Manual de Uso de Integr@

Integr@ 2.2.3_001

Documento nº:	@Firma-Integr@-MAN
Revisión:	053
Fecha:	18-10-2022
Período de retención:	Permanente durante su período de vigencia + 3 años después de su anulación

CONTROL DE MODIFICACIONES

Documento nº:	@Firma-Integr@-MAN
Revisión:	053
Fecha:	18-10-2022

Rev.	001
Fecha	19-04-2011
Descripción	Documento inicial.

Rev.	002
Fecha	07-10-2011
Descripción	Se incluyen descripción de los nuevos componentes que forman el API Integr@: signature y utils. Se añade el anexo 2 con ejemplos de firmas de documentos con los diferentes formatos.

Rev.	003
Fecha	29-12-2011
Descripción	Se modifica el documento para la integración con la plataforma eVisor: se incluye referencia a la invocación de los servicios de eVisor y se incluye un anexo de ejemplo de integración.

Rev.	004
Fecha	22-02-2012
Descripción	Se modifica la descripción del módulo de firma. Asimismo se amplia información sobre las nuevas funcionalidades de firma: cofirma y contrafirma.

Rev.	005
Fecha	14-08-2012
Descripción	Se incluyen nuevos parámetros opcionales en las firmas de tipo XAdES relacionados con el formato del documento a firmar.

Rev.	006
Fecha	06-02-2014
Descripción	Se añade la API necesaria para invocar los servicios DSS (sellado, validación de sello, y resellado) y RFC 3161 (sellado) de TS@. Se añade una nota aclaratoria sobre el método getCertificatePrivateKey() en base a la incidencia 291664. Se añade el concepto del idioma a definir para los mensajes de la plataforma. Se añade nota aclaratoria sobre el cacheo del resultado de validación de certificados contra @Firma respecto a sus servicios nativos (español e inglés) y a su servicio DSS.

Rev.	007
Fecha	16-04-2014
Descripción	Se añade una nota aclaratoria sobre el método getCertificatePrivateKey() en el Apartado 10.

Rev.	008
Fecha	22-10-2014
Descripción	Se elimina del apartado Requisitos Software la posibilidad de hacer uso de la JRE 1.5. Sólo se permite el uso de la JRE 1.6.

Rev.	009
Fecha	15-12-2014
Descripción	Se modifica la estructura, apariencia y estilos del documento. Se añade la información asociada a las mejoras en la generación de firmas (Firmas AdES-T, Servicio de Obtención de Sello de Tiempo, Servicio de Actualización de Firmas, Servicio de Generación de Firmas, Integración con HSM, Políticas de Firma), a las mejoras en la validación de firmas y certificados (Cliente OCSP, Validación de Firmantes, Árbol de Firmantes), y a las mejoras en la generación y validación de firmas PAdES (Validación de Firmas PAdES, Firmas Certified y Approval).

Rev.	010
Fecha	12-02-2015
Descripción	Se elimina cualquier referencia a autor. Se añade la información asociada a la funcionalidad que permite validar la integridad de un sello de tiempo a nivel local, o contra TS@, antes de llevar a cabo una solicitu de renovación de sello de tiempo a TS@. Se corrige el pie de página del documento. Se elimina el control de aprobación.

Rev.	011
Fecha	03-03-2015
Descripción	Se modifica el funcionamiento del cliente OCSP, de manera que antes era necesario indicar la ruta al certificado emisor del certificado a validar y ahora se indica la ruta al almacén de claves que almacena los posibles emisores de los certificados a validar, así como el tipo de dicho almacén y su contraseña.

Rev.	012
Fecha	08-05-2015
Descripción	Se añade una aclaración en los métodos es.gob.afirma.integraFacade.IntegraFacade.generateCoSignature(byte[], byte[], String, PrivateKeyEntry) y es.gob.afirma.integraFacade.IntegraFacade.generateCounterSignature(byte[], String, PrivateKeyEntry) para indicar que ambas funcionalidades sólo están permitidas para CAdES y XAdES, no para PAdES.

Rev.	013
Fecha	08-05-2015
Descripción	Se añaden ejemplos de uso de la funcionalidad que permite validar certificados contra un servidor OCSP.

Rev.	014
Fecha	13-05-2015
Descripción	Se añade una propiedad asociada al fichero con los parámetros de configuración para las políticas de firma de manera que se pueda indicar qué modos de firma están permitidos para una determinadas política de firma.

Rev.	015
Fecha	14-05-2015
Descripción	Se añade el formato de firma PAdES-Basic como válido para su generación desde la clase es.gob.afirma.signature.pades.PadesSigner.

Rev.	016
Fecha	14-05-2015
Descripción	Se añade la indicación de que, en los procesos de actualización de firma desde las clases es.gob.afirma.signature.pades.PadesSigner y es.gob.afirma.integraFacade.IntegraFacade, la actualización de firmas PDF, esto es, PAdES-Basic, PAdES-BES, PAdES-EPES y PAdES-LTV, consiste en añadir un diccionario de sello de tiempo, pasando el formato de la firma a PAdES-LTV, lo tuviese antes o no. Se corrigen erratas en la definición de métodos para las clases Java.

Rev.	017
Fecha	14-05-2015
Descripción	Se añade una nota informativa en la Introducción indicando que Integr@ está certificada para PKCS#11 con el HSM de RealSec Cryptosec, integrado en el producto CryptosecLAN.

Rev.	018
Fecha	14-05-2015
Descripción	Se añade una nota informativa para indicar que si en el archivo tsaServiceInvoker.properties se establece el valor de la propiedad 'renewTimeStampWS.validationLevel' a 0, entonces no se cumple con el estándar de OASIS para la renovación de sellos de tiempo.

Rev.	019
Fecha	14-05-2015
Descripción	Se añade la información de un nuevo fichero de propiedades, ocsp.properties, donde ubicar las actuales propiedades de comunicación con el servidor OCSP definidas en el fichero integraFacade.properties.

Rev.	020
Fecha	18-05-2015
Descripción	Se actualiza la descripción de los métodos definidos en la interfaz es.gob.afirma.signature.Signer y en la clase es.gob.afirma.integraFacade.IntegraFacade. Se actualiza la descripción de los métodos validateCertificate y validatePDFSigner de la clase es.gob.afirma.utils.UtilsSignature. Se actualiza la descripción de los métodos verifySignature de las clases es.gob.afirma.signature.cades.CadesSigner, es.gob.afirma.signature.xades.XadesSigner y es.gob.afirma.signature.pades.PadesSigner. Se define un nivel más de validación para los firmantes en el fichero integraFacade.properties.

Rev.	021
Fecha	30-07-2015
Descripción	En el fichero policy.properties se sustituye la propiedad que indicaba la ruta completa al documento legible de la política de firma por otra que indica el valor del hash de la política de firma. El fichero de propiedades integraFacade.properties se modifica y asigna para ser usado únicamente para la fachada de servicios propios IntegraFacade, y se crea un nuevo fichero de propiedades signer.properties para su uso exclusivo por la interfaz Signer. Se actualizan métodos de validación de certificado y firmantes de la clase es.gob.afirma.utils.UtilsSignature para recibir un parámetro nuevo que indica el origen de la invocación al método.
Rev.	022
Fecha	17-11-2015
Descripción	Se modifica el apartado Requisitos Software con la nueva JRE 1.7.
Rev.	023
Fecha	15-12-2015
Descripción	Se añaden métodos en las clases CoSignRequest, CounterSignRequest y ServerSignerRequest para dar soporte a la posibilidad de realizar firmas sobre el hash de un documento y co-firmas y contra-firmas enviando la firma en la petición.
Rev.	024
Fecha	28-01-2016
Descripción	Se modifica documento adaptándose a la nueva estructura dividida de Integra para la versión 2.0.0_000.
Rev.	025
Fecha	08-02-2016
Descripción	Se añaden las descripciones y ejemplos asociados a las nuevas funcionalidades de la plataforma, esto es: - Se añade la generación de firmas, co-firmas y contra-firmas CAdES Baseline en sus formatos B-Level y T-Level, con y sin política de firma, mediante la interfaz Signer y la Fachada de Firma. - Se añade la generación de firmas, co-firmas y contra-firmas XAdES Baseline en sus formatos B-Level y T-Level, con y sin política de firma, mediante la interfaz Signer y la Fachada de Firma. - Se añade la generación de firmas PAdES Baseline en su formatos B-Level, con y sin política de firma, mediante la interfaz Signer y la Fachada de Firma. - Se añade la actualización de firmantes de una firma CAdES Baseline añadiéndoles un sello de tiempo a cada uno de ellos, siempre que no lo tuviera anteriormente, mediante la interfaz Signer y la Fachada de Firma. - Se añade la actualización de firmantes de una firma XAdES Baseline añadiéndoles un sello de tiempo a cada uno de ellos, siempre que no lo tuviera anteriormente, mediante la interfaz Signer y la Fachada de Firma. - Se añade la inclusión de un diccionario de sello de tiempo a un documento PDF según el estándar PAdES Baseline, mediante la interfaz Signer y la Fachada de Firma. - Se añade la validación de firmantes de una firma CAdES Baseline mediante la interfaz Signer y la Fachada de Firma. - Se añade la validación de firmantes de un documento XML que contiene firmas XAdES Baseline

mediante la interfaz Signer y la Fachada de Firma.

- Se añade la validación de firmantes de un documento PDF que contiene firmas PAdES Baseline mediante la interfaz Signer y la Fachada de Firma.

- Se añade la validación de la estructura y la firma XAdES Baseline o CAdES Baseline contenida dentro de una firma ASiC-S mediante la interfaz Signer y la Fachada de Firma.

- Se añade la actualización de firmantes de una firma XAdES Baseline o CAdES Baseline contenida dentro de una firma ASiC-S mediante la interfaz Signer y la Fachada de Firma.

Rev.	026
Fecha	07-03-2016
Descripción	Se modifica el título del punto 8.17.7.3 para indicar, de manera más clara, que el valor de la propiedad con el resumen del documento legible de la política de firma, especificando que deberá estar codificado en Base64.

Rev.	027
Fecha	09-03-2016
Descripción	Se añade la información asociada a la plantillas del servicio de obtención de firmas registradas de @firma en la descripción de las carpetas parserTemplates y xmlTemplates. Se añade la funcionalidad de indicar con una variable (-DIntegra.config) en las opciones de arranque de la máquina virtual de java la ruta donde se ubican los ficheros de configuración. Se añade una indicación para el método counterSign de la interfaz Signer, y para el método generateCounterSignature de la fachada de firma (IntegraFacade), referente a que, en los procesos de contra-firma de firmas XAdES, sólo se llevará a cabo en firmas Detached y, en caso de no serlo (Enveloped o Enveloping), ésta se reconfigurará para ser Detached y admitir la contra-firma. Se modifica el tipo de letra del documento de Frutiger a Calibri. Se añade información de la necesidad de incluir la librería joda-time-1.6.2.jar para los tipos de integración 2, 3 y 5. Se añade información de la necesidad de incluir la librería junit-4.8.2.jar para todos los tipos de integración. Se añade información de la necesidad de incluir la librería slf4j-api-1.6.1.jar para los tipos de integración 2, 3 y 5.

Rev.	028
Fecha	11-03-2016
Descripción	Se añade el nuevo Tipo de integración 6, que permite el cifrado y descifrado de datos mediante algoritmos simétricos y asimétricos.

Rev.	029
Fecha	15-03-2016
Descripción	Se añade la fachada de los servicios de comunicación por DSS con TS@.

Rev.	030
Fecha	30-03-2016
Descripción	Se añade la nueva funcionalidad de obtener información sobre los datos originalmente firmados en una firma PAdES (Baseline o no), CAdES (Baseline o no) y ASiC-s Baseline.

Rev.	031
Fecha	14-04-2016
Descripción	Se elmina obligatoriedad de incluir applicationOID en las peticiones de sello de tiempo.

Rev.	032
Fecha	18-04-2016
Descripción	Se modifica la gestión de ficheros de propiedades. Se modifican los anexos de ejemplo y la configuración para reflejar el nuevo comportamiento de Integra.

Rev.	033
Fecha	28-04-2016
Descripción	Se añade la funcionalidad de generar multi-firmas PAdES (Baseline o no) y generar firmas y multi-firmas PAdES (Baseline o no) con rúbrica.

Rev.	034
Fecha	07-06-2016
Descripción	Se añade clase XmlSitnatureModeEnum.

Rev.	035
Fecha	23-06-2016
Descripción	Se añade la funcionalidad de generación de firmas ASiCS Baseline conteniendo una firma CAdES Baseline o una firma XAdES Baseline.

Rev.	036
Fecha	04-01-2017
Descripción	Se incluye el parámetro adicional xades.canonicalizationMethod para los métodos sign, cosign y countersign de la interfaz Signer. Se añade información obviada para los casos de generación de firmas ASiC-S Baseline. Se describe toda la funcionalidad, así como las clases y métodos relacionados con el proceso de validación de firmas. Se actualizan los ejemplos relacionados con la validación de firmas. Se añade información obviada acerca de la multi-firma PAdES (Baseline o no).

Rev.	037
Fecha	20-04-2017
Descripción	Se indica la necesidad de usar la JRE 1.8 de Java. Se modifica el uso de la fuente Frutiger-Light por Calibri.

Rev.	038
Fecha	16-10-2017
Descripción	Se actualiza la versión a la 2.2.1_000.

Rev.	039
Fecha	18-11-2019
Descripción	Se actualizan las clases OptionalParameters y UpgradeSignatureRequest para añadir los nuevos elementos ReturnNextUpdate y ProcessAsNotBaseline en la validación y actualización de firma.

Rev.	040
Fecha	09-03-2020
Descripción	Se eliminan las referencias a la librería XMLSec 1.5, se cambia la librería AXIS 1.4 por la versión 1.7.9 (AXIS 2), se incluye la opción de generar firmas CAdES locales mediante el resumen hash del documento y se añaden nuevas propiedades asociadas a la configuración de TS@.

Rev.	041
Fecha	11-03-2020
Descripción	Se corrige pequeña incongruencia entre la documentación y el código con el nombre de la propiedad "trustedstorePath".

Rev.	042
Fecha	13-04-2020
Descripción	Se añade la clase *UtilsSignatureOp* al conjunto de clases definidas para incluir la nueva funcionalidad de Integr@: calcular la fecha de expiración de las firmas.

Rev.	043
Fecha	16-09-2020
Descripción	Se añade el nuevo tipo de integración 7 para la Generación de Informes de Firma..

Rev.	044
Fecha	20-11-2020
Descripción	Se añade el nuevo tipo de integración 7 para la Generación de Informes de Firma.

Rev.	045
Fecha	23-12-2020
Descripción	Se actualiza la versión a la 2.2.2_000

Rev.	046
Fecha	07-02-2022
Descripción	Se corrigen errores detectados en los ejemplos que se presentan en el punto A.6.2. Fachada de Integr@: Generación, validación y actualización de firma, co-firma y contrafirma, en la llamada al método "generateSignature" de IntegraFacade.

Rev.	047
Fecha	22-02-2022
Descripción	Se actualiza la versión a la 2.2.2_001

Rev.	048
Fecha	14-04-2022
Descripción	Se actualiza la versión a la 2.2.2_002. En esta versión, se evoluciona la librería de logging hacia log4j2. Se especifican las dependencias que sustituyen a la anterior versión, así como la instrucción necesaria para referencia al nuevo archivo de propiedades de log4j2 "integra-log4j2.xml". Por último se eliminan las dependencias con sl4fj para evitar conflicto con la nueva versión de log4j tras constatar que no es necesaria.

Rev.	049
Fecha	17-05-2022
Descripción	Se incorporan los cambios realizados para permitir el algoritmo de canonización de las firmas XAdES y la adecuación al estándar 5035.

Rev.	050
Fecha	26-09-2022
Descripción	Se actualiza la versión a la 2.2.3_000.

Rev.	051
Fecha	06-10-2022
Descripción	Se corrigen las dependencias y se cambia la versión a la 2.2.3_001.

