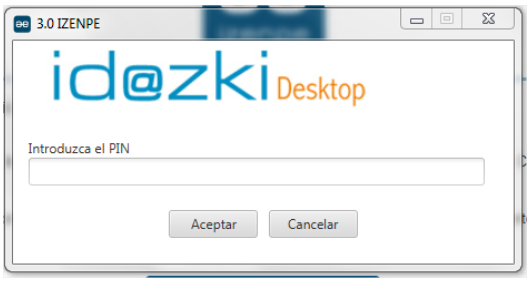
1.4.3 Habilitar cacheo de PIN

A partir de Idazki Desktop v3.0, se permite habilitar el cacheo de PIN a la hora de utilizar las librerías PKCS#11, de forma que no sea necesario introducir el PIN cada vez que se quiera firmar un nuevo documento. El tiempo de cacheo del PIN es configurable, siendo el tiempo máximo permitido 8hrs. El valor viene dado en mili-segundos y establecido por defecto a 4hrs.

izenpesigner-desktop.properties

```
enablePinCache=false
maxPinSession=14400000
```

Mediante la configuración de estos parámetros, se evitará mostrar la siguiente ventana del aplicativo Idazki Desktop.



Este primer cacheo se realiza a nivel de la aplicación Idazki Desktop. Para que la librería PKCS#11 también realice el cacheo de PIN y mantenga la sesión abierta es necesario modificar las siguientes propiedades del fichero de configuración del middleware de Izenpe, denominado *bit4ipki.dll.conf* en Windows, *libbit4ipki.so.conf* en Linux y *libbit4ipki.dylib.conf* en Mac (con las últimas versiones del Middleware el nombre se ha modificado a *bit4xpki.<so>.conf*):

```
dspiniscnspin=true
DSPinUseGui=false
HideCacheDsPinCheck=true
```

```
iDSPinUseGui=false
oDSPinUseGui=false
```

1.4.4 Indicar mecanismo de firma inicial en Windows

Con la última versión de Idazki Desktop a través de Giltz@, es posible indicar (en sistemas operativos Windows) el mecanismo de firma que va a tener prioridad: MSCAPI o PKCS#11. Se trata de facilitar al usuario la usabilidad del aplicativo dependiendo de la lógica de negocio que esté aplicando en un momento u otro. El mecanismo de firma a utilizar por defecto es MSCAPI.

izenpesigner-desktop.properties

```
enablePkcs11=false
```

1.4.5 Librerías PKCS#11

Siempre que se utilice Idazki Desktop a través de la plataforma de autenticación y firma Giltz@, se hará uso del fichero *pkcs11.properties* ubicado en la carpeta de instalación de éste.

{IDAZKI_INSTALL_DIR}/pkcs11.properties

Se trata de un fichero con el listado de las librerías PKCS#11 que se pueden utilizar a la hora de realizar un proceso de firma. En caso de necesitar alguna otra librería que no se encuentre en el fichero, bastará con incluir una nueva línea y reiniciar Idazki Desktop.

1.4.6 Sobrescritura de logs

A partir de Idazki Desktop v3.3.1, se permite indicar si las trazas que se generan se incluyen en un nuevo fichero (sobrescribiendo el existente) o que se utilice el fichero creado con anterioridad, adjuntando las nuevas trazas a las ya existentes. El valor por defecto será 'false', es decir, habrá sobrescritura del log.

izenpesigner-desktop.properties

```
log4jAppend=false
```

1.5 Tercero de confianza

Como se ha comentado con anterioridad para que la solución completa sea operativa se requiere de un servicio que actué como Tercero de confianza, para gestionar toda la comunicación y el intercambio de información entre la aplicación Web e Idazki Desktop. Por tanto los proveedores que deseen incorporar este nuevo sistema de firma por protocolo necesitarán desplegar un servicio que actué como Tercero. En el código fuente suministrado se encuentra la implementación de referencia de Tercero.

El proyecto consiste básicamente en un WAR estándar que contiene un servicio REST desarrollado bajo JAX-RS. Se ha tratando de desarrollar el proyecto de la forma más estándar posible, para facilitar la adaptación del mismo a los distintos contenedores o servidores de aplicaciones. El proyecto ha sido testado en el siguiente entorno:

- Tomcat 7.0.26
- Java 1.7.0_80
- Apache 2.2

1.5.1 HTTPS

La solución implementada ha sido desarrollada para que soporte la comunicación a través de HTTPS. De modo que se recomienda por motivos de seguridad el uso de HTTPS para la comunicación entre el cliente javascript (idazki.js), la aplicación Web y el Tercero. Por tanto a grandes rasgos debemos hacer lo siguiente:

