Índice de contenidos

[1 Introducción 3](#_Toc3197935)

[2 Alcance funcional de los *plugins* 3](#_Toc3197936)

[2.1 Integración en el proceso de firma 4](#_Toc3197937)

[2.1.1 Preprocesando los datos que se van a firmar 4](#_Toc3197938)

[2.1.2 Postprocesado de la firma generada 5](#_Toc3197939)

[2.2 Integración en las pantallas de firma 5](#_Toc3197940)

[2.3 Configuración de los *plugins* 7](#_Toc3197941)

[3 Desarrollo de *plugins* 8](#_Toc3197942)

[3.1 Construir un *plugin* 8](#_Toc3197943)

[3.1.1 Estructura del *plugin* 14](#_Toc3197944)

[3.2 Definir opciones de configuración 15](#_Toc3197945)

[3.2.1 Acceso a las opciones de configuración 19](#_Toc3197946)

[3.3 Implementar los métodos de instalación y desinstalación del *plugin* 19](#_Toc3197947)

[3.4 Implementar los métodos de preproceso de datos, postproceso de firma y/o fin de operación 20](#_Toc3197948)

[3.5 Introducir un botón con acción 21](#_Toc3197949)

[4 Preguntas y respuestas 24](#_Toc3197950)

[4.1 ¿Es posible realizar la instalación masiva de un *plugin* en los equipos de mis usuarios? 24](#_Toc3197951)

# Introducción

AutoFirma es una herramienta de escritorio con interfaz gráfica que permite la ejecución de operaciones de firma de ficheros locales en entornos de escritorio (Windows, Linux y OS X). También puede utilizarse a través de consola o ser invocada por otras aplicaciones mediante protocolo para la ejecución de operaciones de firma. Esta última funcionalidad puede usarseprincipalmente mediante el JavaScript de despliegue del Cliente @firma, que permitiría que se utilizase AutoFirma para generar las firmas de un trámite web.

AutoFirma incluye un sistema básico de *plugins* que permiten modificar el comportamiento de las operaciones de firma y proporcionar nueva funcionalidad cuando se utiliza como aplicación de escritorio. Es decir, los *plugins* sólo se activan cuando la aplicación se ejecuta en modo gráfico. No afecta a las operaciones de firma cuando se invoca desde otra aplicación (como un navegador web) ni cuando se ejecuta mediante línea de comandos.

Un *plugin* consiste en una biblioteca Java que extiende una clase concreta y que proporcionar un fichero JSON con la información necesaria para la integración con AutoFirma. El desarrollo del *plugin* requiere labores de programación, mientras que la instalación y desinstalación de los *plugins* puede ser llevada a cabo por los usuarios de AutoFirma.

El mecanismo de *plugins* actual podría modificarse en un futuro para permitir nuevas funcionalidades y opciones de integración.

AutoFirma es una aplicación de Software Libre que se puede usar, a su elección, bajo licencia *GNU General Public License* versión 2 (GPLv2) o superior o bajo licencia *European Software License* 1.1 (EUPL 1.1) o superior.

AutoFirma puede descargarse desde el Portal de Administración Electrónica:

<https://firmaelectronica.gob.es/Home/Descargas.html>

El código fuente de AutoFirma se encuentra disponible desde el repositorio público de GitHub:

<https://github.com/ctt-gob-es/clienteafirma>

# Alcance funcional de los *plugins*

Un usuario puede instalar, configurar y desinstalar *plugins* de AutoFirma.

Al instalar un *plugin*, el fichero de *plugin* se copia en el directorio de AutoFirma, se ejecuta un proceso de instalación definido por el propio *plugin* y, a partir de ese momento, el *plugin* pasa a estar activo y podrá afectar a los procesos de firma y mostrar botones en las pantallas de AutoFirma.

Los *plugins* pueden implementar una interfaz gráfica de configuración. En caso de hacerlo, en el mismo diálogo de gestión de *plugins*, aparecerá un botón de configuración para iniciar esta interfaz. Con ella, el usuario deberá poder establecer los aspectos configurables del *plugin*.

Al desinstalar un *plugin*, se ejecuta un proceso de desinstalación definido por el propio *plugin*, se descarga de memoria y, finalmente, se elimina del directorio de AutoFirma.

Los *plugins* se integran con AutoFirma de las siguientes formas:

* Integración en el proceso de firma.
  + El *plugin* puede afectar a las operaciones de firma de dos formas:
    - Preprocesando los datos a firmar.
    - Postprocesando las firmas generadas.
* Integración en las pantallas de AutoFirma.
  + Pueden crearse acciones y asignarlas a botones que pueden aparecer en las siguientes pantallas:
    - Carga de datos
    - Resultado de firma.
    - Resultado de firma masiva.
    - Visualización de datos de firma.

## Integración en el proceso de firma

Un *plugin* puede intervenir en el proceso de firma preprocesando los datos que se van a firmar o postprocesando las firmas generadas. Este *plugin* es libre de mostrar diálogos gráficos durante su ejecución, ya sean bloqueantes (por ejemplo, cuando se solicite confirmación o un dato al usuario) o no (por ejemplo, un diálogo de espera).

**IMPORTANTE:** No existe limitación al número de *plugins* instalados que pueden efectuar esta operación y su ejecución siempre se realizará según el orden de carga. Es decir, si tuviésemos varios *plugins* que preprocesasen los datos antes de firmar, el primer *plugin* que se cargue podrá modificarlos, el segundo recibirá la salida del primero, el tercero la salida del segundo y así sucesivamente.