Rev.	052
Fecha	14-10-2022
Descripción	Se corrigen las propiedades de integración con los servicios web de TS@. Se elimina la propiedad "com.serviceWSDLPath" y se completa con las propiedades "secureMode", "endPoint" y "servicePath" el fichero tsaXXXXX.properties .

Rev.	053

Fecha	18-10-2022
Descripción	Se actualiza la versión de la dependencia commons-text a la versión 1.10.0.

CONTROL DE DISTRIBUCIÓN

Documento nº:	@Firma-Integr@-MAN
Revisión:	053
Fecha:	18-10-2022

Copias Electrónicas:

La distribución de este documento ha sido controlada a través del sistema de información.

Copias en Papel:

La vigencia de las copias impresas en papel está condicionada a la coincidencia de su estado de revisión con el que aparece en el sistema electrónico de distribución de documentos.

El control de distribución de copias en papel para su uso en proyectos u otras aplicaciones es responsabilidad de los usuarios del sistema electrónico de información.

Fecha de impresión

Distribución en Papel:

Nombre o Cargo y Organización	Nº de Ejemplares	Referencia de la carta de transmisión y fecha

Índice

1 Objeto

En el presente documento describe cómo configurar y usar Integr@, una API para la integración con servicios de las plataformas @Firma, TS@ y eVisor, con funcionalidades de firma básica (generación, validación y actualización), sello de tiempo (generación, validación y re-sellado) y generación de informes de firma.

2 Alcance

El presente documento contempla los siguientes objetivos:

- Especificar los requisitos de software para el uso de Integr@.
- Describir la configuración e integración de los componentes que componen la API.
- Describir el funcionamiento de cada componente.
- Incluir ejemplos y casos de uso de Integr@.

3 Siglas y Acrónimos

MINECO	Ministerio de Asuntos Económicos y Transformación Digital
API	Application Programming Interface
DSS	Digital Signature Services
WS	Web Services
PDF	Portable Document Format
XML	eXtensible Markup Language
OCSP	Online Certificate Status Protocol
ASN.1	Abstract Syntax Notation One
HSM	Hardware Security Module
JRE	Java Runtime Environment
PKCS	Public-Key Cryptography Standards
SOAP	Simple Object Access Protocol
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
OID	Object Identifier
URN	Uniform Resource Name
URL	Uniform Resource Locator
SAML	Security Assertion Markup Language
JKS	Java KeyStore
JCEKS	Java Cryptography Extension KeyStore
SSL	Secure Sockets Layer
RSA	Rivest, Shamir y Adleman
TCP	Transmission Control Protocol
UTC	Coordinated Universal Time
XSLT	Extensible Stylesheet Language Transformations
XSL-FO	XSL Formatting Objects
XSD	XML Schema Definition
AGE	Administración General del Estado
RFC	Request For Comments
XPath	XML Path Language
SHA	Secure Hash Algorithm
RIPEMD	RACE Integrity Primitives Evaluation Message Digest
CMS	Cryptographic Message Syntax
CAdES	CMS Advanced Electronic Signatures
XAdES	XML Advanced Electronic Signatures
PAdES	PDF Advanced Electronic Signatures

CAdES-BES	CAdES Basic Electronic Signature
CAdES-EPES	CAdES Explicit Policy Electronic Signature
CAdES-T	CAdES Timestamp
CAdES-C	CAdES Complete
CAdES-X	CAdES Extended
CAdES-XL	CAdES Extended Long-Term
CAdES-A	CAdES Archive
XAdES-BES	XAdES Basic Electronic Signature
XAdES-EPES	XAdES Explicit Policy Electronic Signature
XAdES-T	XAdES Timestamp
XAdES-C	XAdES Complete
XAdES-X	XAdES Extended
XAdES-XL	XAdES Extended Long-Term
XAdES-A	XAdES Archive
PAdES-Basic	PAdES Basic
PAdES-BES	PAdES Basic Electronic Signature
PAdES-EPES	PAdES Explicit Policy Electronic Signature
PAdES-LTV	PAdES Long Term Validation
UUID	Identificador Único
CAdES B-Level	CAdES Basic Level
CAdES T-Level	CAdES Trusted Time for Signature Existence Level
CAdES LT-Level	CAdES Long Term Level
CAdES LTA-Level	CAdES Long Term with Archive Time-stamps Level
XAdES B-Level	XAdES Basic Level
XAdES T-Level	XAdES Trusted Time for Signature Existence Level
XAdES LT-Level	XAdES Long Term Level
XAdES LTA-Level	XAdES Long Term with Archive Time-stamps Level
PAdES B-Level	PAdES Basic Level
PAdES T-Level	PAdES Trusted Time for Signature Existence Level
PAdES LT-Level	PAdES Long Term Level
PAdES LTA-Level	PAdES Long Term with Archive Time-stamps Level
ASiC	Associated Signature Containers
ASiC-S	Simple Associated Signature Containers
ENI	Esquema Nacional de Interoperabilidad
TSL	Trusted Service Status List

4 Glosario de Términos y Definiciones

TS@	Plataforma de Sellado de Tiempo
@Firma	Plataforma de Validación y Firma Electrónica
eVisor	Plataforma de Generación y Validación de Informes de Firma
JBoss	Servidor de Aplicaciones Java de Código Abierto
Log4j	Librería Java de Registro de Mensajes

5 Referencias

[XMLSOAP_MAN_AFIRMA]	Manual de Programación de Web Services de @Firma
[XMLSOAP_DSS_AFIRMA]	Perfiles para la adaptación de los Web Services de @Firma 5 al protocolo OASIS-DSS
[XMLSOAP_DSS_EVISOR]	Manual de Programación de Web Services de eVisor
[XMLSOAP_DSS_TSA]	Manual de Servicios de TS@
[XML_TIMESTAMP_OASIS]	XML Timestamping Profile of the OASIS Digital Signature Services Version 1.0
[PDF_1.7]	Document management — Portable document format — Part 1: PDF 1.7
[RFC_3161]	Internet X.509 Public Key Infrastructure Time-Stamp Protocol (TSP)
[RFC_5019]	The Lightweight Online Certificate Status Protocol (OCSP) Profile for High-Volume Environments
[RFC_2560]	X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP
[XAdES_Baseline]	ETSI TS 103 171 V2.1.1 (2012-03)
[CAdES_Baseline]	ETSI TS 103 172 V2.2.2 (2013-04)
[PAdES_Baseline]	ETSI TS 103 173 V2.2.1 (2013-04)
[ASiC_Baseline]	ETSI TS 103 174 V2.2.1 (2013-06)
[PDF_Reference]	PDF Reference sixth edition - Adobe® Portable Document Format - Version 1.7 - November 2006
[@Firma-eVisor-Progs-Templates-MAN]	Manual de Programación de Plantillas para la Generación de Informes
[@Firma-eVisorTemplates-USER-MAN]	Manual de Usuario de eVisorTemplates

6 Introducción

Integr@ es un conjunto de librerías compuestas por clases java, ficheros de configuración y plantillas XML que facilitan la integración de una aplicación con los servicios WS de @Firma, los servicios WS de TS@, el servicio RFC 3161 de TS@, el servicio OCSP de validación de certificados de @Firma, servicios OCSP de validación de certificados ajenos a @Firma y API para la generación de informes de firma. La solución integral está dividida en los siguientes componentes, los cuales, podrán ser combinados, para utilizar a demanda y minimizando las librerías de terceros, solamente las funcionalidades que se desean integrar:

- **Integra-commons-2.2.3_001.jar**: Librería con elementos comunes al resto de módulos y que es de uso obligatorio para cualquier funcionalidad que se desee integrar.

- **Integra-commons-pdf-bc-2.2.3_001.jar**: Librería con elementos comunes al conjunto de módulos que necesitan funcionalidades relativas a funciones criptográficas, al procesamiento de ficheros PDF, y en definitiva a aquellas funcionalidades que no necesiten el procesamiento de estructuras XML.

- **Integra-commons-xml-2.2.3_001.jar**: Librería con elementos comunes al conjunto de módulos que necesitan funcionalidades relativas al procesamiento de estructuras XML.

- **Integra-ocsp-rfc3161-2.2.3_001.jar**: Librería para la conexión OCSP con @firma (para realizar validación de certificados) así como para la comunicación RFC 3161 con TS@ (para realizar solicitudes de sello de tiempo). Para el correcto funcionamiento de la librería es necesario incluir en la integración las librerías **Integra-commons-2.2.3_001.jar** e **Integra-commons-pdf-bc-2.2.3_001.jar**.

- **Integra-ws-2.2.3_001.jar**: Librería para la conexión DSS y WS tanto para @firma (solicitar operaciones de firma) como para TS@ (solicitar sellos de tiempo mediante DSS) y eVisor (en solicitudes al WS). Para el correcto funcionamiento de la librería es necesario incluir en la integración las librerías **Integra-commons-2.2.3_001.jar** e **Integra-commons-xml-2.2.3_001.jar**.

- **Integra-sign-operations-2.2.3_001.jar**: Librería para la generación, validación y actualización de firmas, co-firmas y contra-firmas CAdES (Baseline o no), PAdES (Baseline o no), XAdES (Baseline o no) y ASiC-S Baseline, propiamente desde Integr@. Para el correcto funcionamiento de la librería es necesario incluir en la integración las librerías **Integra-commons-2.2.3_001.jar**, **Integra-commons-pdf-bc-2.2.3_001.jar e Integra-ocsp-rfc3161-2.2.3_001.jar** (esta última para realizar validación OCSP de certificados si se han configurado este tipo de validaciones o para solicitudes de sello de tiempo mediante RFC 3161 a TS@ para ciertos formatos de firma), así como **Integra-commons-xml-2.2.3_001.jar** e **Integra-ws-2.2.3_001.jar** si se han configurado solicitudes de sello de tiempo mediante DSS a TS@ o se desean realizar firmas XAdES (ya que requieren procesamiento de estructuras XML).

- **Integra-utils-2.2.3_001.jar**: Librería horizontal de utilidades y HSM. Se incluirá siempre en cualquiera de las integraciones ya que contiene las constantes y funcionalidades necesarias para el completo uso de las funcionalidades del resto de módulos.

- **Integra-encryption-2.2.3_001.jar**: Librería para el cifrado y descifrados de datos. El cifrado puede ser simétrico utilizando los algoritmos AES, Camellia, 3DES, DES y Blowfish, o asimétrico utilizando los algoritmos RSA-OAEP y RSA-PKCS#1. Para el correcto funcionamiento de la librería es necesario incluir en la integración la librería **Integra-commons-2.2.3_001.jar**.

- **Integra-signature-report-2.2.3_001.jar:** Librería para la generación de informes de firma. Mediante el uso de transformaciones XSL-FO se podrán obtener informes de firma en formato PDF a partir de un documento original firmado, una plantilla XSLT y una serie de parámetros para especificar otros elementos a incluir. Esta librería es completamiento independiente del resto de librrías de Integr@.

- **Integra-tsl-2.2.3_001.jar**: Librería para la validación de certificados a partir de TSL. Mediante este módulo se podrá realizar la validación de un certificado pasado cómo parámetro frente a una TSL, que podrá ser cargada localmente o descargada desde su ruta de publicación. Para el correcto funcionamiento de la librería es necesario incluir en la integración la librería **Integra-commons-2.2.3_001.jar**.

7 Tipos de Integración

Según el tipo de integración que se desee realizar serán necesarias unas librerías u otras. A su vez, dependiendo del tipo de integración que se desee realizar se dispondrá de unas funcionalidades u otras.

A continuación, se enumeran los distintos tipos contemplados.

7.1 Funcionalidades por tipo de integración

A continuación, se muestra una tabla con las funcionalidades que realiza cada tipo de integración propuesto. Hay que tener en cuenta que en función de las configuraciones que se realicen serán necesarios ciertos tipos de integración u otros, es decir, si por ejemplo se desea realizar una firma CAdES propia de integr@ con sello de tiempo obtenido mediante el servicio DSS de TS@ se necesitará el tipo de integración 5 ya que el 4 sólo puede realizar sellos de tiempo obtenidos mediante RFC 3161.

Tenemos los siguientes tipos de integración:

- Integración 1: Acceso a servicios OCSP y RFC 3161
- Integración 2: Acceso a servicios WS y DSS
- Integración 3: Acceso a servicios WS, DSS, OCSP y RFC 3161
- Integración 4: Procesado de firmas CAdES (Baseline o no) y PAdES (Baseline o no)
- Integración 5: Procesado de firmas XAdES (Baseline o no) y ASiC-S Baseline, además de CAdES (Baseline o no) y PAdES (Baseline o no)
- Integración 6: Cifrado y descifrado de datos.
- Integración 7: Generación de informes de firma.
- Integración 8: Validación de Certificados mediante TSL.

	Integración 1	Integración 2	Integración 3	Integración 4	Integración 5	Integración 6	Integración 7	Integración 8
Uso de servicios WS y DSS de @Firma (operaciones de firma sobre la plataforma)	NO	SI	SI	NO	SI	NO	NO	NO
Uso de	NO	SI	SI	NO	SI	NO	NO	NO

servicios DSS de TS@ (sellado de tiempo)								
Uso de servicios WS de eVisor	NO	SI	SI	NO	SI	NO	NO	NO
Validación de certificados por OCSP	SI	NO	SI	SI	SI	NO	NO	NO
Uso de servicios RFC 3161 de TS@ (sellado de tiempo)	SI	NO	SI	SI	SI	NO	NO	NO
Procesado de firmas CAdES, Baseline o no, (generación, actualización y validación de firmas, cofirmas y contrafirmas)	NO	NO	NO	SI	SI	NO	NO	NO
Procesado de firmas PAdES, Baseline o no, (generación de firmas, actualización y validación de firmas, cofirmas y contrafirmas)	NO	NO	NO	SI	SI	NO	NO	NO
Procesado de firmas XAdES, Baseline o no, (generación, actualización y validación de firmas, cofirmas y contrafirmas) cofirmas y contrafirmas)	NO	NO	NO	NO	SI	NO	NO	NO
Procesado de firmas ASiC-S	NO	NO	NO	NO	SI	NO	NO	NO

que contengan una firma CAdES Baseline o XAdES Baseline (generación, validación y actualización)								
Cifrado y descifrado de datos	NO	NO	NO	NO	NO	SI	NO	NO
Generación de Informes de Firma	NO	NO	NO	NO	NO	NO	SI	NO
Validación de Certificados mediante TSL	NO	NO	NO	NO	NO	NO	NO	SI

7.2 Tipo de integración 1. Acceso a servicios OCSP y RFC 3161

Este tipo de integración permite:

+ Validar certificados contra un servidor OCSP.

+ Realizar peticiones a los servicios RFC 3161 de TS@.

Son necesarios los siguientes componentes de Integr@:

- **Integra-commons-2.2.3_001.jar**
- **Integra-commons-pdf-bc-2.2.3_001.jar**
- **Integra-ocsp-rfc3161-2.2.3_001.jar**
- **Integra-utils-2.2.3_001.jar**

Presenta los siguientes requerimientos de software:

- JRE versión 1.8.
- Librerías java de terceros (con las versiones que son compatibles):

Librería	Versión

bcmail-jdk16	1.46
bcprov-jdk16	1.46
bctsp-jdk16	1.46
commons-codec	1.15
commons-logging	1.2
itext	2.2
jakarta.activation-api	1.2.2
jakarta.xml.bind-api	2.3.3
log4j-api	2.17.1
log4j-core	2.17.1
log4j-jcl	2.17.1
slf4j-api	1.7.32
stax2-api	4.2.1
woodstox-core	6.2.6
xmlsec	2.3.0
xmltooling	1.3.2-1

7.3 Tipo de integración 2. Acceso a servicios WS y DSS

Este tipo de integración permite:

+ Realizar peticiones a los servicios web nativos de @firma.
+ Realizar peticiones a los servicios web DSS de @firma.
+ Realizar peticiones a los servicios web de eVisor.
+ Realizar peticiones a los servicios web DSS de TS@.