1. Configurar la aplicación Web para que opere sobre HTTPS.
2. Configurar el Tercero para que opere sobre HTTPS.
3. Configurar la URL del Tercero en el cliente javascript para que acceda al tercero. Esto se realiza en la inicialización del objeto “Idazki” de la siguiente forma:

```
var idazki = new Idazki("https://dominio/idazki-protocol-services");
```

1.5.2 CORS

Si la aplicación Web que integra los servicios de firma (donde se alojará la librería *idazki.js*) y el Tercero se encuentran en distintos dominios, es importante tener en cuenta que se requiere habilitar el acceso a los

servicios del Tercero en el frontal correspondiente de acuerdo al estándar CORS (Cross-origin resource sharing). De lo contrario la librería javascript por motivos de seguridad no podrá acceder a los servicios del Tercero para la ejecución de los procesos de firma.

1.5.3 Servicios REST

Como se ha comentado el Tercero de confianza es un servicio REST estándar. El acceso a los servicio REST está encapsulado en la librería de javascript **idazki.js,** de modo que actualmente no es necesario ni se recomienda el acceso directo a los mismos.

A continuación se detallan las especificaciones de cada uno de los métodos del servicio.

Path	**/version/info/get**
Tipo	GET
Descripción	Devuelve un objeto JSON con datos de versionado para el control de versiones en la aplicación.
Entrada	N/A
Salida	**JSON** (IdazkiProtocolVersionVO): • **clientMinVersion (String):** Versión mínima soportada del cliente. • **clientMaxVersion (String):** Versión máxima soportada del cliente. • **restServerMinVersion (String):** Versión mínima soportada por el servicio. • **restServerMaxVersion (String):** Versión máxima soportada por el servicio. • **restServerLastVersion (String):** Última versión del servicio. • **clientLastVersion (String):** Última versión del cliente. • **clientRepositoryUrl (String):** Url de descarga de las nuevas versiones.

Path	**/transactions/add**
Tipo	POST
Descripción	Crea una nueva transacción
Entrada	N/A
Salida	**String**: Id de la transacción.

Path	/transactions/{transaction-id}/error/add
Tipo	POST
Descripción	Almacena el error producido en Idazki Desktop en el servicio.
Entrada	• **transaction-id (String):** Id de la transacción. • **error-message:** código de error.
Salida	N/A

Path	/transactions/{transaction-id}/error/get
Tipo	GET
Descripción	Devuelve el código del error producido en Idazki Desktop.
Entrada	• **transaction-id (String):** Id de la transacción.
Salida	**TEXT**: Código del error producido en Idazki Desktop.

Path	/transactions/{transaction-id}/remove
Tipo	POST
Descripción	Elimina la translación indicada.
Entrada	• **transaction-id (String):** Id de la transacción.
Salida	N/A

Path	/transactions/{transaction-id}/status/get
Tipo	GET
Descripción	Obtiene el código de estado de la transacción.
Entrada	• **transaction-id (String):** Id de la transacción.
Salida	**INTEGER**: Código de estado de la transacción.

Path	/transactions/{transaction-id}/input/count/get
Tipo	GET

Descripción	Obtiene la cantidad de inputs / outputs asociados a la transacción.
Entrada	• **transaction-id (String):** Id de la transacción.
Salida	**INTEGER**: Cantidad de inputs / outputs asociados a la transacción.

Path	**/transactions/{transaction-id}/signature/add**
Tipo	POST
Descripción	Almacena el resultado de la firma en el servicio.
Entrada	• **transaction-id (String)***:* Id de la transacción. • **idx (Integer)**: Índice del input asociado. • base64-signature-result (String): Resultado de la firma en base64.
Salida	N/A

Path	**/transactions/{transaction-id}/signature/{idx}/get**
Tipo	GET
Descripción	Obtiene el resultado de la firma del indice indicado.
Entrada	• **transaction-id (String):** Id de la transacción. • **idx (Integer)**: Índice del input asociado.
Salida	**JSON**

Path	**/transactions/{transaction-id}/signature/{idx}/get/content/{binary}**
Tipo	GET
Descripción	Obtiene el resultado de la firma del índice indicado.
Entrada	• **transaction-id (String):** Id de la transacción. • **idx (Integer)**: Índice del input asociado. • **binary (Boolean):** indica el formato de salida: base64 (true) o texto plano (false).
Salida	**String:** Resultado de la firma.

Path	**/transactions/{transaction-id}/configuration/get**

Tipo	GET
Descripción	Obtiene la configuración general asociada a la translación.
Entrada	• **transaction-id (String):** Id de la transacción.
Salida	**JSON:** Configuración general asociada a la translación.