Por regla general, sólo se debería tener instalado un *plugin* que preprocese los datos o postprocese la firma y debería ser este el que ejecutase todas las operaciones sobre ambos elementos.

### Preprocesando los datos que se van a firmar

El método de preprocesado de datos nos permite obtener los datos del fichero que cargó el usuario, devolver a la aplicación los datos que realmente se firmarán y reconfigurar la propia operación de firma. Esto es útil si deseamos restringir o condicionar los datos que se firman en nuestro organismo.

Algunos ejemplos en los que se podría necesitar esta operación sería:

* Si los documentos PDF que se firmasen en el organismo siempre deben declarar una serie de metadados.
  + Se podría implementar el método para que detectase cuando la firma a realizar es PAdES, modificase los PDF que vamos a firmar (previa comprobación de que no incluye firmas anteriores), agregase los metadados necesarios y lo devolviese modificado.
* Si se desease que el usuario seleccionase por cada operación si quiere que las firmas CAdES sean implícitas (incluyan los datos) o explicita (no los incluyan).
  + Se podría implementar el método para que detectase cuando la firma a realizar es CAdES, mostrarse un diálogo al usuario de consulta al usuario y, según lo respondido, modificase la configuración para que la firma fuese implícita o explícita.

### Postprocesado de la firma generada

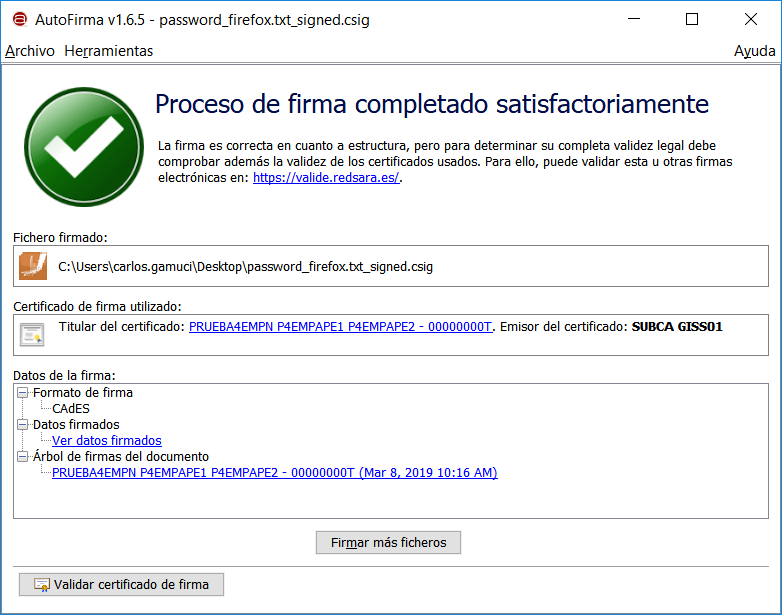
El método de postprocesado de firma nos permite modificar una firma generada antes de que se guarde a disco. Este mecanismo recibe la firma generada y cierta información sobre ella y devuelve los datos que finalmente se almacenaran.

Algunos ejemplos en los que se podría utilizar esta aplicación serían:

* Si todas nuestras firmas deben contener un sello de tiempo, independientemente de que no se presenten ante una entidad.
  + Se podría implementar un método para tomase la firma generada, conectase contra una TSA, generase un sello de tiempo y lo insertase en la firma. Después devolvería la firma ya con el sello.
* Si en nuestro organismo todas las firmas se debiesen guardar en un repositorio o gestor en la nube.
  + Se podría implementar este método para que preguntase al usuario si desea enviar la firma al gestor. Si el usuario respondiese afirmativamente, se podría enviar la firma al servicio remoto. Ya se envíe o no la firma, se devolvería la propia firma y sería esta la que se guardaría en disco.

## Integración en las pantallas de firma

Los *plugins* pueden agregar botones en una barra para tal fin en algunas de las pantallas de AutoFirma. Estos botones pueden ejecutar procesos separados de la propia generación de la firma electrónica y acceder a la información de la pantalla en cuestión.



Las acciones asignadas a los botones agregados pueden mostrar diálogos gráficos durante su ejecución, ya sean bloqueantes (por ejemplo, cuando se solicite confirmación o un dato al usuario) o no (por ejemplo, un diálogo de espera). Incluso pueden mostrar un diálogo que permita realizar operaciones completas totalmente independientes del proceso de firma.

Algunos ejemplos de utilidad para esta función serían:

* Si en nuestro organismo trabajamos con un tipo de fichero concreto, por ejemplo, facturas electrónicas, se podría crear un *plugin* que agregue un botón a la pantalla de carga de datos y que permitiese al usuario que va a firmar pulsarlo para visualizar los datos cargado de tal forma que simplifique determinar si debe firmarse o no. Siguiendo el mismo ejemplo, se podría mostrar una relación con los conceptos y precios de la factura junto con los datos del proveedor.
* Si queremos realizar una validación completa de todas nuestras firmas, se podría implementar un *plugin* que agregue un botón a la pantalla que muestra el resultado de la firma. Al pulsar ese botón se leería la firma y se enviaría a un servicio remoto encargado de validarla.
* Si quisiésemos agregar una funcionalidad a AutoFirma que nada tiene que ver con la firma (por ejemplo, generación de *hashes* de ficheros), se podría implementar un *plugin* que agregue un botón a todas las pantallas que lo permiten y que, al pulsarlo, por ejemplo, muestre un diálogo modal que ofrezca la funcionalidad deseada.