Son necesarios los siguientes componentes de Integr@:

- **Integra-commons-2.2.3_001.jar**
- **Integra-commons-xml-2.2.3_001.jar**
- **Integra-ws-2.2.3_001.jar**
- **Integra-utils-2.2.3_001.jar**

Presenta los siguientes requerimientos de software:

- JRE versión 1.8.
- Librerías java de terceros (con las versiones que son compatibles):

Librería	Versión
activation	1.0.2
axis2-adb	1.7.9
axis2-kernel	1.7.9
axis2-saaj	1.7.9
axis2-transport-http	1.7.9
axis2-transport-local	1.7.9
bcmail-jdk16	1.46
bcprov-jdk16	1.46
bctsp-jdk16	1.46
commons-codec	1.15
commons-lang3	3.3.2
commons-logging	1.2
jakarta.activation-api	1.2.2
jakarta.xml.bind-api	2.3.3
jaxb-api	2.0
jaxb-impl	2.2.4

jaxws-api	2.1
joda-time	1.6.2
jsr173_api	1.0
log4j-api	2.17.1
log4j-core	2.17.1
log4j-jcl	2.17.1
mail	1.4
opensaml	2.5.1-1
openws	1.4.2-1
saaj-api	1.3
serializer	2.7.2
slf4j-api	1.7.32
stax-api	1.0
stax2-api	4.2.1
woodstox-core	6.2.6
wss4j	1.6.6
xalan	2.7.2
xercesImpl	2.10.0
xml-apis	1.4.01
xmlsec	2.3.0
xmltooling	1.3.2-1
xws-security	3.0

7.4 Tipo de integración 3. Acceso a servicios WS, DSS, OCSP y RFC 3161

Este tipo de integración permite:

+ Validar certificados contra un servidor OCSP.
+ Realizar peticiones a los servicios RFC 3161 de TS@.
+ Realizar peticiones a los servicios web nativos de @firma.
+ Realizar peticiones a los servicios web DSS de @firma.
+ Realizar peticiones a los servicios web de eVisor.
+ Realizar peticiones a los servicios web DSS de TS@.

Son necesarios los siguientes componentes de Integr@:

- **Integra-commons-2.2.3_001.jar**
- **Integra-commons-xml-2.2.3_001.jar**
- **Integra-ws-2.2.3_001.jar**
- **Integra-utils-2.2.3_001.jar**
- **Integra-commons-pdf-bc-2.2.3_001.jar**
- **Integra-ocsp-rfc3161-2.2.3_001.jar**

Presenta los siguientes requerimientos de software:

- JRE versión 1.8.
- Librerías java de terceros (con las versiones que son compatibles):

Librería	Versión
activation	1.0.2
axis2-adb	1.7.9
axis2-kernel	1.7.9
axis2-saaj	1.7.9

axis2-transport-http	1.7.9
axis2-transport-local	1.7.9
bcmail-jdk16	1.46
bcprov-jdk16	1.46
bctsp-jdk16	1.46
commons-codec	1.15
commons-lang3	3.3.2
commons-logging	1.2
itext	2.2
jakarta.activation-api	1.2.2
jakarta.xml.bind-api	2.3.3
jaxb-api	2.0
jaxb-impl	2.2.4
jaxws-api	2.1
joda-time	1.6.2
jsr173_api	1.0
log4j-api	2.17.1
log4j-core	2.17.1
log4j-jcl	2.17.1
mail	1.4
opensaml	2.5.1-1
openws	1.4.2-1
saaj-api	1.3
serializer	2.7.2
slf4j-api	1.7.32

stax-api	1.0
stax2-api	4.2.1
woodstox-core	6.2.6
wss4j	1.6.6
xalan	2.7.2
xercesImpl	2.10.0
xml-apis	1.4.01
xmlsec	2.3.0
xmltooling	1.3.2-1
xws-security	3.0

7.5 Tipo de integración 4. Procesado de firmas CAdES (Baseline o no) y PAdES (Baseline o no)

Este tipo de integración permite:

+ Generar firmas CAdES-BES.
+ Generar co-firmas CAdES-BES.
+ Generar contra-firmas CAdES-BES.
+ Generar firmas CAdES-EPES.
+ Generar co-firmas CAdES-EPES.
+ Generar contra-firmas CAdES-EPES.
+ Generar firmas CAdES-T, con y sin política de firma, obteniendo el sello de tiempo de TS@ a través de alguno de sus servicios RFC 3161, validando, o no, el certificado del sello de tiempo obtenido a través de un servidor OCSP.
+ Generar co-firmas CAdES-T, con y sin política de firma, obteniendo el sello de tiempo de TS@ a través de alguno de sus servicios RFC 3161, validando, o no, el certificado del sello de tiempo obtenido a través de un servidor OCSP.

+ Generar contra-firmas CAdES-T, con y sin política de firma, obteniendo el sello de tiempo de TS@ a través de alguno de sus servicios RFC 3161, validando, o no, el certificado del sello de tiempo obtenido a través de un servidor OCSP.

+ Validar los firmantes contenidos en una firma CAdES (Baseline o no), validando, o no, el certificado firmante y el certificado de los sellos de tiempo, que contuviera en atributos signature-time-stamp, contra un servidor OCSP.

+ Actualizar los firmantes contenidos en una firma CAdES (Baseline o no) añadiéndoles un sello de tiempo en un atributo signature-time-stamp (siempre que dicho firmante no posea ya un sello de tiempo en dicho atributo previamente), obteniendo el sello de tiempo de TS@ a través de alguno de sus servicios RFC 3161, validando, o no, cada certificado del sello de tiempo obtenido y cada certificado firmante a través de un servidor OCSP.

+ Generar firmas PAdES-BES que contengan un núcleo de firma CAdES-BES o CAdES-T (sin política de firma), obteniendo el sello de tiempo de TS@ a través de alguno de sus servicios RFC 3161, validando, o no, el certificado del sello de tiempo obtenido a través de un servidor OCSP.

+ Generar firmas PAdES-EPES que contengan un núcleo de firma CAdES-EPES o CAdES-T (con política de firma), obteniendo el sello de tiempo de TS@ a través de alguno de sus servicios RFC 3161, validando, o no, el certificado del sello de tiempo obtenido a través de un servidor OCSP.

+ Validar un documento PDF que contenga una o varias firmas PDF, PAdES-Basic, PAdES-BES, PAdES-EPES o PAdES-LTV, así como el certificado firmante y el certificado de los sellos de tiempo, que contuviera en atributos signature-time-stamp o diccionarios de sello de tiempo, contra un servidor OCSP.

+ Añadir un diccionario de sello de tiempo (estructura PAdES-LTV) a un documento PDF, obteniendo el sello de tiempo de TS@ a través de alguno de sus servicios RFC 3161, validando, o no, el certificado del sello de tiempo obtenido a través de un servidor OCSP.

+ Generar firmas CAdES B-Level, con y sin política de firma.

+ Generar co-firmas CAdES B-Level, con y sin política de firma.

+ Generar contra-firmas CAdES B-Level, con y sin política de firma.

+ Generar firmas CAdES T-Level, con y sin política de firma, obteniendo el sello de tiempo de TS@ a través de alguno de sus servicios RFC 3161, validando, o no, el certificado del sello de tiempo obtenido a través de un servidor OCSP.

+ Generar co-firmas CAdES T-Level, con y sin política de firma, obteniendo el sello de tiempo de TS@ a través de alguno de sus servicios RFC 3161, validando, o no, el certificado del sello de tiempo obtenido a través de un servidor OCSP.

+ Generar contra-firmas CAdES T-Level, con y sin política de firma, obteniendo el sello de tiempo de TS@ a través de alguno de sus servicios RFC 3161, validando, o no, el certificado del sello de tiempo obtenido a través de un servidor OCSP.

+ Generar firmas PAdES B-Level que contengan un núcleo de firma CAdES sin política de firma.

+ Generar firmas PAdES B-Level que contengan un núcleo de firma CAdES con política de firma.

+ Generar firmas PAdES T-Level contengan un núcleo de firma CAdES con sello de tiempo contenido en un atributo signature-time-stamp, con o sin política de firma, obteniendo el sello de tiempo de TS@ a través de alguno de sus servicios RFC 3161, validando, o no, el certificado del sello de tiempo obtenido a través de un servidor OCSP.

+ Validar un documento PDF que contenga una o varias firmas PDF, PAdES B-Level, PAdES T-Level, PAdES LT-Level o PAdES LTa-Level, así como el certificado firmante y el certificado de los sellos de tiempo, que contuviera en atributos signature-time-stamp o diccionarios de sello de tiempo, contra un servidor OCSP.

+ Añadir un diccionario de sello de tiempo (estructura PAdES T-Level) a un documento PDF, obteniendo el sello de tiempo de TS@ a través de alguno de sus servicios RFC 3161, validando, o no, el certificado del sello de tiempo obtenido a través de un servidor OCSP

+ Validar certificados contra un servidor OCSP.

+ Realizar peticiones a los servicios RFC 3161 de TS@.

+ Obtener información acerca de los datos originalmente firmados de una firma CAdES (Baseline o no).

+ Obtener información acerca de los datos originalmente firmados de una firma PAdES (Baseline o no).

+ Generar co-firmas PAdES (Baseline o no).

+ Generar contra-firmas PAdES (Baseline o no).

+ Generar firmas PAdES (Baseline o no) con rúbrica.

+ Generar co-firmas PAdES (Baseline o no) con rúbrica.

+ Generar contra-firmas PAdES (Baseline o no) con rúbrica.

Son necesarios los siguientes componentes de Integr@:

- **Integra-commons-2.2.3_001.jar**

- **Integra-commons-pdf-bc-2.2.3_001.jar**
- **Integra-ocsp-rfc3161-2.2.3_001.jar**
- **Integra-utils-2.2.3_001.jar**
- **Integra-sign-operations-2.2.3_001.jar**

Presenta los siguientes requerimientos de software:

- JRE versión 1.8.
- Librerías java de terceros (con las versiones que son compatibles):

Librería	Versión
bcmail-jdk16	1.46
bcprov-jdk16	1.46
bctsp-jdk16	1.46
commons-codec	1.15
commons-io	2.4
commons-logging	1.2
itext	2.2
jakarta.activation-api	1.2.2
jakarta.xml.bind-api	2.3.3
log4j-api	2.17.1
log4j-core	2.17.1
log4j-jcl	2.17.1
slf4j-api	1.7.32
stax2-api	4.2.1
tika-core	1.2
woodstox-core	6.2.6

xmlsec	2.3.0
xmltooling	1.3.2-1

7.6 Tipo de integración 5. Procesado de firmas XAdES (Baseline o no) y ASiC-S Baseline, además de CAdES (Baseline o no) y PAdES (Baseline o no)

Este tipo de integración permite:

+ Realizar peticiones a los servicios web nativos de @firma.
+ Realizar peticiones a los servicios web DSS de @firma.
+ Realizar peticiones a los servicios web de eVisor.
+ Realizar peticiones a los servicios web DSS de TS@.
+ Generar firmas CAdES-BES.
+ Generar co-firmas CAdES-BES.
+ Generar contra-firmas CAdES-BES.
+ Generar firmas CAdES-EPES.
+ Generar co-firmas CAdES-EPES.
+ Generar contra-firmas CAdES-EPES.
+ Generar firmas CAdES-T, con y sin política de firma, obteniendo el sello de tiempo de TS@ a través de alguno de sus servicios RFC 3161, validando, o no, el certificado del sello de tiempo obtenido a través de un servidor OCSP.
+ Generar co-firmas CAdES-T, con y sin política de firma, obteniendo el sello de tiempo de TS@ a través de alguno de sus servicios RFC 3161, validando, o no, el certificado del sello de tiempo obtenido a través de un servidor OCSP.
+ Generar contra-firmas CAdES-T, con y sin política de firma, obteniendo el sello de tiempo de TS@ a través de alguno de sus servicios RFC 3161, validando, o no, el certificado del sello de tiempo obtenido a través de un servidor OCSP.
+ Validar los firmantes contenidos en una firma CAdES (Baseline o no), validando, o no, el certificado firmante y el certificado de los sellos de tiempo, que contuviera en atributos signature-time-stamp, contra un servidor OCSP.
+ Actualizar los firmantes contenidos en una firma CAdES (Baseline o no) añadiéndoles un sello de tiempo en un atributo signature-time-stamp (siempre que dicho firmante no posea ya

un sello de tiempo en dicho atributo previamente), obteniendo el sello de tiempo de TS@ a través de alguno de sus servicios RFC 3161, validando, o no, cada certificado del sello de tiempo obtenido y cada certificado firmante a través de un servidor OCSP.

+ Generar firmas PAdES-BES que contengan un núcleo de firma CAdES-BES o CAdES-T (sin política de firma), obteniendo el sello de tiempo de TS@ a través de alguno de sus servicios RFC 3161, validando, o no, el certificado del sello de tiempo obtenido a través de un servidor OCSP.

+ Generar firmas PAdES-EPES que contengan un núcleo de firma CAdES-EPES o CAdES-T (con política de firma), obteniendo el sello de tiempo de TS@ a través de alguno de sus servicios RFC 3161, validando, o no, el certificado del sello de tiempo obtenido a través de un servidor OCSP.

+ Validar un documento PDF que contenga una o varias firmas PDF, PAdES-Basic, PAdES-BES, PAdES-EPES o PAdES-LTV, así como el certificado firmante y el certificado de los sellos de tiempo, que contuviera en atributos signature-time-stamp o diccionarios de sello de tiempo, contra un servidor OCSP.

+ Añadir un diccionario de sello de tiempo (estructura PAdES-LTV) a un documento PDF, obteniendo el sello de tiempo de TS@ a través de alguno de sus servicios RFC 3161, validando, o no, el certificado del sello de tiempo obtenido a través de un servidor OCSP.

+ Generar firmas CAdES B-Level, con y sin política de firma.

+ Generar co-firmas CAdES B-Level, con y sin política de firma.

+ Generar contra-firmas CAdES B-Level, con y sin política de firma.

+ Generar firmas CAdES T-Level, con y sin política de firma, obteniendo el sello de tiempo de TS@ a través de alguno de sus servicios RFC 3161, validando, o no, el certificado del sello de tiempo obtenido a través de un servidor OCSP.

+ Generar co-firmas CAdES T-Level, con y sin política de firma, obteniendo el sello de tiempo de TS@ a través de alguno de sus servicios RFC 3161, validando, o no, el certificado del sello de tiempo obtenido a través de un servidor OCSP.

+ Generar contra-firmas CAdES T-Level, con y sin política de firma, obteniendo el sello de tiempo de TS@ a través de alguno de sus servicios RFC 3161, validando, o no, el certificado del sello de tiempo obtenido a través de un servidor OCSP.

+ Generar firmas PAdES B-Level que contengan un núcleo de firma CAdES sin política de firma.

+ Generar firmas PAdES B-Level que contengan un núcleo de firma CAdES con política de firma.

+ Generar firmas PAdES T-Level contengan un núcleo de firma CAdES con sello de tiempo contenido en un atributo signature-time-stamp, con o sin política de firma, obteniendo el sello de tiempo de TS@ a través de alguno de sus servicios RFC 3161, validando, o no, el certificado del sello de tiempo obtenido a través de un servidor OCSP.

+ Validar un documento PDF que contenga una o varias firmas PDF, PAdES B-Level, PAdES T-Level, PAdES LT-Level o PAdES LTa-Level, así como el certificado firmante y el certificado de los sellos de tiempo, que contuviera en atributos signature-time-stamp o diccionarios de sello de tiempo, contra un servidor OCSP.

+ Añadir un diccionario de sello de tiempo (estructura PAdES T-Level) a un documento PDF, obteniendo el sello de tiempo de TS@ a través de alguno de sus servicios RFC 3161, validando, o no, el certificado del sello de tiempo obtenido a través de un servidor OCSP.