Path	**/transactions/{transaction-id}/configuration/add**
Tipo	POST
Descripción	Almacena la configuración general de la transacción.
Entrada	• **transaction-id (String):** Id de la transacción. • configuration-as-json (String): Configuración general de la transacción.
Salida	N/A

Path	**/transactions/{transaction-id}/action/configuration/get**
Tipo	GET
Descripción	Obtiene la configuración especifica de la acción a realizar.
Entrada	• **transaction-id (String):** Id de la transacción.
Salida	**JSON:** Configuración especifica de la acción a realizar.

Path	**/transactions/{transaction-id}/action/configuration/add**
Tipo	POST
Descripción	Almacena la configuración especifica de la acción a realizar.
Entrada	• **transaction-id (String):** Id de la transacción. • **configuration-as-json (String):** Configuración especifica de la acción a realizar.
Salida	N/A

Path	**/transactions/{transaction-id}/action/output/get**
Tipo	GET

Descripción	Obtiene el resultado de la acción ejecutada.
Entrada	• **transaction-id (String):** Id de la transacción.
Salida	**String:** Resultado de la acción ejecutada.

Path	**/transactions/{transaction-id}/action/output/add**
Tipo	POST
Descripción	Almacena el resultado de la acción ejecutada.
Entrada	• **transaction-id (String):** Id de la transacción. • action-output (String): Resultado de la acción ejecutada.
Salida	N/A

Path	**/transactions/{transaction-id}/input/add**
Tipo	POST
Descripción	Almacena un la configuración de input / output de un elemento.
Entrada	• **transaction-id (String):** Id de la transacción. • **idx (Integer)**: Índice del input / output. • **intype (String):** Tipo de input. • **in (String):** Contenido del input. • **outtype (String):** Tipo de output. • **out_folder (String):** Contenido del output.
Salida	N/A

Path	**/transactions/{transaction-id}/input/{idx}/get**
Tipo	GET
Descripción	Obtiene un la configuración de input / output de un elemento.
Entrada	• **transaction-id (String):** Id de la transacción. • **idx (Integer)**: Índice del input / output.
Salida	**JSON:** Configuración de input / output de un elemento.

Path	**/transactions/inputfile-upload/{transaction-id}/{idx}/add**
Tipo	POST Multi-part
Descripción	Almacena el documento, y la configuración de input / output de un elemento. Solo para uso desde el servidor.
Entrada	• **transaction-id (String):** Id de la transacción. • **idx (Integer)**: Índice del input / output. • **intype (String):** Tipo de input. • **in (String):** Contenido del input. • **outtype (String):** Tipo de output. • **out_folder (String):** Contenido del output. • **file(binary):** Documento a firmar.
Salida	N/A

Path	**/transactions/{transaction-id}/signaturefile/{idx}/get**
Tipo	GET
Descripción	Obtiene el fichero con la firma del índice indicado. Solo para uso desde el servidor.
Entrada	• **transaction-id (String):** Id de la transacción. • **idx (Integer)**: Índice del input asociado.
Salida	**Binary:** Fichero con la firma.

1.5.4 DAO - Persistencia de las transacciones

La implementación de referencia del Tercero dispone de un sistema de gestión de las transacciones que persiste los datos en memoria. La persistencia de este sistema ha sido desarrollado para que sea “plugable” pudiendo así adaptase a las necesidades de los distintos proveedores.

Para ello se ha centralizado todo acceso a la capa de persistencia en una interfaz JAVA que habrá que implementar para modificar dicho comportamiento. Para dar de alta una nueva implementación por tanto es necesario realizarlo lo siguiente:

1. Desarrollar una clase que implemente la interfaz **DaoTransactionManager.**
2. Declarar dicha clase en el fichero **protocol-service.properties.**

Existen tres implementaciones actualmente.

com.izenpe.idazki.protocol.services.dao.DaoTransactionManagerInMemoryImpl

com.izenpe.idazki.protocol.services.dao.DaoTransactionManagerRelationalDataBaseImpl

com.izenpe.idazki.protocol.services.dao.DaoTransactionManagerJndiImpl

Una persiste los datos en memoria y las otras en una base de datos (la persistencia de base de datos se apoya en una tabla que debe ser creada con el script correspondiente (accesible en la zona de descargas) Los posibles motores de base de datos son

1. Oracle (oracle-create-table.ddl)
2. MySQL (mysql-create-table.ddl)
3. MS-SQLServe (sqlserver-create-table.ddl)

El fichero de propiedades se encuentra en la siguiente ruta:

***/com/izenpe/idazki/protocol/services/properties/*protocol-service.properties**

```
dao.transaction.impl.class=com.izenpe.idazki.protocol.services.dao.DaoTransactionManagerRelationalDataBaseImp
relational.db.driver.class.name=oracle.jdbc.driver.OracleDriver
relational.db.url=jdbc:oracle:thin:@oracle1024.zylk.net:1521:orcl10
relational.db.user=idazki
relational.db.password=idazki
relational.db.table.name=transactions
relational.db.table.obj.field=obj
relational.db.table.key.field=uuid
relational.db.jndi.name=jdbc/nombre
files.db.base.path=/home/zylk/files-repository
```

Como se puede observar por defecto se encuentra configurada la clase DaoTransactionManagerRelationalDataBaseImpl

que es una es una implementación que persiste los datos una base de datos.

El properties se leerá de cualquier path que esté en el classapath, y dependiendo de la impl que se use se deberán proporcionar valores a las propiedades.

Para la impl en memoria, ningún valor del tipo relational.db debe ser informado.

Para la impl por conexión directa, todos los valores menos el jndi.name

Para la impl por jndi, el nombre jndi, nombre de tabla, nombre de campo obj y nombre de key. Los scripts de creación de tablas deberán ajustarse a los nombres de los campos y tabla especificados en el properties, por defecto se usan obj, uuid y transactions.

Por otro lado, para la persistencia de ficheros se utiliza un path de filesystem (solo en caso de utilizar la gestión de ficheros en el servidor). Este path debe ser configurado correctamente mediante el atributo "files.db.base.path".

Y el código de la interfaz **DaoTransactionManager** a desarrollar es la siguiente:

DaoTransactionManager.java

```
public interface DaoTransactionManager {

        public void add(String uuid) throws DaoTransactionException;

        public String addDocument(InputStream bytes, String extension, String                    uuid)
throws DaoTransactionException;

        public void remove(String uuid) throws DaoTransactionNotStartedException;

        public IdazkiProtocolTransactionVO get(String uuid) throws
        DaoTransactionException, DaoTransactionNotStartedException;

        public List<IdazkiProtocolTransactionVO> getAll() throws
        DaoTransactionException, DaoTransactionNotStartedException;

        public IdazkiProtocolTransactionVO update(String uuid,
        IdazkiProtocolTransactionVO transaction) throws DaoTransactionException,
```

```
        DaoTransactionNotStartedException;

        public IdazkiProtocolTransactionVO updateAddInput(String uuid, Integer idx, String
intype, String in, String outtype, String out_folder) throws DaoTransactionException,
DaoTransactionNotStartedException;

        public void updateInputFile(String uuid, Integer idx, InputStream data) throws
DaoTransactionException, DaoTransactionNotStartedException;

        public InputStream getInputFile(String uuid, Integer idx) throws DaoTransactionException,
DaoTransactionNotStartedException;

        public void updateSignatureFile(String uuid, Integer idx, InputStream data) throws
DaoTransactionException, DaoTransactionNotStartedException;

        public InputStream getSignatureFile(String uuid, Integer idx) throws
DaoTransactionException, DaoTransactionNotStartedException;

}
```

A continuación se explica el funcionamiento de cada una de las funciones de la interfaz.

Método	**add**
Descripción	Crea una nueva transacción.
Entrada	• **uuid (String):** Identificador único de la transacción.
Salida	N/A
Excepciones	• **DaoTransactionException:** Excepción genérica para indicar errores inesperados.

Método	**addDocument**
Descripción	Almacena un documento y lo asocia a la transacción especificada por el parámetro uuid.
Entrada	• **bytes (InputStream):** Contenido del documento. • **extension (String):** Extensión del documento. • **uuid (String):** Identificador único de la transacción.
Salida	• **String:** Identificador del documento.
Excepciones	• **DaoTransactionException:** Excepción genérica para indicar errores inesperados.

Método	**remove**
Descripción	Borra la transacción especificada por el parámetro uuid.
Entrada	• **uuid (String):** Identificador único de la transacción.
Salida	• N/A
Excepciones	• **DaoTransactionNotStartedException:** Indica que la transacción no se encuentra en el estado “comenzada”.

Método	**get**
Descripción	Devuelve la transacción especificada por el parámetro uuid.
Entrada	• **uuid (String):** Identificador único de la transacción.
Salida	• **IdazkiProtocolTransactionVO:** Objeto con la información de la transacción asociada al uuid indicado.

Excepciones	• **DaoTransactionException:** Excepción genérica para indicar errores inesperados.

Método	**getAll**
Descripción	Devuelve un listado con todas las transacciones activas.
Entrada	• N/A
Salida	• **List<IdazkiProtocolTransactionVO>:** Listado de transacciones activas.
Excepciones	• **DaoTransactionException:** Excepción genérica para indicar errores inesperados.