Las pantallas a las que se pueden agregar acciones en botón son:

* Pantalla de carga de datos
  + Permite procesar los datos de entrada y el formato de firma que se va a emplear.
* Pantalla de resultado de una firma simple.
  + Permite procesar la firma generada, el formato de firma empleado, los certificados contenidos por la firma y el certificado utilizado para firmar.
* Pantalla de resultado de un proceso de firma masiva.
  + Permite procesar las firmas generadas, el formato de firma empleado para cada uno de ellos, los certificados contenidos por la firma y el certificado utilizado para firmar.
* Pantalla del visor de firma.
  + Permite procesar la firma mostrada, su formato y el árbol de firmantes contenido en la misma.

## Configuración de los *plugins*

Los *plugins* puede permitir o no su configuración al usuario. Por ejemplo, un *plugin* que valide el certificado de firma utilizado contra su *Distribution Point* (propiedad del propio certificado y en la que se incluye la URL para su validación), podría no necesitar configuración por parte del usuario. En cambio, un *plugin* que valide firmas contra una plataforma externa de la que el usuario deba configurar sus credenciales sí que lo necesitaría.

Las opciones de configuración de un *plugin* que lo permita se encuentran en el diálogo de gestión de *plugins*. En esta pantalla se mostrará un botón para el acceso a las opciones de configuración del *plugin*.

La interfaz gráfica de configuración deberá proporcionarla el propio *plugin* y desde esta se podrán definir y configurar las distintas opciones. AutoFirma se encargará de almacenar las propiedades definidas, no será necesario que lo haga el propio *plugin*, y se las proporcionará a las distintas funciones del *plugins* (operaciones de preproceso de los datos y postproceso de la firma, acciones asignadas a los distintos botones definidos, etc.). Estas mismas opciones se devolverán a la propia interfaz de configuración cuando se vuelva a abrir para que muestre predefinidas las opciones configuradas.

# Desarrollo de *plugins*

Un *plugin* de AutoFirma es una biblioteca Java en la que una de sus clases extiende la clase base de *plugins*. Para extender esta clase, será necesario importar en nuestro proyecto el módulo que contiene las clases base para el uso de los *plugins*. Un desarrollador puede importar este módulo utilizando Apache Maven y la dependencia:

<dependency>

<groupId>es.gob.afirma</groupId>

<artifactId>afirma-ui-simpleafirma-plugins</artifactId>

<version>1.6.5</version>

</dependency>

AutoFirma es compatible con Java 8 y superiores. En concreto, las distribuciones de 32bits de AutoFirma se incorpora Java 8 y las de 64bits con Java 11. Si deseamos que nuestros *plugins* sean compatibles con todas las instalaciones de AutoFirma, deberán implementarse para ser compatibles con Java 8.

Actualmente, AutoFirma sólo soporta *plugins* empaquetados en forma de un único archivo JAR. Los *plugins* que se implementen deberán incluir todas sus clases y dependencias en un único JAR o implementar por su cuenta un mecanismo de carga de dependencias. Nunca se deben insertar en el JAR del *plugin* las clases del módulo “afirma-ui-simpleafirma-plugins” ni ninguna otra clase de AutoFirma, ya que estas ya se encontrarán disponibles en la aplicación.

## Construir un *plugin*

Como se ha mencionado, y salvo que se implementen mecanismos de carga propios, un *plugin* se presenta en forma de un único JAR que contiene todas las clases y recursos que necesita. Ya que en AutoFirma se pueden cargar múltiples *plugins* simultáneamente y desconocemos donde almacenarán sus recursos, cada *plugin* debería establecer un nombre de paquete propio y agregar todas sus clases y recursos en dicho paquete.

Una vez importado el módulo, se deberán seguir los siguientes pasos para desarrollar nuestro *plugin*:

1. **Crear un fichero plugin.json**

Este fichero se agregará al paquete que hayamos definido y contendrá los metadatos de nuestro *plugin* y referencias a todos los elementos que definen su comportamiento (botones que agrega, diálogo gráfico para su configuración, etc). Este fichero deberá codificarse en UTF-8.

La estructura del JSON contendrá dos elementos:

* info: Este contendrá los datos y la configuración del *plugin*. Los elementos que albergará son:
  + name: Nombre interno del *plugin* (sin espacios).
  + title: Nombre legible del *plugin*.
  + version\_code: Opcional. Número natural que identifica la versión del *plugin*. Por defecto, 1.
  + version: Opcional. Cadena con la versión del *plugin* en formato libre. Por defecto, “1.0”.
  + authors: Opcional. Lista con el nombre de los autores del *plugin*.
  + contacts: Opcional. Lista con la información de contacto de los autores.
  + description: Descripción del *plugin*.
  + configuration\_panel: Opcional. Nombre de la clase que implementa el panel de configuración. Si no se indica, no habrá panel de configuración del *plugin*.
* buttons: Opcional. Lista de botones que se deberán mostrar en la interfaz de AutoFirma. Cada botón define:
  + title: Opcional. Texto a mostrar en el botón. El botón debería definir este atributo y/o un icono.
  + icon: Opcional. Imagen a mostrar en el botón a modo de icono. El botón debería definir este atributo y/o un texto.
  + tooltip: Opcional. Texto de ayuda que asociar al botón.
  + accesible\_description: Opcional. Texto accesible que asociar al botón.
  + window: Ventana en la que mostrar el botón.
  + action: Nombre de la clase que define la acción a realizar por el botón.