+ Generar firmas XAdES-BES.

+ Generar co-firmas XAdES-BES.

+ Generar contra-firmas XAdES-BES.

+ Generar firmas XAdES-EPES.

+ Generar co-firmas XAdES-EPES.

+ Generar contra-firmas XAdES-EPES.

+ Generar firmas XAdES-T, con y sin política de firma, obteniendo el sello de tiempo de TS@ a través de alguno de sus servicios RFC 3161 o DSS, validando, o no, el certificado del sello de tiempo obtenido a través de un servidor OCSP.

+ Generar co-firmas XAdES-T, con y sin política de firma, obteniendo el sello de tiempo de TS@ a través de alguno de sus servicios RFC 3161 o DSS, validando, o no, el certificado del sello de tiempo obtenido a través de un servidor OCSP.

+ Generar contra-firmas XAdES-T, con y sin política de firma, obteniendo el sello de tiempo de TS@ a través de alguno de sus servicios RFC 3161 o DSS, validando, o no, el certificado del sello de tiempo obtenido a través de un servidor OCSP.

+ Validar los firmantes contenidos en un documento XML (Baseline o no), validando, o no, el certificado firmante y el certificado de los sellos de tiempo, que contuviera en elementos xades:SignatureTimeStamp, contra un servidor OCSP.

+ Actualizar los firmantes contenidos en una firma XAdES (Baseline o no) añadiéndoles un sello de tiempo en un elemento xades:SignatureTimeStamp (siempre que dicho firmante no posea ya un sello de tiempo en dicho atributo previamente), obteniendo el sello de tiempo de TS@ a través de alguno de sus servicios RFC 3161 o DSS, validando, o no, cada certificado de cada sello de tiempo obtenido y cada certificado firmante a través de un servidor OCSP.

+ Generar firmas XAdES B-Level, con y sin política de firma.

+ Generar co-firmas XAdES B-Level, con y sin política de firma.

+ Generar contra-firmas XAdES B-Level, con y sin política de firma.

+ Generar firmas XAdES T-Level, con y sin política de firma, obteniendo el sello de tiempo de TS@ a través de alguno de sus servicios RFC 3161 o DSS, validando, o no, el certificado del sello de tiempo obtenido a través de un servidor OCSP.

+ Generar co-firmas XAdES T-Level, con y sin política de firma, obteniendo el sello de tiempo de TS@ a través de alguno de sus servicios RFC 3161 o DSS, validando, o no, el certificado del sello de tiempo obtenido a través de un servidor OCSP.

+ Generar contra-firmas XAdES T-Level, con y sin política de firma, obteniendo el sello de tiempo de TS@ a través de alguno de sus servicios RFC 3161 o DSS, validando, o no, el certificado del sello de tiempo obtenido a través de un servidor OCSP.

+ Validar la estructura de firmas ASiC-S Baseline, así como los firmantes del documento XML firmado o de la firma CAdES Baseline que contenga.

+ Actualizar los firmantes del documento XML o de la firma CAdES Baseline contenido dentro de una firma ASiC-S, siguiendo el modo de actualización para firmas XAdES Baseline y CAdES Baseline comentados anteriormente.

+ Validar certificados contra un servidor OCSP.

+ Realizar peticiones a los servicios RFC 3161 de TS@.

+ Obtener información acerca de los datos originalmente firmados de una firma CAdES (Baseline o no).

+ Obtener información acerca de los datos originalmente firmados de una firma PAdES (Baseline o no).

+ Obtener información acerca de los datos originalmente firmados de una firma ASiC-S Baseline.

+ Generar firmas ASiC-S conteniendo una firma CAdES Baseline o una firma XAdES Baseline.

Son necesarios los siguientes componentes de Integr@:

- **Integra-commons-2.2.3_001.jar**
- **Integra-commons-pdf-bc-2.2.3_001.jar**
- **Integra-ocsp-rfc3161-2.2.3_001.jar**

- **Integra-sign-operations-2.2.3_001.jar**
- **Integra-commons-xml-2.2.3_001.jar**
- **Integra-ws-2.2.3_001.jar**
- **Integra-utils-2.2.3_001.jar**

Presenta los siguientes requerimientos de software:

- JRE versión 1.8.
- Librerías java de terceros (con las versiones que son compatibles):

Librería	Versión
activation	1.0.2
axis2-adb	1.7.9
axis2-kernel	1.7.9
axis2-saaj	1.7.9
axis2-transport-http	1.7.9
axis2-transport-local	1.7.9
bcmail-jdk16	1.46
bcprov-jdk16	1.46
bctsp-jdk16	1.46
commons-codec	1.15
commons-io	2.4
commons-lang3	3.3.2
commons-logging	1.2
itext	2.2
jakarta.activation-api	1.2.2
jakarta.xml.bind-api	2.3.3

jaxb-api	2.0
jaxb-impl	2.2.4
jaxws-api	2.1
joda-time	1.6.2
jsr173_api	1.0
log4j-api	2.17.1
log4j-core	2.17.1
log4j-jcl	2.17.1
mail	1.4
opensaml	2.5.1-1
openws	1.4.2-1
saaj-api	1.3
serializer	2.7.2
slf4j-api	1.7.32
stax-api	1.0
stax2-api	4.2.1
tika-core	1.2
woodstox-core	6.2.6
wss4j	1.6.6
xalan	2.7.2
xercesImpl	2.10.0
xml-apis	1.4.01
xmlsec	2.3.0
xmltooling	1.3.2-1
xws-security	3.0

7.7 Tipo de integración 6. Cifrado y descifrado de datos.

Este tipo de integración permite:

+ Cifrado y descifrado simétrico utilizando uno de los siguientes algoritmos: AES, Camellia, 3DES, DES y Blowfish.

+ Cifrado y descifrado asimétrico utilizando uno de los siguientes algoritmos: RSA-OAEP y RSA-PKCS#1.

Son necesarios los siguientes componentes de Integr@:

- **Integra-encryption-2.2.3_001.jar**
- **Integra-commons-2.2.3_001.jar**

Presenta los siguientes requerimientos de software:

- JRE versión 1.8.
- Librerías java de terceros (con las versiones que son compatibles):

Librería	Versión
bcprov-jdk16	1.46
commons-codec	1.15
commons-logging	1.2
jakarta.activation-api	1.2.2
jakarta.xml.bind-api	2.3.3
log4j-api	2.17.1
log4j-core	2.17.1
log4j-jcl	2.17.1
slf4j-api	1.7.32
stax2-api	4.2.1
woodstox-core	6.2.6

xmlsec	2.3.0
xmltooling	1.3.2-1

7.8 Tipo de integración 7. Generación de informes de firma.

Este tipo de integración permite:

+ Generación de informes de firma mediante proceso XSL-FO.

Son necesarios los siguientes componentes de Integr@:

- **Integra-signature-report-2.2.3_001.jar**

  No presenta dependencias con ningún otro componente de Integr@ ni ningún componente de Integr@ presenta dependencia con él.

Presenta los siguientes requerimientos de software:

- JRE versión 1.8.
- Librerías java de terceros (con las versiones que son compatibles):

Librería	Versión
avalon-framework-api	4.3.1
avalon-framework-impl	4.3.1
barcode4j-fop-ext-complete	2.1
batik-anim	1.9
batik-awt-util	1.9
batik-bridge	1.9
batik-constants	1.9
batik-css	1.9
batik-dom	1.9
batik-ext	1.9
batik-extension	1.9

batik-gvt	1.9
batik-i18n	1.9
batik-parser	1.9
batik-script	1.9
batik-svg-dom	1.9
batik-svggen	1.9
batik-transcoder	1.9
batik-util	1.9
batik-xml	1.9
bcmail-jdk16	1.46
bcprov-jdk16	1.46
bctsp-jdk16	1.46
commons-io	1.3.1
commons-lang3	3.3.2
commons-logging	1.1.1
commons-text	1.10.0
core	3.3.0
fontbox	2.0.4
fop	2.2
itext	2.1.7
jai-imageio-core	1.3.1
javase	3.3.0
jcommander	1.48
log4j-api	2.17.1
log4j-core	2.17.1

log4j-jcl	2.17.1
serializer	2.7.2
tika-core	1.24.1
xalan	2.7.2
xml-apis	1.3.04
xml-apis-ext	1.3.04
xmlgraphics-commons	2.2

7.9 Tipo de integración 8. Validación de Certificados mediante TSL.

Este tipo de integración permite:

+ Validación de Certificados mediante TSL.

Son necesarios los siguientes componentes de Integr@:

- **Integra-common-tsl-2.2.3_001.jar**
- **Integra-tsl-2.2.3_001.jar**

Presentan los siguientes requerimientos de software:

- JRE versión 1.8.
- Librerías java de terceros (con las versiones que son compatibles):

Librería	Versión
afirma-core	1.6.2
afirma-crypto-core-xml	1.6.2
afirma-crypto-validation	1.6.2
afirma-crypto-xades	1.6.2
afirmaSchemaXMLTSLv5	119612v020101

bcpkix-jdk15on	1.60
bcprov-jdk15on	1.60
commons-codec	1.15
commons-io	2.4
commons-logging	1.2.4
httpclient	4.5.6
httpcore	4.4.10
jcifs	1.3.14-kohsuke-1
jldap	20091007
jxades	0.2.0
log4j-api	2.17.1
log4j-core	2.17.1
log4j-jcl	2.17.1
XAdES	01903v132
xmlbeans-afirma	2.3.2000
XMLDSig	2000v09
xmlSchema	2001
xmltooling	1.3.2-1
xom	1.2.10

8 Configuración

Para usar las API de Integr@ se deben incluir en la variable de entorno CLASSPATH las librerías de Integr@ requeridas para el tipo de integración que se desea realizar, y todas las librerías de terceros requeridas y citadas en el punto 1 en cada una de los tipos de integración. Además, existen una serie de archivos de propiedades y carpetas de configuración que, en función del componente a utilizar, serán de uso obligatorio o no. Dichos elementos pueden ser incluidos en CLASSPATH o bien pueden

ser localizados en una ruta concreta que puede ser indicada como variable (-Dintegra.config) en las opciones de arranque de la máquina virtual de java como se indica a continuación:

-Dintegra.config=D:/configuracion

Los elementos citados se enumeran a continuación.

Estáticos: Elementos de configuración con nombre estático.

- Carpeta **transformersTemplates**.
- Carpeta **xml**.
- Archivo **hsm.properties**.
- Archivo **Language.properties**.
- Archivo **parserParameters.properties**.
- Archivo **transformers.properties**.
- Archivo **integra-log4j2.xml. Este archivo, debe ser referenciado con su propia variable de arranque: -Dlog4j2.configurationFile=file:/[ruta_config]/integra-log4j2.xml**

Dinámicos: Elementos de configuración cuyo nombre depende del integrador y deben estar mapeados para ser utilizados.

- Archivo **mappingFiles.properties.** Fichero donde se mapearán los nombres del resto.
- Archivo con configuraciones generales con nombre a establecer por el integrador. Se propone **integra.properties**.
- Archivo con configuraciones de acceso a aFirma con nombre a establecer por el integrador. Se propone **afirmaXXXXX.properties** (donde XXXXX será el nombre de la aplicación afirma a utilizar).
- Archivo con configuraciones de acceso a TSA con nombre a establecer por el integrador. Se propone **tsaXXXXX.properties** (donde XXXXX será el nombre de la aplicación tsa a utilizar).
- Archivo con configuraciones de acceso a eVisor con nombre a establecer por el integrador. Se propone **evisorXXXXX.properties** (donde XXXXX será el nombre de la aplicación evisor a utilizar).

A continuación se especificará que elementos son necesarios en función del tipo de integración que se desee realizar y se describirá cada uno de los elementos.

8.1 Elementos necesarios para el tipo de integración 1 "Acceso a servicios OCSP y RFC 3161"

Para este tipo de integración se necesitan los siguientes componentes:

- Archivo **hsm.properties** (sólo si se va a hacer uso de HSM).
- Archivo **Language.properties**.
- Archivo **integra-log4j2.xml**.
- Archivo **mappingFiles.properties.**
- Archivo con configuraciones de acceso a TSA con nombre a establecer por el integrador. Se propone **tsaXXXXX.properties,** donde XXXXX será el nombre de la aplicación tsa a utilizar (sólo si se van a realizar peticiones de generación de sello de tiempo a los servicios RFC 3161 de TS@).
- Archivo con configuraciones generales con nombre a establecer por el integrador. Se propone **integra.properties**.

8.2 Elementos necesarios para el tipo de integración 2 "Acceso a servicios WS y DSS"

Para este tipo de integración se necesitan los siguientes componentes:

- Carpeta **transformersTemplates**.
- Carpeta **xml**.
- Archivo **hsm.properties** (sólo si se va a hacer uso de HSM).
- Archivo **Language.properties**.
- Archivo **integra-log4j2.xml**.
- Archivo **parserParameters.properties**.
- Archivo **transformers.properties**.
- Archivo **mappingFiles.properties.**
- Archivo con configuraciones generales con nombre a establecer por el integrador. Se propone **integra.properties**
- Archivo con configuraciones de acceso a aFirma con nombre a establecer por el integrador. Se propone **afirmaXXXXX.properties,** donde XXXXX será el nombre de la

aplicación afirma a utilizar (sólo si se van a realizar peticiones a los servicios web nativos o DSS de @firma).

- Archivo con configuraciones de acceso a evisor con nombre a establecer por el integrador. Se propone **evisorXXXXX.properties,** donde XXXXX será el nombre de la aplicación afirma a utilizar (sólo si se van a realizar peticiones a los servicios de eVisor).

- con configuraciones de acceso a TSA con nombre a establecer por el integrador. Se propone **tsaXXXXX.properties,** donde XXXXX será el nombre de la aplicación tsa a utilizar (sólo si se van a realizar peticiones a los servicios web DSS de TS@).

8.3 Elementos necesarios para el tipo de integración 3 "Acceso a servicios WS, DSS, OCSP y RFC 3161"

Para este tipo de integración se necesitan los siguientes componentes:

- Archivo **hsm.properties** (sólo si se va a hacer uso de HSM).

- Archivo **Language.properties**.

- Archivo **integra-log4j2.xml**.

- Archivo **tsaServiceInvoker.properties** (sólo si se van a realizar peticiones a los servicios web DSS o RFC 3161 de TS@).

- Carpeta **transformersTemplates** (sólo si se van a realizar peticiones a los servicios web nativos o DSS de @firma, a los servicios web de eVisor, o bien a los servicios web DSS de TS@).

- Carpeta **xml** (sólo si se van a realizar peticiones a los servicios web nativos o DSS de @firma, a los servicios web de eVisor, o bien a los servicios web DSS de TS@).

- Archivo **parserParameters.properties** (sólo si se van a realizar peticiones a los servicios web nativos o DSS de @firma, a los servicios web de eVisor, o bien a los servicios web DSS de TS@).

- Archivo **transformers.properties** (sólo si se van a realizar peticiones a los servicios web nativos o DSS de @firma, a los servicios web de eVisor, o bien a los servicios web DSS de TS@).

- Archivo **mappingFiles.properties.**

- Archivo con configuraciones generales con nombre a establecer por el integrador. Se propone **integra.properties**

- Archivo con configuraciones de acceso a aFirma con nombre a establecer por el integrador. Se propone **afirmaXXXXX.properties,** donde XXXXX será el nombre de la aplicación afirma a utilizar (sólo si se van a realizar peticiones a los servicios web nativos o DSS de @firma).

- Archivo con configuraciones de acceso a evisor con nombre a establecer por el integrador. Se propone **evisorXXXXX.properties,** donde XXXXX será el nombre de la aplicación afirma a utilizar (sólo si se van a realizar peticiones a los servicios de eVisor).

- con configuraciones de acceso a TSA con nombre a establecer por el integrador. Se propone **tsaXXXXX.properties,** donde XXXXX será el nombre de la aplicación tsa a utilizar (sólo si se van a realizar peticiones a los servicios web DSS de TS@).