2 Guía de migración Idazki Applet → Idazki Desktop

Este punto esta destinado a aquellos proveedores que tenían el applet Idazki integrado es sus aplicaciones y desean migrar al nuevo sistema de firma por protocolo. Para poder migrar al nuevo sistema el proveedor necesita:

- Adaptar el código del applet existente en la aplicación Web.
- Disponer de un servidor **Tercero** operativo.
- Que el usuario disponga de la aplicación **Idazki Desktop** correctamente instalada en el equipo.

La puesta en marcha del Tercero ya ha sido cubierta en el punto anterior, y la instalación de Idazki Desktop es una tarea que recae sobre el usuario, de modo que en este apartado nos vamos a centrar en la adaptación del código javascript. Este código antes se hacía mediante acceso al API del applet y ahora debemos utilizar la librería idazki.js. Como para el diseño de la librería javascript se ha tenido muy presente el API del applet la adaptación del código no debería ser nada complejo.

Nota: Para poder asegurar un correcto funcionamiento en firmas cuyos parámetros contenga caracteres especiales es necesario que todos los participantes implicados en el proceso de firma usen UTF-8 como encoding.

2.1.1 Función "sign"

La función **sign** funciona exactamente igual que en Idazki, es decir mediante el uso de las funciones addInput y getOutputContent. De modo que la migración se puede realizar de la siguiente forma.

Código con el applet:

```
var applet = document.applet;

applet.clearInputs();
applet.setOption("dlgcertsel-enableocsp", "0");
applet.signSetAdESLevel("bes");
```

```
applet.addInput("inline-text", document.form.texto_entrada.value, "inline", "null");

var loadOk = applet.setCryptoStoreAuto();

if (loadOk) {
        var signOk = applet.sign("xades-sign-enveloping");
        if (signOk) {
                document.form.texto_salida.value = applet.getOutputContent(0, false);
         } else {
                document.form.texto_salida.value = applet.getLastError();
         }
} else {
        document.form.texto_salida.value = applet.getLastError();
}
```

Código correspondiente con idazki.js

```
var idazki = new Idazki("http://localhost:8080/idazki-protocol-services");

idazki.setOption("dlgcertsel-enableocsp", "0");
idazki.signSetAdESLevel("bes");
idazki.setCryptoStoreAuto();

idazki.setErrorCallback(function(errorCode) {
        document.form.texto_salida.value = errorCode;
        idazki.endTransaction();
});

idazki.addInput("inline-text", document.form.texto_entrada.value, "inline", null);

idazki.sign("xades-sign-enveloping", function(outputData) {

        idazki.getOutputContent(0, false, function(result) {
                document.form.token.value = idazki.getToken();
                document.form.texto_salida.value = result;
                idazki.endTransaction();
```

```
        });
});
```

2.1.2 Función “signEnvelopedSelectNodeBase64”

Por otro lado la migración de método *signEnvelopedSelectNodeBase64* se puede realizar de la siguiente forma.

Código con el applet:

```
var applet = document.applet;

applet.clearInputs();
applet.setOption("dlgcertsel-enableocsp", "0");
applet.signSetAdESLevel("bes");

var loadOk = applet.setCryptoStoreAuto();
if (loadOk) {
        document.form.texto_salida.value =
applet.signXAdESEnvelopedSelectNodeBase64(document.form.texto_entrada.value,"#documento5","#respuesta5","TR
ANSFORM_C14N_OMIT_COMMENTS", "utf-8");
} else {
        document.form.texto_salida.value = applet.getLastError();
}
```

Código correspondiente con idazki.js

```
var idazki = new Idazki("http://localhost:8080/idazki-protocol-services");

idazki.setOption("dlgcertsel-enableocsp", "0");
idazki.signSetAdESLevel("bes");
idazki.setCryptoStoreAuto();

idazki.setErrorCallback(function(errorCode) {
        document.form.texto_salida.value = errorCode;
```

```
        idazki.endTransaction();
});

idazki.signXAdESEnvelopedSelectNodeBase64(document.form.texto_entrada.value, "#documento5", "#respuesta5",
"TRANSFORM_C14N_OMIT_COMMENTS", "utf-8", function(outputData) {
        document.form.token.value = idazki.getToken();
        document.form.texto_salida.value = outputData;
        idazki.endTransaction();
});
```

2.1.3 Tratamiento de errores

En el applet Idazki la gestión de errores se realizaba mediante la función **getLastError().** En Idazki Desktop en cambio se ha definido una función con un callback que se ejecutará cuando se produzca un error en cualquier punto de proceso de firma. Los códigos de error son los mismos que en applet.

```
idazki.setErrorCallback(function(errorCode) {
        document.form.texto_salida.value = errorCode;
        idazki.endTransaction();
});
```