A continuación, se presenta la declaración formal del JSON mediante su esquema (conforme al borrador 7 del documento de definición de esquemas JSON):

{

"$schema": *"http://json-schema.org/draft-07/schema#"*,

"definitions": {

"info": {

"type": *"object"*,

"properties": {

"name": { "type": *"string"* },

"title": { "type": *"string"* },

"version\_code": { "type": *"integer"*, "default": 1 },

"version": { "type": *"string"*, "default": *"1.0"* },

"authors": {

"type": *"array"*,

"items": { "type": *"string"* },

"default": []

},

"contacts": {

"type": *"array"*,

"items": { "type": *"string"* },

"default": []

},

"description": { "type": *"string"* },

"configuration\_panel": { "type": *"string"* }

},

"required": [*"name"*, *"title"*, *"description"*]

},

"button": {

"type": *"object"*,

"properties": {

"title": { "type": *"string"* },

"icon": { "type": *"string"* },

"tooltip": { "type": *"string"* },

"accesible\_description":{ "type": *"string"* },

"window": {

"type": {

"enum": [

*"input"*,

*"single\_result"*,

*"multi\_result"*,

*"visor"* ]

}

},

"action": { "type": *"string"* }

},

"required": [*"window"*, *"action"*]

}

},

"type": *"object"*,

"properties": {

"info": { "$ref": *"#/definitions/info"* },

"buttons": {

"type": *"array"*,

"items": { "$ref": *"#/definitions/button"* },

"default": []

}

},

"required": [*"info"*]

}

Un ejemplo de “plugin.json” sería:

{

"info":

{

"name" : *"validate\_certs"*,

"title" : *"Validación de certificados"*,

"version\_code" : 1,

"version" : *"1.0"*,

"authors" : [ *"Gobierno de España"* ],

"description" : *"Plugin para la comprobación del estado de certificados de firma.<br><br>Este plugin permite validar los certificados utilizados para firmar con AutoFirma. La validación se realiza mediante mecanismos predefinidos para los certificados de algunos prestadores de servicios de confianza y, en caso de no tener estos mecanismos predefinidos, contra la CRL o el OCSP declarado en el propio certificado.<br><br>Asegúrese de utilizar únicamente certificados emitidos por prestadores de confianza para firmar, ya que un certificado de un origen desconocido podría declarar mecanismos de validación no seguros."*

},

"buttons" : [

{

"title" : *"Validar certificado de firma"*,

"icon" : *"es/gob/afirma/plugin/certvalidation/certificate\_16.png"*,

"tooltip" : *"Validar el certificado con el que se ha realizado la firma"*,

"accesible\_description" : *"Valida el certificado utilizado para firmar"*,

"window" : *"single\_result"*,

"action" : *"es.gob.afirma.plugin.certvalidation.ValidateCertAction"*

},

{

"title" : *"Validar certificado de firma"*,

"icon" : *"es/gob/afirma/plugin/certvalidation/certificate\_16.png"*,

"tooltip" : *"Validar el certificado con el que se ha realizado la firma"*,

"accesible\_description" : *"Valida el certificado utilizado para firmar"*,

"window" : *"multi\_result"*,

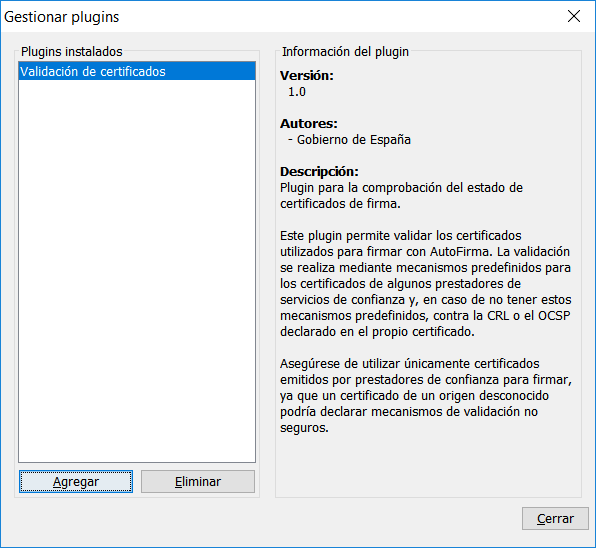
"action" : *"es.gob.afirma.plugin.certvalidation.ValidateCertAction"*

}

]

}

Al instalar este *plugin* en AutoFirma, su aspecto en el diálogo de gestión de *plugins* será:



1. **Implementar AfirmaPlugin**

Todo *plugin* deberá tener una clase que herede de AfirmaPlugin. Esta será la clase principal del *plugin* y deberá encontrarse en el mismo paquete que el fichero “plugin.json”. En esta clase se podrán implementar los siguientes comportamientos:

* Instalación del *plugin*:
  + Se ejecuta una única vez al importar el *plugin* en AutoFirma. Si fuese necesario, en él se deben configurar el sistema y preparar los recursos que requiera el *plugin* para su funcionamiento.
* Desinstalación del *plugin*:
  + Se ejecuta una única vez al eliminar el *plugin* de AutoFirma.
  + En él se deben deshacer las opciones realizadas en el método de instalación.
* Preproceso de los datos
  + Se ejecuta antes de la firma de cada uno de los datos de entrada.
  + Este método permite manipular o realizar operaciones en base a los datos que van a firmarse y devuelve los datos que realmente se firmarán.
* Postproceso de la firma
  + Se ejecuta después de generar una firma electrónica y antes de almacenarla en disco.
  + Este método permite manipular la firma resultante o realizar operaciones en base a ella y devuelve los datos que finalmente se almacenarán en disco. También recibe el formato de firma utilizado y la cadena de certificación del certificado de firma.
* Fin de operación
  + Se ejecuta después de haber finalizado la operación:
    - En el caso de una firma simple, se ejecutará después del postprocesado de la firma.
    - En caso de una firma masiva, se ejecutará una única vez, tras haber postprocesado todas las firmas.
  + Puede usarse si se almacena información en el proceso de postfirma que afecte a todas las firmas, para poder restablecer los datos a sus valores iniciales.
    - Por ejemplo, en una firma masiva, se podría comprobar en el postprocesado si el usuario ha aceptado manipular la firma de alguna manera. Si aún no se le preguntó, se le pregunta; si ya se preguntó, se hace lo que se decidiera. En el método de fin de operación se eliminaría la respuesta del usuario para que en futuras operaciones se volviese a preguntar.

Para saber más sobre cómo implementar los métodos de instalación y desinstalación del *plugin*, consulte el apartado 3.3 Implementar los métodos de instalación y desinstalación del *plugin*.

Para saber más sobre cómo implementar los métodos de preproceso de datos, postproceso de firma y fin de la operación, consulte el apartado 3.4 Implementar los métodos de preproceso de datos, postproceso de firma y/o fin de operación.

Un *plugin* podría no sobrescribir ninguno de los métodos de AfirmaPlugin si no actúa en ninguno de los procesos listados. En ese caso, la clase aparecerá vacía y la interacción del usuario con el *plugin* se deberá realizar a través de los botones declarados en “plugin.json”. Un ejemplo de clase de este tipo sería:

**public** **class** MiNuevoPlugin **extends** AfirmaPlugin {

// No se define ningun comportamiento especial para el plugin

}

1. **Declarar un servicio AfirmaPlugin**

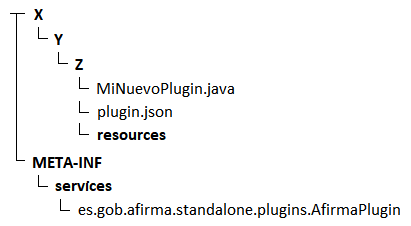
Es necesario notificar a AutoFirma cuál de las clases de nuestra biblioteca es la que implementa la clase AfirmaPlugin. Para hacer esto, anunciaremos nuestra clase como servicio creando en el directorio “META-INF/services” del proyecto el fichero “es.gob.afirma.standalone.plugins.AfirmaPlugin”. El contenido de este fichero debe ser el nombre completo de la clase que implementa AfirmaPlugin.

Por ejemplo,

es.gob.afirma.plugin.certvalidation.ValidateCertsPlugin

### Estructura del *plugin*

Una vez se hayan seguido los pasos anteriores para la construcción básica del *plugin*, la estructura mínima del proyecto debe ser la siguiente:



En el diagrama mostrado:

* **X**, **Y** y **Z**, representan el paquete al que pertenece nuestro *plugin* y que no debería coincidir con el de ningún otro.
* MiNuevoPlugin.java representa a la clase que implementa AfirmaPlugin.
* **resources** sería un directorio en el que podríamos almacenar los recursos (imágenes, ficheros de propiedades o cualquier otra cosa) que requiera nuestro *plugin*. Este directorio no tiene porqué existir, podría no haber recursos que almacenar o incluirse estos directamente en la estructura del paquete sin incluir un directorio adicional; podría tener cualquier otro nombre o podrían separarse los recursos en diversos directorios.

En este caso, el contenido del fichero “es.gob.afirma.standalone.plugins.AfirmaPlugin” sería:

X.Y.Z.MiNuevoPlugin

## Definir opciones de configuración

Diversos tipos de *plugins* podrían permitir al usuario configurar su comportamiento incluso el configurarlos podría ser totalmente necesario para su funcionamiento. Por ejemplo, un *plugin* podría permitir configurar el algoritmo que utilizará al realizar una operación, mientras que otro podría necesitar que el usuario configurase sus credenciales para acceder a un almacén externo.

La configuración de todos los *plugins* deberá establecerse desde el diálogo de gestión de *plugins*. Cuando un usuario seleccione un *plugin* y este disponga de opciones configurables, se mostrará un botón “Configurar”. Al pulsarlo, se mostrará al usuario un diálogo en el que se mostrarán las opciones configurables.

La apariencia de este diálogo y las opciones que contiene se definirá programáticamente mediante la herencia de la clase ConfigurationPanel. Esta clase es un JPanel sobre el que se deberá construir la apariencia del diálogo de configuración. Nuestra clase deberá ser pública e implementar el constructor por defecto (sin parámetros), en el que se agregarán los distintos componentes gráficos que se requieran para la configuración (cajas de texto, listas de selección, cuadros de verificación, etc.) estructurados en directamente en el panel, en subpaneles o en pestañas según se desee.

La clase ConfigurationPanel deberá además implementar los siguientes métodos:

* Properties getConfiguration()
  + Este método construirá un objeto Properties en el que almacenará todos los valores configurados asignándoles un nombre de propiedad a cada uno de ellos.
  + El nombre asignado a cada opción de configuración y con el que se guarda en el Properties será el nombre con el que se podrá recuperar el valor durante la ejecución del *plugin*.
  + Todas las opciones configurables deberían tener una propiedad asignada, incluso si no se estableció un valor para ella, ya que el no hacerlo podría resultar en que se utilicen valores nulos al solicitar las opciones configuradas.
  + Este método se llama al aceptar el diálogo de selección de la configuración y se encarga de guardar todas las propiedades definidas.
* **void** init(Properties config)
  + Este método recibe un objeto Properties con los valores actualmente asignados a las opciones configurables. Si nunca se les ha asignado un valor, devolverá nulo.
  + En este método se deberían obtener los valores configurados en el objeto y establecerlos en las opciones de configuración de nuestro diálogo gráfico.
  + Las propiedades que definen el valor configurado para cada opción de configuración son las que se le hayan asignado en la llamada a getConfiguration().
  + Este método se llama al pulsar el botón “Configurar” para abrir el diálogo de configuración del *plugin*.