8.4 Elementos necesarios para el tipo de integración 4 "Procesado de firmas CAdES (Baseline o no) y PAdES (Baseline o no)"

Para este tipo de integración se necesitan los siguientes componentes:

- Archivo **hsm.properties** (sólo si se va a hacer uso de HSM).

- Archivo **Language.properties**.

- Archivo **integra-log4j2.xml**.

- Archivo **mappingFiles.properties.**

- Archivo con configuraciones generales con nombre a establecer por el integrador. Se propone **integra.properties**

8.5 Elementos necesarios para el tipo de integración 5 "Procesado de firmas XAdES (Baseline o no) y ASiC-S Baseline, además de CAdES (Baseline o no) y PAdES (Baseline o no)"

Para este tipo de integración se necesitan los siguientes componentes:

- Carpeta **transformersTemplates** (sólo si se van a realizar peticiones a los servicios web nativos o DSS de @firma, a los servicios web de eVisor, o bien a los servicios web DSS de TS@).

- Carpeta **xml** (sólo si se van a realizar peticiones a los servicios web nativos o DSS de @firma, a los servicios web de eVisor, o bien a los servicios web DSS de TS@).

- Archivo **hsm.properties** (sólo si se va a hacer uso de HSM).

- Archivo **Language.properties**.
- Archivo **integra-log4j2.xml**.
- Archivo **parserParameters.properties** (sólo si se van a realizar peticiones a los servicios web nativos o DSS de @firma, a los servicios web de eVisor, o bien a los servicios web DSS de TS@).
- Archivo **transformers.properties** (sólo si se van a realizar peticiones a los servicios web nativos o DSS de @firma, a los servicios web de eVisor, o bien a los servicios web DSS de TS@).
- Archivo **mappingFiles.properties.**
- Archivo con configuraciones generales con nombre a establecer por el integrador. Se propone **integra.properties**
- Archivo con configuraciones de acceso a aFirma con nombre a establecer por el integrador. Se propone **afirmaXXXXX.properties,** donde XXXXX será el nombre de la aplicación afirma a utilizar (sólo si se van a realizar peticiones a los servicios web nativos o DSS de @firma).
- Archivo con configuraciones de acceso a evisor con nombre a establecer por el integrador. Se propone **evisorXXXXX.properties,** donde XXXXX será el nombre de la aplicación afirma a utilizar (sólo si se van a realizar peticiones a los servicios de eVisor).
- con configuraciones de acceso a TSA con nombre a establecer por el integrador. Se propone **tsaXXXXX.properties,** donde XXXXX será el nombre de la aplicación tsa a utilizar (sólo si se van a realizar peticiones a los servicios web DSS de TS@).

8.6 Elementos necesarios para el tipo de integración 6 "Cifrado y descifrado de datos)"

Para este tipo de integración se necesitan los siguientes componentes:

- Archivo **Language.properties**.
- Archivo **integra-log4j2.xml**.

8.7 Elementos necesarios para el tipo de integración 7 "Generación de informes de firma"

Todos los elementos necesarios para este tipo de integración son provistos a través de parámetros de entrada al invocar el API.

8.8 Elementos necesarios para el tipo de integración 8 "Validación de Certificados mediante TSL"

Todos los elementos necesarios para este tipo de integración son provistos a través de parámetros de entrada al invocar el API.

8.9 Carpeta transformersTemplates

Esta carpeta incluye todas las plantillas asociadas al paquete *transformers*. La función de estas plantillas es definir la estructura final de las interfaces XML a generar y los parámetros a extraer de las respuestas XML en los procesos de comunicación con los servicios web de @Firma, eVisor y TS@ . La carpeta contiene a su vez 2 carpetas:

8.10 Carpeta parserTemplates

Esta carpeta incluye las plantillas asociadas a las respuestas XML obtenidas de los diferentes servicios de @Firma, TS@ y eVisor. Las plantillas que lo componen son:

- **DSSArchiveRetrievalResponse_V1.xml**: Plantilla para el procesado de respuestas asociadas al servicio de obtención de firmas registradas.
- **DSSAsyncResponseStatus_V1.xml**: Plantilla para el procesado de las respuestas asociadas a los servicios de consulta de peticiones asíncronas, de @Firma.
- **DSSBatchResponse_V1.xml**: Plantilla para el procesado de las respuestas asociadas a los servicios de validaciones en lote, de @Firma.
- **DSSCounterSignResponse_V1.xml**: Plantilla para el procesado de las respuestas asociadas al servicio firma servidor CounterSign, de @Firma.
- **DSSSignResponse_V1.xml**: Plantilla para el procesado de las respuestas asociadas a los servicios de firma delegada, de @Firma.
- **DSSTSAReTimestampResponse_V1.xml**: Plantilla para el procesado de las respuestas asociadas al servicio de renovación de sello de tiempo, de TS@.
- **DSSTSATimestampResponse_V1.xml**: Plantilla para el procesado de las respuestas asociadas al servicio de creación de sello de tiempo, de TS@.
- **DSSTSATimestampValidationResponse_V1.xml**: Plantilla para el procesado de las respuestas asociadas al servicio de validación de sello de tiempo, de TS@.

- **DSSVerifyCertificateResponse_V1.xml**: Plantilla para el procesado de las respuestas asociadas al servicio de validación de certificados, de @Firma.

- **DSSVerifyResponse_V1.xml**: Plantilla para el procesado de las respuestas asociadas al servicio de validación y actualización de firmas, de @Firma.

- **EVisor_GenerateReportResponse_V1.xml**: Plantilla para el procesado de las respuestas asociadas al servicio de generación de informes, de eVisor.

- **EVisor_ValidationReportResponse_V1.xml**: Plantilla para el procesado de las respuestas asociadas al servicio de validación de informes firmados, de eVisor.

8.11 Carpeta xmlTemplates

Esta carpeta incluye las plantillas asociadas a la construcción de las peticiones XML hacia los diferentes servicios de @Firma, TS@ y eVisor. Las plantillas que la componen son:

- **AlmacenarDocumento_V1.xml**: Plantilla para la construcción de las peticiones al servicio para almacenar documentos, de @Firma.

- **DSSAfirmaSignRequest_V1.xml**: Plantilla para la construcción de las peticiones a los servicios de firma delegada, de @Firma.

- **DSSArchiveRetrievalRequest_V1.xml**: Plantilla para la construcción de peticiones al servicio de obtención de firma registrada, de @Firma.

- **DSSAsyncRequestStatus_V1.xml**: Plantilla para la construcción de las peticiones al servicio de consulta de peticiones asíncronas, de @Firma.

- **DSSBatchRequest_V1.xml**: Plantilla para la construcción de las peticiones a los servicios de validaciones por lote, de @Firma.

- **DSSTSAReTimestampRequest_V1.xml**: Plantilla para la construcción de las peticiones al servicio de renovación de sello de tiempo, de TS@.

- **DSSTSATimestampRequest_V1.xml**: Plantilla para la construcción de las peticiones al servicio de generación de sello de tiempo, de TS@.

- **DSSTSATimestampValidationRequest_V1.xml**: Plantilla para la construcción de las peticiones al servicio de validación de sello de tiempo, de TS@.

- **DSSVerifyRequest_V1.xml**: Plantilla para la construcción de las peticiones a los servicios de validación y actualización de firmas, de @Firma.

- **EliminarContenidoDocumento_V1.xml:** Plantilla para la construcción de peticiones al servicio para eliminar el contenido de documentos, de @Firma.

- **EVisor_GenerateReportRequest_V1.xml**: Plantilla para la construcción de las peticiones al servicio de generación de informes, de eVisor.

- **EVisor_ValidationReportRequest_V1.xml**: Plantilla para la construcción de las peticiones al servicio de validación de informes firmados, de eVisor.

- **FirmaServidor_V1.xml**: Plantilla para la construcción de las peticiones al servicio de firma servidor, de @Firma.

- **FirmaServidorCoSign_V1.xml**: Plantilla para la construcción de las peticiones al servicio de firma servidor CoSign, de @Firma.

- **FirmaServidorCounterSign_V1.xml**: Plantilla para la construcción de las peticiones al servicio de firma servidor CounterSign, de @Firma.

- **FirmaUsuario2FasesF2_V1.xml**: Plantilla para la construcción de las peticiones al servicio fase 2 de firma usuario 2 fases, de @Firma.

- **FirmaUsuario3FasesF1_V1.xml**: Plantilla para la construcción de las peticiones al servicio fase 1 de firma usuario 3 fases, de @Firma.

- **FirmaUsuario3FasesF1CoSign_V1.xml**: Plantilla para la construcción de las peticiones al servicio fase 1 de firma usuario 3 fases CoSign, de @Firma.

- **FirmaUsuario3FasesF1CounterSign_V1.xml**: Plantilla para la construcción de las peticiones al servicio fase 1 de firma usuario 3 fases CounterSign, de @Firma.

- **FirmaUsuario3FasesF3_V1.xml**: Plantilla para la construcción de las peticiones al servicio fase 3 de firma usuario 3 fases, de @Firma.

- **GetInfoCertificate_V1.xml**: Plantilla para la construcción de las peticiones al servicio de obtención de información de certificado, en inglés, de @Firma.

- **GetTransactionSignature_V10.xml**: Plantilla para la construcción de las peticiones al servicio de obtener una firma a partir de su identificador de transacción, en inglés, de @Firma.

- **ObtenerContenidoDocumento_V1.xml**: Plantilla para la construcción de las peticiones al servicio para obtener el contenido de un documento, de @Firma.

- **ObtenerContenidoDocumentoId_V1.xml**: Plantilla para la construcción de las peticiones al servicio para obtener el contenido de un documento a partir de su identificador, de @Firma.

- **ObtenerFirmaTransaccion_V1.xml**: Plantilla para la construcción de las peticiones al servicio de obtener una firma a partir de su identificador de transacción, de @Firma.

- **ObtenerIdDocumento_V1.xml**: Plantilla para la construcción de las peticiones al servicio para obtener el identificador de un documento a partir de su identificador de transacción, de @Firma.

- **ObtenerInfoCertificado_V1.xml**: Plantilla para la construcción de las peticiones al servicio de obtención de información de certificado, de @Firma.

- **ServerSignature_V1.xml**: Plantilla para la construcción de las peticiones al servicio de firma servidor, en inglés, de @Firma.

- **ServerSignatureCoSign_V1.xml**: Plantilla para la construcción de las peticiones al servicio de firma servidor CoSign, en inglés, de @Firma.

- **ServerSignatureCounterSign_V1.xml**: Plantilla para la construcción de las peticiones al servicio de firma servidor CounterSign, en inglés, de @Firma.

- **Signature_Validation_V1.xml**: Plantilla para la construcción de las peticiones al servicio de validación de firmas, en inglés, de @Firma.

- **StoreDocument_V1.xml**: Plantilla para la construcción de las peticiones al servicio para almacenar documentos, en inglés, de @Firma.

- **ThreePhaseUserSignatureF1_V1.xml**: Plantilla para la construcción de las peticiones al servicio fase 1 de firma usuario 3 fases, en inglés, de @Firma.

- **ThreePhaseUserSignatureF1CoSign_V1.xml**: Plantilla para la construcción de las peticiones al servicio fase 1 de firma usuario 3 fases CoSign, en inglés, de @Firma.

- **ThreePhaseUserSignatureF1CounterSign_V1.xml**: Plantilla para la construcción de las peticiones al servicio fase 1 de firma usuario 3 fases CounterSign, en inglés, de @Firma.

- **ThreePhaseUserSignatureF3_V1.xml**: Plantilla para la construcción de las peticiones al servicio fase 3 de firma usuario 3 fases, en inglés, de @Firma.

- **TwoPhaseUserSignatureF2_V1.xml**: Plantilla para la construcción de las peticiones al servicio fase 2 de firma usuario 2 fases, en inglés, de @Firma.

- **Validacion_Firma_V1.xml**: Plantilla para la construcción de las peticiones al servicio de validación de firmas, de @Firma.

- **ValidarCertificado_V1.xml**: Plantilla para la construcción de las peticiones al servicio de validación de certificados, de @Firma.

- **ValidateCertificate_V1.xml**: Plantilla para la construcción de las peticiones al servicio de validación de certificados, en inglés, de @Firma.

8.12 Carpeta xml

Esta carpeta incluye definiciones de esquemas XML y definiciones de tipos de documentos necesarios para el correcto funcionamiento de los WS de TS@. Los archivos que la componen son:

- **claimedIdentity-schema.xsd**
- **datatypes.dtd**
- **oasis-dss-core-schema-v1.0-os.xsd**
- **oasis-dss-profiles-timestamping-schema-v1.0-r1.xsd**
- **oasis-sstc-saml-schema-protocol-1.1.xsd**
- **oasis-wss-wssecurity-secext-1.1.xsd**
- **XAdES.xsd**
- **xenc-schema.xsd**
- **xml.xsd**
- **xmldsig-core-schema.xsd**
- **XMLSchema.dtd**

8.13 Archivo mappingFiles.properties

En este archivo se mapean los nombre de los ficheros de propiedades con nombre dinámico a elección del integrador.

Pueden mapearse N ficheros de propiedades atendiendo a una serie de reglas básicas.

Cada línea tendrá el nombre de la variable que identifica en integra a un fichero de propiedades y la asignación del nombre del fichero a utilizar cuando se resuelva la variable. Sirva como ejemplo lo siguiente:

```
tsapruebasTest = tsapruebasTest.properties
afirmaafirmaTest = afirmaafirmaTest.properties
evisorafirmaTestEVisor = evisorafirmaTestEVisor.properties
integra = integra.properties
```

Para el fichero de propiedades generales integra buscará en mappingFiles.properties el valor para la variable **integra.**

Para el acceso a TSA, integra buscará en mappingFiles.properties el valor para la variable que comience por tsa y siga por el nombre de la aplicación TSA que se invoca. Por ejemplo
`tsapruebasTest = tsapruebasTest.properties.`

Para el acceso a Afirma, integra buscará en mappingFiles.properties el valor para la variable que comience por afirma y siga con el nombre de la aplicación afirma que se invoca. Por ejemplo
`afirmaafirmaTest = afirmaafirmaTest.properties.`

Para el acceso a eVisor, integra buscará en mappingFiles.properties el valor para la variable que comience por evisor y siga con el nombre de la aplicación eVisor que se invoca. Por ejemplo
`evisorafirmaTestEVisor = evisorafirmaTestEVisor.properties.`

Atendiendo a las reglas expuestas sería posible tener, por ejemplo varios ficheros de configuración de acceso a afirma (uno por aplicación) y se mapearían de forma similar a como se expone a continuación.

```
afirmaApp1 = afirma1.properties
afirmaApp2 = afirma2.properties
afirmaApp3 = afirma3.properties
```

8.14 Archivo integra.properties

Archivo con configuraciones generales. El nombre del fichero queda a elección del integrador y se definirá en el fichero mappingFiles.properties. Se propone como nombre integra.properties y habría que mapearlo de la siguiente forma

```
integra = integra.properties.
```

8.14.1 Parámetros Comunes a Todas las Aplicaciones de @Firma, eVisor y TS@:

- **com.trustedstorePath:** Ruta al almacén de certificados de confianza usado en comunicaciones seguras.

- **com.trustedstorePassword:** Contraseña del almacén de certificados de confianza usado en comunicaciones seguras.

8.14.2 Parámetros Comunes a Todas las Aplicaciones de @Firma:

- **com.certificatesCache.use:** Indicador para cachear las respuestas de validación de certificados por los servicios DSS simple, nativo en inglés y nativo en español para cada certificado. Los posibles valores son:

- **true** → Se cachean las respuestas de validación de certificados.

- **false** → No se cachean las respuestas de validación de certificados.

- **com.certificatesCache.entries:** Número de entradas que formarán la caché de respuestas de validación de certificados. Cada vez que se alcance esta cifra de respuestas de validación de certificados cacheadas, la próxima vez que se vaya a almacenar una entrada en dicha caché se eliminará previamente la entrada más antigua.