2.1.4 Filechooser

El método del applet **promptFilesystem** no está soportado en Idazki.js, en su defecto se ha implementado un nuevo tipo de Input / Output, denominado “filechooser”.

```
idazki.addInput("filechooser", null, "filechooser", null);
```

Mediante esta sentencia Idazki Desktop se encargará de preguntar al usuario el fichero a firmar, y en segunda instancia en lugar donde alojar el resultado de la firma. Este nuevo tipo de entrada / salida puede ser combinado indistintamente con el resto de tipos de Inputs / Outputs (al igual que en Idazki).

3 Manual de Instalación Idazki Desktop

Idazki Desktop es una aplicación de escritorio que posibilita la integración de la firma digital en aplicaciones Web mediante el registro de un nuevo protocolo en el equipo.

3.1 Requisitos de la instalación

- Dispositivos criptográficos correctamente instalados en el caso que se quieran utilizar.
- Acceso a las URLs de TSA y OCSP de Izenpe.

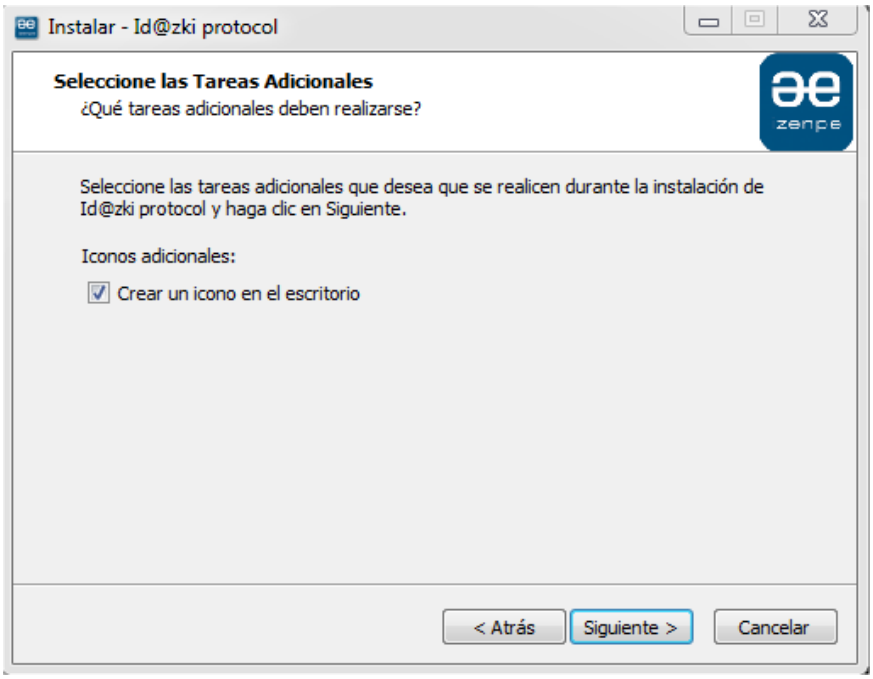
3.2 Guía de instalación

En primera instancia debemos descargar el instalador de la web de Izenpe. Dado que la aplicación es multi-plataforma (Windows, MAC OS y Linux) debemos descargar el empaquetado correspondiente a nuestro sistema operativo:

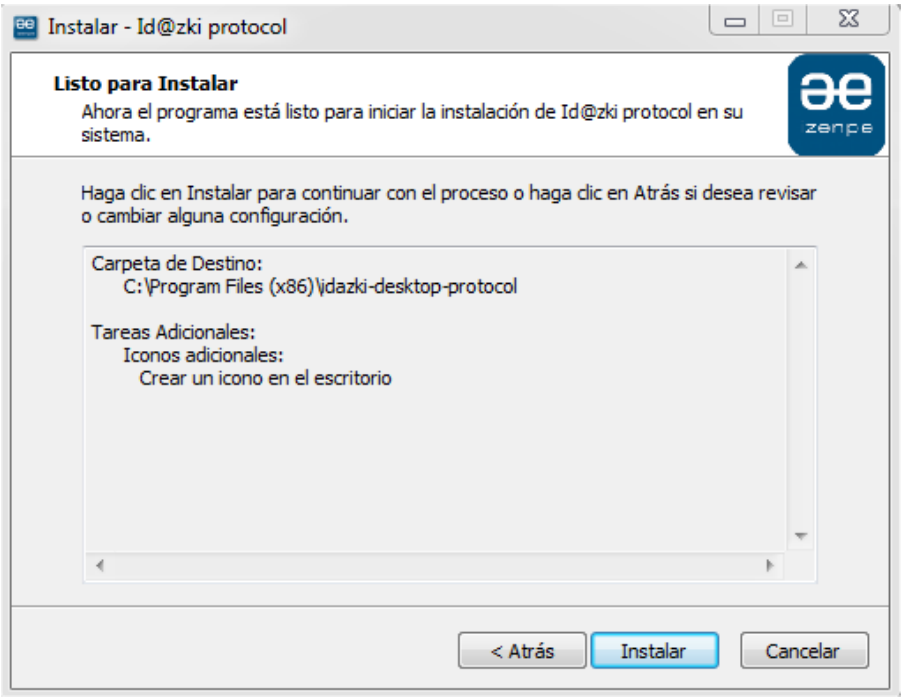
- idazki_2.1.0_macosx_x64_installer.dmg
- idazki-desktop-setup.exe
- idazki-desktop-linux32-install.jar
- idazki-desktop-linux64-install.jar