Un ejemplo de implementación de diálogo de configuración en el que se permite la configuración de un cuadro de texto sería la siguiente:

**public** **class** MiDialogoConfiguracionPanel **extends** ConfigurationPanel {

**private** **static** **final** String ***PROP\_NOMBRE\_FIRMANTE*** = "nombreFirmante";

**private** **final** JTextField tfNombreFirmante;

**public** MiDialogoConfiguracionPanel() {

// Establecemos un layout

setLayout(**new** GridBagLayout());

// Construimos los distintos componentes graficos

**this**.tfNombreFirmante = **new** JTextField();

**final** JLabel lbNombreFirmante = **new** JLabel("Nombre del firmante:");

lbNombreFirmante.setLabelFor(**this**.tfNombreFirmante);

// Posicionamos los componentes en el panel

**final** GridBagConstraints c = **new** GridBagConstraints();

c.fill = GridBagConstraints.***HORIZONTAL***;

c.weightx = 1.0;

c.gridy = 0;

add(lbNombreFirmante, c);

c.gridy++;

add(**this**.tfNombreFirmante, c);

}

@Override

**public** **void** init(**final** Properties config) {

// Inicializamos el campo con el valor previamente configurado o con

// un valor por defecto si no se encontro

String nombre = config.getProperty(***PROP\_NOMBRE\_FIRMANTE***, "");

**this**.tfNombreFirmante.setText(nombre);

}

@Override

**public** Properties getConfiguration() {

// Creamos el objeto de propiedades y guardamos en el valor del

// campo que permitimos configurar

**final** Properties config = **new** Properties();

config.setProperty(***PROP\_NOMBRE\_FIRMANTE***, **this**.tfNombreFirmante.getText());

**return** config;

}

}

Para poder indicar al *plugin* cuál es nuestra clase de diálogo de configuración estableceremos el nombre de la clase en la propiedad “configuration\_panel” en el apartado “Info” del fichero “plugin.json”.

{

"info":

{

"name" : *"miplugin"*,

"title" : *"Inserción del nombre del firmante"*,

"version\_code" : 1,

"version" : *"1.0"*,

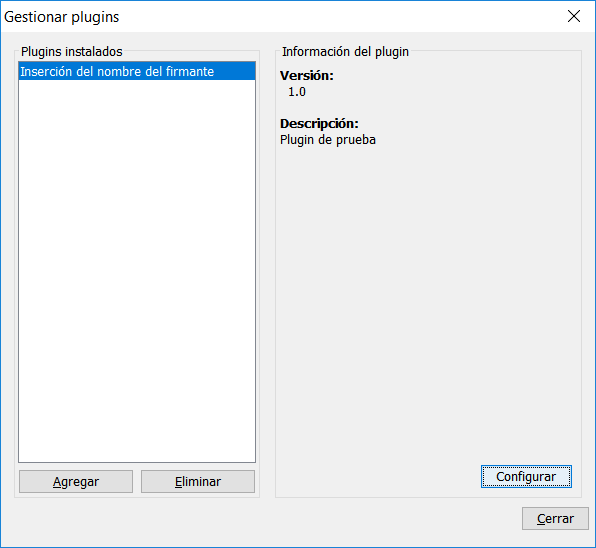
"description" : *"Plugin de prueba"*,

"configuration\_panel" : *"X.Y.Z.MiDialogoConfiguracionPanel"*

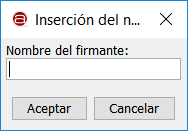
}

…

Con un fichero “plugin.json” con el del ejemplo, se mostraría la información del *plugin* y el botón de configuración como sigue:



En caso de implementar el diálogo de configuración como en el ejemplo, al pulsar el botón de “Configurar” se mostraría:



### Acceso a las opciones de configuración

AutoFirma implementa un mecanismo para la recuperación de la configuración del *plugin* desde cualquier punto de su código. Esto se hará mediante la llamada al método getConfig() de las clases AfirmaPlugin y PluginAction.

Properties getConfig()

Este método devuelve el conjunto de propiedades en el que hayamos almacenado los valores establecidos en nuestro diálogo de configuración.

Así, por ejemplo, desde las acciones de preprocesado de datos, postprocesado de firma o la acción de un botón, podemos hacer una llamada solicitando las opciones de configuración que tenemos establecidas:

@Override

**public** **byte**[] postSignProcess(

**byte**[] signature,

String format,

Certificate[] certChain)

**throws** PluginControlledException {

String firmante = getConfig().getProperty(

MiDialogoConfiguracionPanel.***PROP\_NOMBRE\_FIRMANTE***);

…

}

## Implementar los métodos de instalación y desinstalación del *plugin*

Los *plugins* permiten establecer un procedimiento de instalación y desinstalación que preparen el sistema para su funcionamiento y lo devuelvan a su estado original, respectivamente. Estos procedimientos se implementan en la clase principal del *plugin*, extensión de AfimaPlugin:

* Instalación del *plugin*:
  + Genera o establece la configuración necesaria en el sistema para el funcionamiento del *plugin*.