- **com.certificatesCache.lifeTime:** Tiempo de validez de cada entrada que formará la caché de respuestas de validación de certificados, en segundos. Cuando una entrada supere ese tiempo de validez será eliminada de la caché.

8.14.3 Propiedades para la Validación de Firmantes

Estas propiedades son necesarias para completar correctamente los procesos de firma, co-firma, contra-firma, actualización y validación de firmas.

- **CERTIFICATE_VALIDATION_LEVEL:** Nivel de validación de los firmantes. Esta propiedad está asociada a la validación de firmas CAdES (Baseline o no), XAdES (Baseline o no), PAdES (Baseline o no), y ASiC-S. Los valores posibles son:

  - **0** → Nivel de validación nulo. No se realiza verificación alguna sobre el certificado. Este modo no es aplicable a los procesos de actualización, es decir, si para un proceso de actualización de firma se ha indicado este nivel de verificación de certificado se realizará la validación indicada en el nivel con valor **1**.

  - **1** → Nivel de validación simple. Se verifica el periodo de validez y caducidad del certificado.

  - **2** → Nivel de validación completo. Se verifica el periodo de validez y caducidad del certificado, y el estado de revocación del certificado mediante servicio OCSP.

8.14.4 Propiedades para la Comunicación con TS@

Estas propiedades son necesarias para completar correctamente los procesos de firma, co-firma, contra-firma y actualización de firmas.

- **TSA_APP_ID:** Identificador de la aplicación cliente para la comunicación con TS@.

- **TSA_COMMUNICATION_TYPE:** Tipo de comunicación a usar para obtener el sello de tiempo de TS@. Los valores posibles son:

- **DSS** → Obtención del sello de tiempo mediante servicio web DSS.
- **RFC3161-TCP** → Obtención del sello de tiempo mediante servicio RFC 3161 - TCP.
- **RFC3161-HTTPS** → Obtención del sello de tiempo mediante servicio RFC 3161 - HTTPS.
- **RFC3161-SSL** → Obtención del sello de tiempo mediante servicio RFC 3161 - SSL.

- **TSA_TIMESTAMP_TYPE:** Tipo de sello de tiempo a solicitar a TS@. Los valores posibles son:
  - **ASN1**→ Sello de tiempo ASN.1.
  - **XML**→ Sello de tiempo XML.

8.14.5 Propiedades para la Fachada de Generación de Firmas, Co-Firmas y Contra-Firmas

Estas propiedades son necesarias para completar correctamente los procesos de firma, co-firma y contra-firma desde la fachada de invocación para los servicios propios de generación de firmas, co-firmas y contra-firmas de Integr@.

- **FACADE_SIGNATURE_ALGORITHM:** Algoritmo de firma a utilizar en la generación de firmas, co-firmas y contra-firmas. Los valores posibles son:
  - **SHA1withRSA**
  - **SHA256withRSA**
  - **SHA384withRSA**
  - **SHA512withRSA**

- **FACADE_SIGNATURE_TYPE:** Tipo de firma a generar. Los valores posibles son:
  - **CAdES**
  - **XAdES**
  - **PAdES**
  - **CAdES Baseline**
  - **XAdES Baseline**
  - **PAdES Baseline**

- **ASiC-S CAdES Baseline**
- **ASiC-S XAdES Baseline**

8.14.6 Propiedades de política

Propiedades necesarias para la validación de firmas con política de firma asociada. Admite múltiples implementaciones de políticas de firma, diferenciándolas en función del tipo de firma sobre la que aplique, esto es, ASN.1, XML y PDF. Debe tenerse en cuenta que cuando se nombra ASN.1 se engloba también a firmas ASiC-S Baseline que contengan una firma CAdES Baseline. Igualmente, cuando se nombra XML se engloba también a firmas ASiC-S Baseline que contengan una firma XAdES Baseline.

Para facilitar la comprensión de las propiedades, éstas se dividirán por bloques. Cada bloque permite la inclusión de nuevas propiedades.

Listado de OIDs para Elementos ASN.1

Este bloque define los elementos ASN.1 a los que se hará referencia en las distintas políticas de firma, de manera que por cada uno de ellos, se le asociará su respectivo OID. Así, la clave de cada elemento ASN.1 será un texto descriptivo con su nombre, y el valor será su OID. Por defecto, se definen los siguientes elementos ASN.1:

```
ContentType = 1.2.840.113549.1.9.3
MessageDigest = 1.2.840.113549.1.9.4
SigningCertificate = 1.2.840.113549.1.9.16.2.12
SigningCertificateV2 = 1.2.840.113549.1.9.16.2.47
SigningTime = 1.2.840.113549.1.9.5
SignaturePolicyIdentifier = 1.2.840.113549.1.9.16.2.15
ContentHints = 1.2.840.113549.1.9.16.2.4
ContentReference = 1.2.840.113549.1.9.16.2.10
ContentIdentifier = 1.2.840.113549.1.9.16.2.7
SignerLocation = 1.2.840.113549.1.9.16.2.17
SignerAttributes = 1.2.840.113549.1.9.16.2.18
ContentTimeStamp = 1.2.840.113549.1.9.16.2.20
CounterSignature = 1.2.840.113549.1.9.6
CommitmentTypeIndication = 1.2.840.113549.1.9.16.2.16
ArchiveTimeStamp = 1.2.840.113549.1.9.16.2.48
CompleteCertificateRefs = 1.2.840.113549.1.9.16.2.21
CompleteRevocationRefs = 1.2.840.113549.1.9.16.2.22
TimestampedCertsCRLs = 1.2.840.113549.1.9.16.2.26
Data = 1.2.840.113549.1.7.1
```

Todo elemento ASN.1 que se indique en la configuración de alguna política de firma deberá estar definido siguiendo el modelo antes indicado.

Listado de URIs para Algoritmos de Hash en Firmas XML

Este bloque define los algoritmos de hash para firmas XML a los que se hará referencia en las distintas políticas de firma, de manera que por cada uno de ellos, se le asociará su respectiva URI. Así, la clave de cada algoritmo de hash tendrá el formato **XML_HASH-HASH_ALGORITHM** donde:

- **XML_HASH** es una cadena de texto fijo
- **HASH_ALGORITHM** es una cadena de texto con el nombre algoritmo de hash

Por defecto, se definen los siguientes algoritmos de hash para firmas XML:

```
XML_HASH-SHA1 = http://www.w3.org/2000/09/xmldsig#sha1
XML_HASH-SHA256 = http://www.w3.org/2001/04/xmlenc#sha256
XML_HASH-SHA512 = http://www.w3.org/2001/04/xmlenc#sha512
```

Todo algoritmo de hash para firmas XML que se indique en la configuración de alguna política de firma deberá estar definido siguiendo el modelo antes indicado.

Listado de URIs para Algoritmos de Firma en Firmas XML

Este bloque define los algoritmos de firma para firmas XML a los que se hará referencia en las distintas políticas de firma, de manera que por cada uno de ellos, se le asociará su respectiva URI. Así, la clave de cada algoritmo de firma tendrá el formato **XML_SIGN_HASH-HASH_ALGORITHM** donde:

- **XML_SIGN_HASH** es una cadena de texto fijo
- **HASH_ALGORITHM** es una cadena de texto con el nombre del algoritmo de firma

Por defecto, se definen los siguientes algoritmos de firma para firmas XML:

```
XML_SIGN_HASH-SHA1WithRSA = http://www.w3.org/2000/09/xmldsig#rsa-sha1
XML_SIGN_HASH-SHA256WithRSA = http://www.w3.org/2001/04/xmldsig-more#rsa-sha256
XML_SIGN_HASH-SHA512WithRSA = http://www.w3.org/2001/04/xmldsig-more#rsa-sha512
```

Todo algoritmo de firma para firmas XML que se indique en la configuración de alguna política de firma deberá estar definido siguiendo el modelo antes indicado.

Listado de OIDs para Algoritmos de Hash en Firmas ASN.1 y PDF

Este bloque define los algoritmos de hash para firmas ASN.1 y PDF a los que se hará referencia en las distintas políticas de firma, de manera que por cada uno de ellos, se le asociará su respectivo OID. Así, la clave de cada algoritmo de hash tendrá el formato **ASN1_HASH-HASH_ALGORITHM** donde:

- **ASN1_HASH** es una cadena de texto fijo
- **HASH_ALGORITHM** es una cadena de texto con el nombre algoritmo de hash

Por defecto, se definen los siguientes algoritmos de hash para firmas ASN.1 y PDF:

```
ASN1_HASH-SHA1 = 1.3.14.3.2.26
ASN1_HASH-SHA256 = 2.16.840.1.101.3.4.2.1
ASN1_HASH-SHA512 = 2.16.840.1.101.3.4.2.3
```

Todo algoritmo de hash para firmas ASN.1 y PDF que se indique en la configuración de alguna política de firma deberá estar definido siguiendo el modelo antes indicado.

Listado de OIDs para Algoritmos de Firma en Firmas ASN.1 y PDF

Este bloque define los algoritmos de firma para firmas ASN.1 y PDF a los que se hará referencia en las distintas políticas de firma, de manera que por cada uno de ellos, se le asociará su respectivo OID. Así, la clave de cada algoritmo de firma tendrá el formato **ASN1_SIGN_HASH-HASH_ALGORITHM** donde:

- **ASN1_SIGN_HASH** es una cadena de texto fijo
- **HASH_ALGORITHM** es una cadena de texto con el nombre del algoritmo de firma

Por defecto, se definen los siguientes algoritmos de firma para firmas ASN.1 y PDF:

```
ASN1_SIGN_HASH-SHA1WithRSA = 1.2.840.113549.1.1.5
ASN1_SIGN_HASH-SHA256WithRSA = 1.2.840.113549.1.1.11
ASN1_SIGN_HASH-SHA512WithRSA = 1.2.840.113549.1.1.13
```

Todo algoritmo de firma para firmas ASN.1 y PDF que se indique en la configuración de alguna política de firma deberá estar definido siguiendo el modelo antes indicado.

Identificadores de Política de Firma por Tipo

Este bloque define 3 identificadores de política de firma, uno por cada tipo de firma:

- **XML_POLICY_ID:** Identificador de la política de firma a usar en la generación de firmas XML. El valor no puede contener los caracteres dos puntos (:), igual (=), o espacios en blanco. Por ejemplo:

```
XML_POLICY_ID = XML_AGE_1.9_URL
```

- **ASN1_POLICY_ID:** Identificador de la política de firma a usar en la generación de generar firmas ASN.1. El valor no puede contener los caracteres dos puntos (:), igual (=), o espacios en blanco. Por ejemplo:

```
ASN1_POLICY_ID = ASN1_AGE_1.9
```

- **PDF_POLICY_ID:** Identificador de la política de firma a usar en la generación de generar firmas PDF. El valor no puede contener los caracteres dos puntos (:), igual (=), o espacios en blanco. Por ejemplo:

```
PDF_POLICY_ID = PDF_AGE_1.9
```

Normas Asociadas a una Política de Firma

Este bloque define el conjunto de normas y restricciones para una política de firma concreta. Toda clave asociada a una política de firma se compone de 2 partes separadas por el carácter guión (-), de manera que la parte situada a la izquierda del guión se corresponderá con el identificador de la política de firma, y la parte situada a la derecha del guión se corresponderá con el identificador de la clave de norma. Por ejemplo, para indicar que el algoritmo de resumen a usar en el cálculo de la huella digital del documento legible de la política de firma con identificador *XML_AGE_1.8_XML* debe ser SHA-1 se identificaría de la siguiente manera:

```
XML_AGE_1.9_URL-HASH_ALGORITHM = SHA-1
```

A continuación, se describirá cada una de las normas:

8.14.6.1.1 Identificador de la Política de Firma

Este identificador puede servir para firmas ASN.1, XML y PDF.

Si la política de firma se refiere a firmas ASN.1 tendrá el formato **ID_POLICY-IDENTIFIER_ASN1** donde:

- **ID_POLICY** es el identificador de política de firma en este archivo
- **IDENTIFIER_ASN1** es una cadena de texto fijo

Por ejemplo:

```
ASN1_AGE_1.9-IDENTIFIER_ASN1 = 2.16.724.1.3.1.1.2.1.9
```

Si la política de firma se refiere a firmas XML tendrá el formato **ID_POLICY-IDENTIFIER_ XML** donde:

- **ID_POLICY** es el identificador de política de firma en este archivo
- **IDENTIFIER_ XML** es una cadena de texto fijo

Por ejemplo:

```
XML_AGE_1.9_URL-IDENTIFIER_XML =
http://administracionelectronica.gob.es/es/ctt/politicafirma/politica_firma_AGE_v1
_9.pdf
```

Si la política de firma se refiere a firmas PDF tendrá el formato **ID_POLICY-IDENTIFIER_ASN1** donde:

- **ID_POLICY** es el identificador de política de firma en este archivo
- **IDENTIFIER_PDF** es una cadena de texto fijo

Por ejemplo:

```
PDF_AGE_1.9-IDENTIFIER_PDF = 2.16.724.1.3.1.1.2.1.9
```

8.14.6.1.2 Algoritmo de Resumen a Usar para Calcular la Huella Digital del Documento Legible de Política de Firma

Tendrá el formato **ID_POLICY-HASH_ALGORITHM** donde:

- **ID_POLICY** es el identificador de política de firma en este archivo
- **HASH_ALGORITHM** es una cadena de texto fijo

Valores permitidos:

- SHA-1
- SHA-256
- SHA-512

Por ejemplo:

```
XML_AGE_1.9_URL-HASH_ALGORITHM = SHA-1
```

8.14.6.1.3 Valor del Resumen del Documento Legible de Política de Firma codificado en Base64

Tendrá el formato **ID_POLICY-HASH_VALUE** donde:

- **ID_POLICY** es el identificador de política de firma en este archivo
- **HASH_VALUE** es una cadena de texto fijo

Por ejemplo:

```
XML_AGE_1.9_URL-HASH_VALUE = 7SxX3erFuH31TvAw9LZ70N7p1vA=
```

8.14.6.1.4 Algoritmos de Resumen Válidos

Cada elemento estará separado por coma (,). Tendrá el formato **ID_POLICY-ALLOWED_HASH_ALGORITHM** donde

- **ID_POLICY** es el identificador de política de firma en este archivo
- **ALLOWED_HASH_ALGORITHM** es una cadena de texto fijo

Si el algoritmo de resumen es para una firma ASN.1 o PDF su nombre se cogerá de la lista de OIDs para algoritmos de hash en firmas ASN.1 y PDF descrita en 0. Por ejemplo:

```
ASN1_AGE_1.9-ALLOWED_HASH_ALGORITHM = ASN1_HASH-SHA1,ASN1_HASH-SHA256,ASN1_HASH-
SHA512
```

Si el algoritmo de resumen es para una firma XML su nombre se cogerá de la lista de URIs para algoritmos de hash en firmas XML descrita en 0. Por ejemplo:

```
XML_AGE_1.9_URL-ALLOWED_HASH_ALGORITHM = XML_HASH-SHA1,XML_HASH-SHA256,XML_HASH-
SHA512
```

8.14.6.1.5 Algoritmos de Firma Válidos

Cada elemento estará separado por coma (,). Tendrá el formato **ID_POLICY-ALLOWED_SIGN_ALGORITHM** donde

- **ID_POLICY** es el identificador de política de firma en este archivo
- **ALLOWED_SIGN_ALGORITHM** es una cadena de texto fijo

Si el algoritmo de firma es para una firma ASN.1 o PDF su nombre se cogerá de la lista de OIDs para algoritmos de firma en firmas ASN.1 y PDF descrita en 0. Por ejemplo:

```
ASN1_AGE_1.9-ALLOWED_SIGN_ALGORITHM = ASN1_SIGN_HASH-SHA1WithRSA,ASN1_SIGN_HASH-
SHA256WithRSA,ASN1_SIGN_HASH-SHA512WithRSA
```