Una vez descargado debemos hacer doble click sobre el ejecutable el cual iniciará el asistente de instalación:

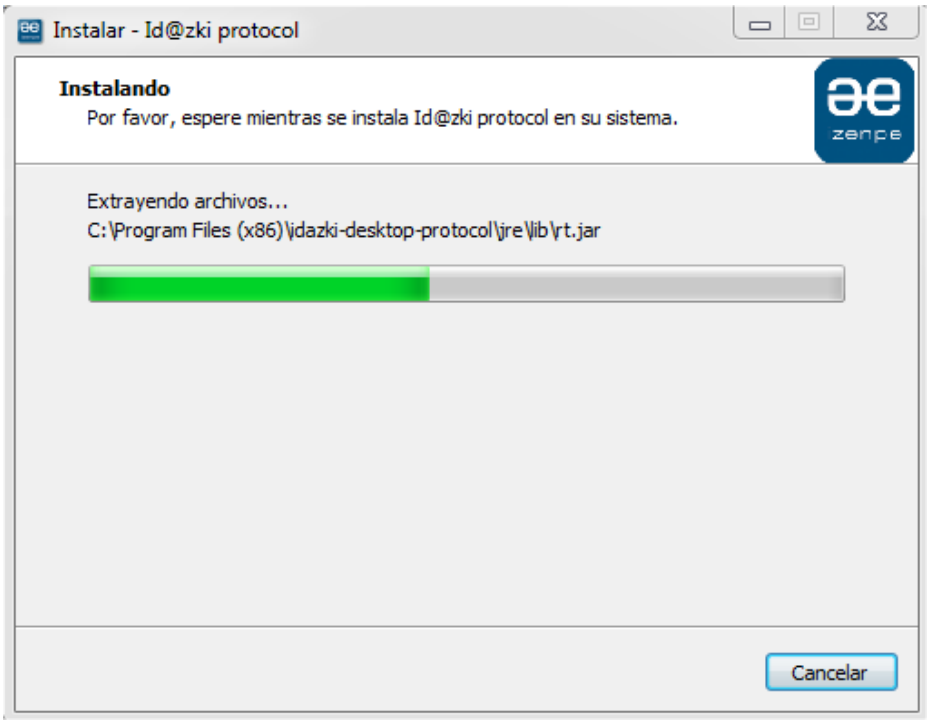
Pulsando el botón **Siguiente,** se mostrarán la opción de creación de un acceso directo en el escritorio del equipo:



En el **siguiente** paso se muestra la información de instalación de la aplicación:

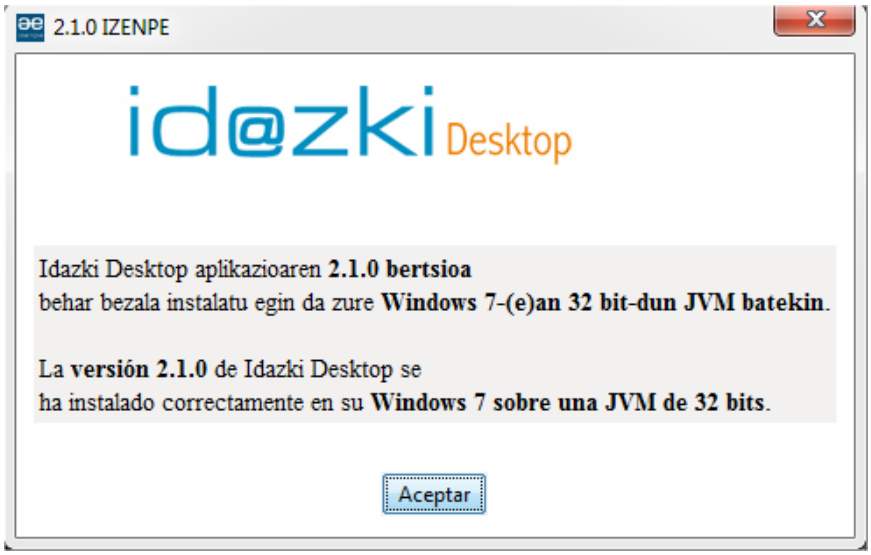


Y por último pulsando sobre el botón **Instalar** se iniciará definitivamente el proceso de instalación.