**void** install() **throws** PluginControlledException

* + Se ejecuta durante el proceso de importación del *plugin* en la aplicación.
  + En caso de error irrecuperable, se lanzaría una excepción de tipo PluginControlledException. En ese caso, se mostraría un mensaje de error al usuario, se desharían las tareas de instalación realizadas y se ejecutaría el proceso de desinstalación establecido por el propio *plugin*.
* Desinstalación del *plugin*:
  + Deshace las acciones realizadas durante el proceso de instalación, además de eliminar cualquier otro recurso agregado o deshacer las acciones realizadas durante el funcionamiento del *plugin*.

**void** uninstall() **throws** PluginControlledException

* + Se ejecuta durante el proceso de desinstalación de un *plugin*.
  + En caso de error irrecuperable, se lanzaría una excepción de tipo PluginControlledException. En ese caso, se mostraría un mensaje de error al usuario y se eliminaría el *plugin* de la aplicación, aunque no se completase el proceso.

## Implementar los métodos de preproceso de datos, postproceso de firma y/o fin de operación

Los métodos de preproceso de los datos antes de firmarlos, postproceso de la firma antes de almacenarla y fin de la operación de firma se implementan en la clase principal del *plugin*, extensión de AfirmaPlugin:

* Preproceso de los datos:
  + Procesa o actúa sobre los datos a firmar.

**byte**[] preSignProcess(**byte**[] data, String format)

**throws** PluginControlledException

* + Recibe los datos que van a firmarse y el formato de firma que se utilizará.
  + Devuelve los datos que finalmente se firmarán.
  + En caso de error irrecuperable, se lanzaría una excepción de tipo PluginControlledException. En ese caso, se mostraría un mensaje de error al usuario, pero el proceso de firma continuará normalmente.
* Postproceso de la firma:
  + Procesa o actúa sobre la firma recién generada.

**byte**[] preSignProcess(**byte**[] data, String format)

**throws** PluginControlledException

* + Recibe los datos que van a firmarse y el formato de firma que se utilizará.
  + Devuelve los datos que finalmente se firmarán.
  + En caso de error irrecuperable, se lanzaría una excepción de tipo PluginControlledException. En ese caso, se mostraría un mensaje de error al usuario, pero el proceso de firma continuará normalmente.
* Finalización del proceso de firma:
  + Finaliza un proceso de firma simple o masiva, restableciendo lo que sea necesario para permitir una futura operación.

**void** reset() **throws** PluginControlledException

* + En caso de error irrecuperable, se lanzaría una excepción de tipo PluginControlledException. En ese caso, se mostraría un mensaje de error al usuario, pero la aplicación continuaría funcionando normalmente.

## Introducir un botón con acción

Los botones de acción son botones que aparecen en diversas pantallas de AutoFirma y que permiten realizar operaciones con la información cargada en dicha pantalla, como los datos que se van a firmar o la firma que ya se ha generado. Estos botones aparecen ordenados y en serie en una barra de botones situada en la zona inferior de AutoFirma. Al pulsar un botón, se desencadena la opción configurada.

Para el uso de botones de acción en un *plugin* es necesario realizar dos pasos:

1. **Implementar la acción del botón**

Las acciones de los botones se implementarán en clases Java que hereden de la clase PluginAction. De esta clase se puede sobrescribir el siguiente método para definir su comportamiento:

**void** start(**final** Window parent)

Sin embargo, si se desea que nuestra acción utilice los datos cargados en la pantalla de AutoFirma en la que se muestre, nuestra acción deberá heredar de alguna de las clases hija de PluginAction:

* DataProcessAction: Permite utilizar los datos cargados en AutoFirma.
* SignatureProcessAction: Permitir utilizar la firma generada o mostrada en AutoFirma y sus certificados.

Según la pantalla en la que deseemos mostrar el botón, se deberá implementar una u otra clase:

* Pantalla de carga de datos: DataProcessAction.
* Pantalla con el resultado de una firma simple: SignatureProcessAction.
* Pantalla con el resultado de una firma masiva: SignatureProcessAction.
* Pantalla de visualización de datos de firma: SignatureProcessAction.

Al heredar de la clase DataProcessAction se debe sobrescribir el método:

**void** processData(InputData[] data, Window parent)

Este método recibe los datos de entrada de la pantalla y la ventana que padre en la que se muestran.

* El listado de objetos de datos de entrada. Este listado contendrá un único elemento en las operaciones de firma simple y varios en las operaciones de firma masiva. Cada uno de estos objetos nos da acceso a:
  + El fichero de datos cargado.
  + El formato de firma con el que se firmarían esos datos.
* La ventana padre nos sirve para utilizarla como ventana padre de cualquier diálogo gráfico que deseemos mostrar.

Al heredar la clase SignatureProcessAction se debe sobrescribir el método:

**void** processSignatures(OutputData[] outputs, X509Certificate signingCert,

Window parent)

Este método recibe los datos de salida de la pantalla, el certificado de firma utilizado y la ventana que padre en la que se muestran los datos.