Si el algoritmo de firma es para una firma XML su nombre se cogerá de la lista de OIDs para algoritmos de firma en firmas XML descrita en 0. Por ejemplo:

```
XML_AGE_1.9_URL-ALLOWED_SIGN_ALGORITHM = XML_SIGN_HASH-SHA1WithRSA,XML_SIGN_HASH-
SHA256WithRSA,XML_SIGN_HASH-SHA512WithRSA
```

8.14.6.1.6 Descripción de la Política de Firma

Tendrá el formato **ID_POLICY-DESCRIPTION** donde:

- **ID_POLICY** es el identificador de política de firma en este archivo
- **DESCRIPTION** es una cadena de texto fijo

Por ejemplo:

```
PDF_AGE_1.9-DESCRIPTION = Política de Firma Electrónica y de Certificados de la
Administración General del Estado 1.9 para firmas PAdES
```

8.14.6.1.7 Elementos Firmados Obligatorios

Los elementos pueden ser XML o ASN.1. Cada elemento estará separado por coma (,). Para especificar que se debe elegir entre varios se usará el operador lógico OR (|). Tendrá el formato **ID_POLICY-MANDATORY_SIGNED_ELEMENTS** donde

- **ID_POLICY** es el identificador de política de firma en este archivo
- **MANDATORY_SIGNED_ELEMENTS** es una cadena de texto fijo

Si el elemento es ASN.1 su nombre se cogerá de la lista de OIDs descrita en 0. Por ejemplo:

```
ASN1_AGE_1.9-MANDATORY_SIGNED_ELEMENTS =
ContentType,MessageDigest,SigningCertificate|SigningCertificateV2,SigningTime,Sig
naturePolicyIdentifier,ContentHints
```

Si el elemento es XML se incluirá sin el prefijo asociado a su espacio de nombres. Sólo se admiten elementos cuyo prefijo para el espacio de nombres pueda ser *ds*, *xades* o *xadesv141*. Por ejemplo:

```
XML_AGE_1.9_URL-MANDATORY_SIGNED_ELEMENTS =
SigningTime,SigningCertificate,SignaturePolicyIdentifier,DataObjectFormat
```

8.14.6.1.8 Elementos Firmados Opcionales

Los elementos pueden ser XML o ASN.1. Cada elemento estará separado por coma (,). Para especificar que se debe elegir entre varios se usará el operador lógico OR (|). Tendrá el formato **ID_POLICY-OPTIONAL_SIGNED_ELEMENTS** donde

- **ID_POLICY** es el identificador de política de firma en este archivo
- **OPTIONAL_SIGNED_ELEMENTS** es una cadena de texto fijo

Si el elemento es ASN.1 su nombre se cogerá de la lista de OIDs descrita en 0. Por ejemplo:

```
ASN1_AGE_1.9-OPTIONAL_SIGNED_ELEMENTS =
ContentReference,ContentIdentifier,CommitmentTypeIndication,SignerLocation,Signer
Attributes,ContentTimeStamp
```

Si el elemento es XML se incluirá sin el prefijo asociado a su espacio de nombres. Sólo se admiten elementos cuyo prefijo para el espacio de nombres pueda ser *ds*, *xades* o *xadesv141*. Por ejemplo:

```
XML_AGE_1.9_URL-OPTIONAL_SIGNED_ELEMENTS =
SignatureProductionPlace,SignerRole,CommitmentTypeIndication,AllDataObjectsTimeSt
amp,IndividualDataObjectsTimeStamp
```

8.14.6.1.9 Elementos no Firmados Obligatorios

Los elementos pueden ser XML o ASN.1. Cada elemento estará separado por coma (,). Para especificar que se debe elegir entre varios se usará el operador lógico OR (|). Tendrá el formato **ID_POLICY-MANDATORY_UNSIGNED_ELEMENTS** donde

- **ID_POLICY** es el identificador de política de firma en este archivo
- **MANDATORY_UNSIGNED_ELEMENTS** es una cadena de texto fijo

Si el elemento es ASN.1 su nombre se cogerá de la lista de OIDs descrita en 0. Por ejemplo:

```
ASN1_AGE_1.9-MANDATORY_UNSIGNED_ELEMENTS =
CompleteCertificateRefs,CompleteRevocationRefs,ArchiveTimeStamp|TimestampedCertsC
RLs
```

Si el elemento es XML se incluirá sin el prefijo asociado a su espacio de nombres. Sólo se admiten elementos cuyo prefijo para el espacio de nombres pueda ser *ds*, *xades* o *xadesv141*. Por ejemplo:

```
XML_AGE_1.9_URL-MANDATORY_UNSIGNED_ELEMENTS =
SignatureTimeStamp,SigAndRefsTimeStamp|RefsOnlyTimeStamp,ArchiveTimeStamp
```

8.14.6.1.10 Elementos no Firmados Opcionales

Los elementos pueden ser XML o ASN.1. Cada elemento estará separado por coma (,). Para especificar que se debe elegir entre varios se usará el operador lógico OR (|). Tendrá el formato **ID_POLICY-OPTIONAL_UNSIGNED_ELEMENTS** donde

- **ID_POLICY** es el identificador de política de firma en este archivo
- **OPTIONAL_UNSIGNED_ELEMENTS** es una cadena de texto fijo

Si el elemento es ASN.1 su nombre se cogerá de la lista de OIDs descrita en 0. Por ejemplo:

```
ASN1_AGE_1.9-OPTIONAL_UNSIGNED_ELEMENTS = CounterSignature
```

Si el elemento es XML se incluirá sin el prefijo asociado a su espacio de nombres. Sólo se admiten elementos cuyo prefijo para el espacio de nombres pueda ser *ds*, *xades* o *xadesv141*. Por ejemplo:

```
XML_AGE_1.9_URL-OPTIONAL_UNSIGNED_ELEMENTS = CounterSignature
```

8.14.6.1.11 Elementos Firmados no Permitidos

Los elementos pueden ser XML o ASN.1. Cada elemento estará separado por coma (,). Tendrá el formato **ID_POLICY-NOT_ALLOWED_SIGNED_ELEMENTS** donde

- **ID_POLICY** es el identificador de política de firma en este archivo
- **NOT_ALLOWED_SIGNED_ELEMENTS** es una cadena de texto fijo

Si el elemento es ASN.1 su nombre se cogerá de la lista de OIDs descrita en 0. Por ejemplo:

```
ASN1_AGE_1.9-NOT_ALLOWED_SIGNED_ELEMENTS =
ContentType,MessageDigest,SigningCertificateV2
```

Si el elemento es XML se incluirá sin el prefijo asociado a su espacio de nombres. Sólo se admiten elementos cuyo prefijo para el espacio de nombres pueda ser *ds*, *xades* o *xadesv141*. Por ejemplo:

```
XML_AGE_1.9_URL-NOT_ALLOWED_SIGNED_ELEMENTS =
SigningTime,SigningCertificate,DataObjectFormat
```

8.14.6.1.12 Elementos no Firmados no Permitidos

Los elementos pueden ser XML o ASN.1. Cada elemento estará separado por coma (,). Tendrá el formato **ID_POLICY-NOT_ALLOWED_UNSIGNED_ELEMENTS** donde

- **ID_POLICY** es el identificador de política de firma en este archivo
- **NOT_ALLOWED_UNSIGNED_ELEMENTS** es una cadena de texto fijo

Si el elemento es ASN.1 su nombre se cogerá de la lista de OIDs descrita en 0. Por ejemplo:

```
ASN1_AGE_1.9-NOT_ALLOWED_UNSIGNED_ELEMENTS =
CompleteCertificateRefs,CompleteRevocationRefs
```

Si el elemento es XML se incluirá sin el prefijo asociado a su espacio de nombres. Sólo se admiten elementos cuyo prefijo para el espacio de nombres pueda ser *ds*, *xades* o *xadesv141*. Por ejemplo:

```
XML_AGE_1.9_URL-NOT_ALLOWED_UNSIGNED_ELEMENTS =
SignatureTimeStamp,SigAndRefsTimeStamp,ArchiveTimeStamp
```

8.14.6.1.13 Entradas Obligatorias

Incluye una lista con los nombres de las entradas de carácter obligatorio que debe contener la firma. Cada entrada se incluirá con el prefijo barra inclinada (/). Esta propiedad sólo se usará en el caso de firmas PAdES. Cada entrada estará separada por coma (,). Para especificar que se debe elegir entre varias se usará el operador lógico OR (|). Tendrá el formato **ID_POLICY-REQUIRED_ENTRIES** donde

- **ID_POLICY** es el identificador de política de firma en este archivo
- **REQUIRED_ENTRIES** es una cadena de texto fijo

Por ejemplo:

```
PDF_AGE_1.9-NOT_ALLOWED_ENTRIES = /M,/Location,/Reason|/Cert
```

8.14.6.1.14 Entradas Opcionales

Incluye una lista con los nombres de las entradas de carácter opcional que puede contener la firma. Cada entrada se incluirá con el prefijo barra inclinada (/). Esta propiedad sólo se usará en el caso de firmas PAdES. Cada entrada estará separada por coma (,). Para especificar que se debe elegir entre varios se usará el operador lógico OR (|). Tendrá el formato **ID_POLICY-OPTIONAL_ENTRIES** donde

- **ID_POLICY** es el identificador de política de firma en este archivo
- **OPTIONAL_ENTRIES** es una cadena de texto fijo

Por ejemplo:

```
PDF_AGE_1.9-OPTIONAL_ENTRIES = /M,/Location
```

8.14.6.1.15 Entradas no Permitidas

Incluye una lista con los nombres de las entradas que no puede contener la firma. Cada entrada se incluirá con el prefijo barra inclinada (/). Esta propiedad sólo se usará en el caso de firmas PAdES. Cada entrada estará separada por coma (,). Tendrá el formato **ID_POLICY- NOT_ALLOWED_ENTRIES** donde

- **ID_POLICY** es el identificador de política de firma en este archivo
- **NOT_ALLOWED_ENTRIES** es una cadena de texto fijo

Por ejemplo:

```
PDF_AGE_1.9-NOT_ALLOWED_ENTRIES = /Cert,/Reason
```

8.14.6.1.16 Elementos Hijos Obligatorios

Para un elemento se puede especificar su lista de hijos requeridos. Los elementos sólo pueden ser XML. Si el elemento es XML se incluirá SIN el prefijo asociado a su espacio de nombres. Sólo se admiten elementos cuyo espacio de nombres pueda ser *ds*, *xades* o *xadesv141*. Cada elemento estará separado por coma (,). Para especificar que se debe elegir entre varios se usará el operador lógico OR (|). Tendrá el formato **ID_POLICY-[ELEMENT_NAME]-REQUIRED_CHILD** donde

- **ID_POLICY** es el identificador de política de firma en este archivo
- **ELEMENT_NAME** es el nombre del elemento XML
- **REQUIRED_CHILD** es una cadena de texto fijo

Por ejemplo:

```
XML_AGE_1.9_URL-[SignerRole]-REQUIRED_CHILD = ClaimedRoles|CertifiedRoles
```

8.14.6.1.17 Elementos Hijos Opcionales

Para un elemento se puede especificar su lista de hijos opcionales. Los elementos sólo pueden ser XML. Si el elemento es XML se incluirá SIN el prefijo asociado a su espacio de nombres. Sólo se admiten elementos cuyo espacio de nombres pueda ser *ds*, *xades* o *xadesv141*. Cada elemento estará separado por coma (,). Para especificar que se debe elegir entre varios se usará el operador lógico OR (|). Tendrá el formato **ID_POLICY-[ELEMENT_NAME]-OPTIONAL_CHILD** donde

- **ID_POLICY** es el identificador de política de firma en este archivo
- **ELEMENT_NAME** es el nombre del elemento XML
- **OPTIONAL_CHILD** es una cadena de texto fijo

Por ejemplo:

```
XML_AGE_1.9_URL-[SignerRole]-OPTIONAL_CHILD = ClaimedRoles|CertifiedRoles
```

8.14.6.1.18 Elementos Hijos no Permitidos

Para un elemento se puede especificar su lista de hijos opcionales. Los elementos sólo pueden ser XML. Si el elemento es XML se incluirá SIN el prefijo asociado a su espacio de nombres. Sólo se admiten elementos cuyo espacio de nombres pueda ser *ds*, *xades* o *xadesv141*. Cada elemento estará separado por coma (,). Tendrá el formato **ID_POLICY-[ELEMENT_NAME]-NOT_ALLOWED_CHILD** donde

- **ID_POLICY** es el identificador de política de firma en este archivo
- **ELEMENT_NAME** es el nombre del elemento XML
- **NOT_ALLOWED_CHILD** es una cadena de texto fijo

Por ejemplo:

```
XML_AGE_1.9_URL-[SignerRole]-NOT_ALLOWED_CHILD = ClaimedRoles,CertifiedRoles
```

8.14.6.1.19 Valor Obligatorio de Elemento

Para un elemento se puede especificar su valor requerido. Los elementos pueden ser XML, ASN.1 o PDF. Si el elemento admite más de un valor posible se debe elegir entre varios mediante el operador lógico OR (|). Tendrá el formato **ID_POLICY-[ELEMENT_NAME]- REQUIRED_VALUE** donde

- **ID_POLICY** es el identificador de política de firma en este archivo
- **ELEMENT_NAME** es el nombre del elemento
- **REQUIRED_VALUE** es una cadena de texto fijo

Si el elemento es ASN.1 su nombre se cogerá de la lista de OIDs descrita en 0. Por ejemplo:

```
ASN1_AGE_1.9-[ContentType]-REQUIRED_VALUE = Data
```

Si el elemento es XML se incluirá SIN el prefijo asociado a su espacio de nombres. Sólo se admiten elementos cuyo espacio de nombres pueda ser *ds*, *xades* o *xadesv141*. Por ejemplo:

```
XML_AGE_1.9_URL-[ClaimedRoles]-REQUIRED_VALUE =
supplier|emisor|customer|receptor|third party|tercero
```

Si el elemento es PDF hará referencia a una entrada. La entrada se incluirá con el prefijo barra inclinada (/). Por ejemplo:

```
PDF_AGE_1.9-[/SubFilter]-REQUIRED_VALUE = ETSI.CAdES.detached
```

8.14.6.1.20 Valores no Permitidos para un Elemento

Para un elemento se puede especificar los valores no permitidos. Los elementos pueden ser XML, ASN.1 o PDF. Si el elemento no admite ningún valor de una lista de posibles, cada valor estará separado por coma (,). Tendrá el formato **ID_POLICY-[ELEMENT_NAME]-NOT_ALLOWED_VALUE** donde

- **ID_POLICY** es el identificador de política de firma en este archivo
- **ELEMENT_NAME** es el nombre del elemento
- **NOT_ALLOWED_VALUE** es una cadena de texto fijo

Si el elemento es ASN.1 su nombre se cogerá de la lista de OIDs descrita en 0. Por ejemplo:

```
ASN1_AGE_1.9-[ContentType]-NOT_ALLOWED_VALUE = Data
```

Si el elemento es XML se incluirá SIN el prefijo asociado a su espacio de nombres. Sólo se admiten elementos cuyo espacio de nombres pueda ser *ds*, *xades* o *xadesv141*. Por ejemplo:

```
XML_AGE_1.9_URL-[ClaimedRoles]-NOT_ALLOWED_VALUE = emisor,receptor,tercero
```

Si el elemento es PDF hará referencia a una entrada. La entrada se incluirá con el prefijo barra inclinada (/). Por ejemplo:

```
PDF_AGE_1.9-[/SubFilter]-NOT_ALLOWED_VALUE = adbe.pkcs7.s5
```

8.14.6.1.21 Modos de Firma Permitidos

Sólo aplicable a firmas XML y ASN.1. Establece los modos de firma permitidos. Si el tipo de firma admite más de un modo de firma, cada valor estará separado por coma (,). Para el caso de firmas ASN.1 los modos de firma posibles son el implícito, (la firma incluye los datos originales), y el explícito, (la firma no incluye los datos originales), identificados respectivamente por:

- Implicit
- Explicit

Por ejemplo, para indicar que los modos de firma permitidos para firmas ASN.1 son ambos, (implícito y explícito), se definiría:

```
ASN1_AGE_1.9-ALLOWED_SIGNING_MODES = Implicit,Explicit
```

Para el caso de firmas XML los modos de firma posibles son “detached”, (la firma incluye una referencia a los datos firmados), “enveloped”, (la firma incluye los datos firmados), y “enveloping”, (la firma posee una estructura XML que contiene internamente el contenido firmado en un nodo propio), identificados respectivamente por:

- Detached
- Enveloped
- Enveloping

Por ejemplo, para indicar que el modo de firma permitido para firmas XML es únicamente “enveloped” se definiría:

```
XML_AGE_1.9_URL-ALLOWED_SIGNING_MODES = Enveloped
```

8.14.7 Propiedades para acceso OCSP

Propiedades asociadas a la comunicación con un servidor OCSP para la validación del estado de revocación de certificados.