Cuando el proceso de finalice se muestra la posibilidad de comprobar la instalación.

Con la opción **Ejecutar Idazki protocolo** y pulsando el botón **Finalizar** se mostrará la siguiente ventana.



Una vez hecho esto la aplicación estará correctamente instalada en el equipo, lo que permitirá realizar la firma digital en los servicios que hagan uso de **Idazki Desktop**.

4 Anexo Izenpe

4.1 Cambios en el proceso de carga de properties

En la versión anterior, la carga del fichero de properties se recorre todo el classpath de la aplicación y carga todas las properties de todos los ficheros cuya extensión sea ".properties". Este proceso es una búsqueda por fuerza bruta del posible fichero de properties de Idazki.

Para evitar una búsqueda tan extensa se ha decidido que se establecerá un nombre identificativo del fichero de properties de Idazki. De esta forma el proceso, en lugar de obtener todos los ficheros de properties de classpath, busca sólo el fichero establecido y detiene la búsqueda una vez encuentre una coincidencia.

NOTA: Dado que es necesario también obtener el fichero de configuración del log4j se usará el mismo procedimiento que con el fichero de properties.

Los cambios realizados en la aplicación para articular estos cambios son los siguientes:

1. Se han añadido en el web.xml de la aplicación parámetros de contexto con los nombres establecidos por consenso para el fichero de properties de la aplicación y el fichero de configuración de log4j.

   NOTA: Los nombres actuales son nuestra propuesta

```
<context-param>¶
»   <param-name>idazki-properties-file</param-name>¶
»   <param-value>idazki-protocol-service.properties</param-value>¶
</context-param>¶
<context-param>¶
»   <param-name>log4j-config-file</param-name>¶
»   <param-value>idazki-log4j.properties</param-value>¶
</context-param>¶
```

2. Se ha creado un ServletContextListener que se encarga de la carga de las properties de la aplicación y de la inicialización del log4j. Se ha añadido la entrada del listener en el web.xml **NOTA: Dado que este listener se encarga de la carga de las properties se ha eliminado la referencia al servlet que las cargaba antes.**

```
<listener>¶
»   <listener-class>com.izenpe.idazki.protocol.listener.ContextListener</listener-class>¶
</listener>¶
```

3. El listener se encarga de que, al inicializarse el contexto web, cargar las properties e inicializar el log4j.

   Tanto en el caso de los properties de aplicación como en el caso del fichero del log4j el proceso funciona de la misma manera.

   a. Se busca en classpath un fichero cuyo nombre coincida con el nombre que se está buscando (tanto en directorio raíz como en subdirectorios del classpath). Si se encuentra se devuelve este fichero, si no se devuelve null. **NOTA: En el classpath se compara mediante un String::endsWith por lo que se cualquier fichero cuyo nombre acabe como el**

que se está buscando será considerado un fichero valido. Ej: izenpe-idazki-protocol-service.properties

b. Si en el paso anterior se ha obtenido un fichero se usa ese fichero para cargar las properties o inicializar el log4j (según sea el caso). Si del paso anterior se obtiene un null se obtiene el fichero por defecto (estos ficheros se han depositado en el directorio **/WEB-INF/defaults**)

NOTA: Se ha detectado durante las pruebas que, en el caso de cancelar el permiso de ejecución de la aplicación de escritorio, la aplicación cliente está generando llamadas sin parar al método *getTransactionStatus* de la clase *IdazkiProtocolRestServices* del servicio REST lo que genera una traza cada 2 segundos. Una vez que se cierra la página que ha realizado la prueba se paran las llamadas y la generación de trazas

El contenido de los ficheros **por defecto** es el siguiente:

- **Para fichero de properties** se define para que la transacción se mantenga en **memoria**. Se ha decidido usar esta versión porque de esta forma no hay que establecer ningún parámetro para configurar la BBDD. Como contra mencionar que este tipo de gestión de transacciones da problemas en los clusters.
- **Para el fichero de log4j** se ha definido un appender que saque las trazas por consola.