* El listado de objetos de datos de salida. Este listado contendrá un único elemento en las operaciones de firma simple y varios en las operaciones de firma masiva. Cada uno de estos objetos nos da acceso a:
  + El fichero en el que se encuentra la firma. En caso de no existir, la firma debió finalizar con errores.
  + Si la firma terminó correctamente o no.
  + El formato de la firma.
  + Los certificados contenidos en la firma, que será el certificado utilizado en la firma (si se ha firmado) y los certificados de firmas previas.
* El certificado de firma nos permite saber con qué certificado ha firmado el usuario. Este parámetro podría llegar nulo si el usuario no hubiese firmado, por ejemplo, porque nos encontremos en la pantalla de visualización de firmas, en cuyo caso se cargó una firma que ya estaba hecha y no una que acabe de hacer el usuario.
* La ventana padre nos sirve para utilizarla como ventana padre de cualquier diálogo gráfico que deseemos mostrar.

Por ejemplo, si quisiéramos crear un *plugin* que permitiese validar los certificados de firma, podríamos crear una clase que herede de SignatureProcessAction, que implemente el método processSignatures(), que tomase los certificados de la firma, los enviase a validar en base a su *Distribution Point* y mostrase un mensaje al usuario con el resultado de la validación.

1. **Definir el aspecto del botón y la ventana en la que aparecerá**

Una vez creada la acción del botón deberá registrarse en el *plugin* para indicar dónde aparecerá y su aspecto. Esto se hará desde el apartado “buttons” del fichero plugin.json.

Para cada uno de los botones se podrá definir los siguientes atributos:

* title: Opcional. Texto a mostrar en el botón. El botón debería definir este atributo y/o un icono.
* icon: Opcional. Imagen a mostrar en el botón a modo de icono. El botón debería definir este atributo y/o un texto.
* tooltip: Opcional. Texto de ayuda que asociar al botón.
* accesible\_description: Opcional. Texto accesible que asociar al botón para ser leído por lectores de pantalla. Su uso está especialmente indicado cuando el botón muestra un icono y no texto.
* window: Ventana en la que mostrar el botón. Este atributo debe contener uno de los siguientes valores:
  + input: El botón se mostrará en la pantalla de carga de datos.
  + single\_result: El botón se mostrará en la pantalla de resultado de una operación firma simple.
  + multi\_result: El botón se mostrará en la pantalla de resultado de una operación de firma masiva.
  + visor: El botón se mostrará en la pantalla en la que se visualizan los datos de la firma a través de la opción “Ver firma” o al abrir la firma desde el sistema operativo.
* action: Nombre de la clase que define la acción a realizar por el botón. Esta clase debe heredar de PluginAction o de alguna de las clases que herede de ella.

Por ejemplo, la subestructura JSON para definir un único botón que aparecería en la ventana del resultado de una firma simple sería:

…  
"buttons" : [

{

"title" : *"Validar certificado de firma"*,

"icon" : *"es/gob/afirma/plugin/certvalidation/certificate\_16.png"*,

"tooltip" : *"Validar el certificado con el que se ha realizado la firma"*,

"accesible\_description" : *"Valida el certificado utilizado para firmar"*,

"window" : *"single\_result"*,

"action" : *"es.gob.afirma.plugin.certvalidation.ValidateCertAction"*

}

]  
…

Si se desease que el botón apareciera en varias de las pantallas admitidas sería necesario declararlo múltiples veces en el fichero plugin.json.

Debe recordarse que, si se desean utilizar los datos disponibles en una pantalla, nuestra acción deberá heredar de una clase concreta para esa pantalla (DataProcessAction, SignatureProcessAction…) ya que es la que recoge esos datos de la pantalla. Consulte en el paso 1 de este mismo apartado la acción que deberá heredarse para poder utilizar los datos de cada una de las pantallas que lo admiten.

En caso de querer realizar una misma acción desde diversas ventanas y que los datos de entrada no sean necesarios, podríamos implementar nuestra clase de acción heredando directamente de la clase PluginAction.

# Preguntas y respuestas

## ¿Es posible realizar la instalación masiva de un *plugin* en los equipos de mis usuarios?

No existe un mecanismo particular para la instalación masiva de *plugins* de AutoFirma. Para realizar esto, sería necesario copiar al equipo de los usuarios objetivos el fichero con el *plugin* y modificar el fichero con la relación de *plugins* instalados.

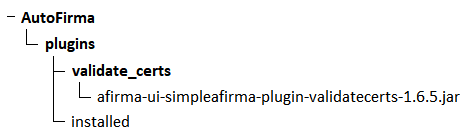
El directorio de instalación de los *plugins* varia de un sistema a otro:

* Directorio Windows:
  + %ALLUSERSPROFILE%/AutoFirma
* Directorio Linux:
  + ~/AutoFirma.
* Directorio macOS:
  + ~/Library/Application Support/AutoFirma

En esos directorios se crea un subdirectorio “plugins” y dentro del mismo se sitúa un fichero con el listado de *plugins* instalados (installed) y un directorio por cada *plugin*. Para instalar programáticamente cada *plugin* es necesario:

1. Crear dentro del directorio “plugins” un subdirectorio con el nombre interno del *plugin* que deseamos instalar (el nombre indicado en la propiedad “name” de su “plugin.json”).
2. Copiar el JAR del *plugin* al directorio creado para el mismo.
3. Agregar al fichero “installed” una nueva línea con el nombre del *plugin* instalado y su número de versión (tal como se indica en la propiedad “version\_code” de su “plugin.json”).

La estructura resultante sería:



El contenido del fichero “installed”, en este caso, podría ser:

validate\_certs:1

[Creative Commons License](#Licencia_Creative_Commons)

Esta obra está bajo una licencia [Creative Commons Reconocimiento-NoComercial-CompartirIgual 3.0 Unported](#Licencia_Creative_Commons).