- **OCSP_URL:** URL de acceso al servicio OCSP.

- **OCSP_ISSUER_KEYSTORE_PATH:** Ruta del almacén de claves donde se almacenan los certificados raíz emisores del certificado a validar.

- **OCSP_ISSUER_KEYSTORE_TYPE:** Tipo del almacén de claves donde se almacenan los certificados raíz emisores del certificado a validar. Los valores posibles son:

  - **PKCS12**
  - **JKS**
  - **JCEKS**

- **OCSP_ISSUER_KEYSTORE_PASSWORD:** Contraseña del almacén de claves donde se almacenan los certificados raíz emisores del certificado a validar.

- **OCSP_RESPONSE_CERTIFICATE_PATH:** Ruta del certificado con el que firma la respuesta el servidor OCSP.

- **OCSP_TIMEOUT:** Tiempo de vida definido para la comunicación con el servidor OCSP, en milisegundos.

- **OCSP_SIGN_REQUEST:** Indicador para firmar la petición OCSP. Los valores posibles son:

  - **true** → La petición OCSP debe ir firmada.
  - **false** → La petición OCSP no debe ir firmada.

- **OCSP_REQUEST_KEYSTORE_PATH:** Ruta al almacén de claves donde se encuentra almacenada la clave privada a usar para firmar la petición SOAP en el caso de autenticación por certificado.

- **OCSP_REQUEST_KEYSTORE_TYPE:** Tipo de almacén de claves donde se encuentra almacenada la clave privada a usar para firmar la petición SOAP en el caso de autenticación por certificado. Los valores posibles son:

  - **PKCS12**
  - **JKS**
  - **JCEKS**

- **OCSP_REQUEST_KEYSTORE_PASSWORD:** Contraseña del almacén de claves donde se encuentra almacenada la clave privada a usar para firmar la petición SOAP en el caso de autenticación por certificado.

- **OCSP_REQUEST_PRIVATE_KEY_ALIAS:** Alias de la clave privada a usar para firmar la petición SOAP en el caso de autenticación por certificado.

- **OCSP_REQUEST_PRIVATE_KEY_PASSWORD:** Contraseña de la clave privada a usar para firmar la petición SOAP en el caso de autenticación por certificado.

- **OCSP_APP_ID:** Identificador de la aplicación cliente para indicar en la petición OCSP.

- **OCSP_USE_GET:** Indicador para realizar la comunicación con el servidor OCSP mediante una petición GET o POST. Los valores posibles son:

  - **true** → La petición es de tipo GET.

  - **false** → La petición es de tipo POST.

- **OCSP_TRUSTEDSTORE_PATH:** Ruta al almacén de confianza (JKS) para conexiones seguras.

- **OCSP_TRUSTEDSTORE_PASSWORD:** Contraseña del almacén de confianza (JKS) para conexiones seguras.

- **OCSP_HTTPS_USE_AUTH_CLIENT:** Indicador para utilizar autenticación HTTPS mediante certificado cliente. Los valores posibles son:

  - **true** → El cliente se autentica mediante certificado.

  - **false** → El cliente no requiere de autenticación mediante certificado.

- **OCSP_HTTPS_KEYSTORE_PATH:** Ruta al almacén de claves donde se encuentra almacenada la clave privada a usar para la autenticación HTTPS del cliente por certificado.

- **OCSP_HTTPS_KEYSTORE_TYPE:** Tipo de almacén de claves donde se encuentra almacenada la clave privada a usar para la autenticación HTTPS del cliente por certificado. Los valores posibles son:

  - **PKCS12**

  - **JKS**

  - **JCEKS**

- **OCSP_HTTPS_KEYSTORE_PASSWORD:** Contraseña del almacén de claves donde se encuentra almacenada la clave privada a usar para la autenticación HTTPS del cliente por certificado.

8.15 Archivo tsaXXXXX.properties

Archivo con configuraciones de acceso a TS@. El nombre del fichero queda a elección del integrador y se definirá en el fichero mappingFiles.properties. Se propone como nombre tsaXXXXX.properties donde XXXXX se refiere a la aplicación a configurar y habría que mapearlo de la siguiente forma

```
tsaNombreAplicacion = tsaXXXXX.properties.
```

Este archivo define las propiedades asociadas a la invocación de los servicios de TS@. Es necesario para el uso de las clases definidas en los paquetes **tsaServiceInvoker** y **rfc3161TSAServiceInvoker**.

8.15.1 Parámetros Específicos a Cada Una de las Aplicaciones Configuradas en la Plataforma de TS@

- **secureMode:** Indicador de invocación del servicio web mediante un canal seguro. Los valores posibles son:
  - **true** → La invocación al servicio web se hará mediante un canal seguro.
  - **false** → La invocación al servicio web se hará mediante un canal no seguro.
- **endPoint:** IP y puerto donde se encuentran publicados los distintos WS. Sigue el formato **IP:Puerto** donde:
  - **IP** es la IP donde se encuentran publicados los distintos WS
  - **Puerto** es el número del puerto donde se encuentran publicados los distintos WS
- **servicePath:** Ruta donde se encuentran publicados los distintos servicios web de TS@.
- **callTimeout:** Tiempo de vida para las peticiones SOAP, en milisegundos.
- **renewTimeStampWS.validationLevel:** Modo de validación para los sellos de tiempo que van a ser renovados. Los valores posibles son:
  - 0 → No se llevará a cabo ninguna validación. **Este modo va a en contra del estándar definido por OASIS que establece que en una operación de renovación de sello de tiempo el cliente debe validar el sello de tiempo previamente**.
  - 1 → Validación de la integridad. Se comprobará si el documento de entrada (*InputDocument*) asociado al sello de tiempo es correcto.
  - 2 → Validación completa. El sello de tiempo a renovar será previamente validado contra TS@.

- **authorizationMethod:** Tipo de autenticación a utilizar para las peticiones SOAP. Los valores posibles son:

  - **UserNameToken** → Autorización por usuario y contraseña.
  - **X509CertificateToken** → Autorización por certificado.
  - **SAMLToken** → Autorización por SAML.

- **UserNameToken.userName:** Nombre de usuario para el caso de autenticación por usuario y contraseña para la petición SOAP.

- **UserNameToken.userPassword:** Contraseña de usuario para el caso de autenticación por usuario y contraseña para la petición SOAP.

- **X509CertificateToken.inclusionMethod:** Mecanismo de inclusión del certificado para el caso de autorización por certificado para la petición SOAP. Los valores posibles son:

  - **Direct** → Binary Security Token.
  - **Identifier** → Key Identifier.
  - **IssuerSerialNumber** → Issuer and Serial Number.

- **X509CertificateToken.keystorePath:** Ruta al almacén de claves donde se encuentra almacenada la clave privada a usar para firmar la petición SOAP en el caso de autenticación por certificado.

- **X509CertificateToken.keystoreType:** Tipo de almacén de claves donde se encuentra almacenada la clave privada a usar para firmar la petición SOAP en el caso de autenticación por certificado. Los valores posibles son:

  - **PKCS12**
  - **JKS**
  - **JCEKS**

- **X509CertificateToken.keystorePassword:** Contraseña del almacén de claves donde se encuentra almacenada la clave privada a usar para firmar la petición SOAP en el caso de autenticación por certificado.

- **X509CertificateToken.privateKeyAlias:** Alias de la clave privada a usar para firmar la petición SOAP en el caso de autenticación por certificado.

- **X509CertificateToken.privateKeyPassword:** Contraseña de la clave privada a usar para firmar la petición SOAP en el caso de autenticación por certificado.

- **SAMLToken.method:** Método de confirmación del sujeto para el caso de autorización por SAML. Los valores posibles son:

  - **HOK** → Holder-of-Key.
  - **SV** → Sender-Vouches.

- **SAMLToken.keystorePath:** Ruta al almacén de claves donde se encuentra almacenada la clave privada a usar para firmar la petición SOAP en el caso de autenticación por SAML.

- **SAMLToken.keystoreType:** Tipo de almacén de claves donde se encuentra almacenada la clave privada a usar para firmar la petición SOAP en el caso de autenticación por SAML. Los valores posibles son:

  - **PKCS12**
  - **JKS**
  - **JCEKS**

- **SAMLToken.keystorePassword:** Contraseña del almacén de claves donde se encuentra almacenada la clave privada a usar para firmar la petición SOAP en el caso de autenticación por SAML.

- **SAMLToken.privateKeyAlias:** Alias de la clave privada a usar para firmar la petición SOAP en el caso de autenticación por SAML.

- **SAMLToken.privateKeyPassword:** Contraseña de la clave privada a usar para firmar la petición SOAP en el caso de autenticación por SAML.

- **request.symmetricKey.use:** Indicador para cifrar las peticiones SOAP con clave simétrica o no. Los valores posibles son:

  - **true** → Las peticiones SOAP irán cifradas.
  - **false** → Las peticiones SOAP no irán cifradas.

- **request.symmetricKey.alias:** Alias de la clave simétrica a usar para cifrar las peticiones SOAP.

- **request.symmetricKey.value**: Valor de la clave simétrica, en hexadecimal, a usar para cifrar las peticiones SOAP.

- **request.symmetricKey.algorithm:** Cadena que identifica de forma única el algoritmo a utilizar para realizar la firma en el cifrado simétrico.

- **Response.validate:** Indicador para validar las respuestas SOAP firmadas o no. Los valores posibles son:

- **true** → Las respuestas SOAP se validarán.
- **false** → Las respuestas SOAP no se validarán.

- **response.keystorePath:** Ruta al almacén de claves donde se encuentra almacenado el certificado a usar para validar las respuestas SOAP que se encuentren firmadas.
- **response.keystoreType:** Tipo de almacén de claves donde se encuentra almacenado el certificado a usar para validar las respuestas SOAP que se encuentren firmadas. Los valores posibles son:
  - **PKCS12**
  - **JKS**
  - **JCEKS**
- **response.keystorePassword:** Contraseña del almacén de claves donde se encuentra almacenado el certificado a usar para validar las respuestas SOAP que se encuentren firmadas.
- **response.certificateAlias:** Alias del certificado a usar para validar las respuestas SOAP que se encuentren firmadas.
- **response.SAML.keystorePath:** Ruta al almacén de claves donde se encuentra almacenado el certificado a usar para validar las respuestas SOAP aseguradas con SAML.
- **response.SAML.keystoreType:** Tipo de almacén de claves donde se encuentra almacenado el certificado a usar para validar las respuestas SOAP aseguradas con SAML. Los valores posibles son:
  - **PKCS12**
  - **JKS**
  - **JCEKS**
- **response.SAML.keystorePassword:** Contraseña del almacén de claves donde se encuentra almacenado el certificado a usar para validar las respuestas SOAP aseguradas con SAML.
- **response.SAML.certificateAlias:** Alias del certificado a usar para validar las respuestas SOAP aseguradas con SAML.
- **response.symmetricKey.alias:** Alias de la clave simétrica a usar para descifrar las respuestas SOAP cifradas con clave simétrica.
- **response.symmetricKey.value:** Valor de la clave simétrica, en hexadecimal, a usar para descifrar las respuestas SOAP cifradas con clave simétrica.

- **rfc3161.host:** Dirección host donde se encuentra desplegado el servicio RFC 3161.

- **rfc3161.timestampPolicyOID:** OID de la política de sello de tiempo a indicar en la petición.

- **rfc3161.applicationOID:** OID de la aplicación a indicar en la petición. No es necesario en caso de atacar a una TSA que no requiera aplicación cliente en la invocación.

- **rfc3161.Timeout:** Tiempo de vida para las peticiones al servicio RFC 3161, en milisegundos.

- **rfc3161.shaAlgorithm:** Algoritmo de resumen que aplicar sobre los datos a sellar. Los valores posibles son:

    - **SHA**
    - **SHA-256**
    - **SHA-512**
    - **RIPEMD-160**

- **rfc3161.portNumber:** Número del puerto donde se encuentra desplegado el servicio RFC 3161.

- **rfc3161HTTPS.portNumber:** Número del puerto donde se encuentra desplegado el servicio RFC 3161 - HTTPS.

- **rfc3161HTTPS.context:** Contexto para la conexión con el servicio RFC 3161 por HTTPS.

- **rfc3161HTTPS.useAuthClient:** Indicador para utilizar autenticación HTTPS mediante certificado cliente o no. Los valores posibles son:

    - **true** → El cliente se autentica mediante certificado.
    - **false** → El cliente no requiere de autenticación mediante certificado.

- **rfc3161HTTPS.keystorePath:** Ruta al almacén de claves donde se encuentra almacenada la clave privada a usar para la autenticación HTTPS del cliente por certificado.

- **rfc3161HTTPS.keystoreType:** Tipo de almacén de claves donde se encuentra almacenada la clave privada a usar para la autenticación HTTPS del cliente por certificado. Los valores posibles son:

    - **PKCS12**
    - **JKS**
    - **JCEKS**

- **rfc3161HTTPS.keystorePassword:** Contraseña del almacén de claves donde se encuentra almacenada la clave privada a usar para la autenticación HTTPS del cliente por certificado.

- **rfc3161SSL.portNumber:** Número del puerto donde se encuentre desplegado el servicio RFC 3161 que permite autenticación por SSL.

- **rfc3161SSL.keystorePath:** Ruta al almacén de claves donde se encuentra almacenada la clave privada a usar para la autenticación por SSL.

- **rfc3161SSL.keystoreType:** Tipo de almacén de claves donde se encuentra almacenada la clave privada a usar para la autenticación por SSL. Los valores posibles son:

    - **PKCS12**
    - **JKS**
    - **JCEKS**

- **rfc3161SSL.keystorePassword:** Contraseña del almacén de claves donde se encuentra almacenada la clave privada a usar para la autenticación por SSL.

8.16 Archivo afirmaXXXXX.properties

Archivo con configuraciones de acceso a aFirma. El nombre del fichero queda a elección del integrador y se definirá en el fichero mappingFiles.properties. Se propone como nombre afirmaXXXXX.properties donde XXXXX se refiere a la aplicación a configurar y habría que mapearlo de la siguiente forma

```
afirmaNombreAplicacion = afirmaXXXXX.properties.
```

Este archivo define las propiedades asociadas a la invocación de los servicios de @Firma. Es necesario para el uso de las clases definidas en el paquete **afirma5ServiceInvoker**.

8.16.1 Parámetros Específicos a Cada Una de las Aplicaciones Configuradas en la Plataforma de @Firma:

- **secureMode:** Indicador de invocación del servicio web mediante un canal seguro. Los valores posibles son:

    - **true** → La invocación al servicio web se hará mediante un canal seguro.
    - **false** → La invocación al servicio web se hará mediante un canal no seguro.

- **endPoint:** IP y puerto donde se encuentran publicados los distintos WS. Sigue el formato **IP:Puerto** donde:

- **IP** es la IP donde se encuentran publicados los distintos WS
- **Puerto** es el número del puerto donde se encuentran publicados los distintos WS

- **servicePath:** Ruta donde se encuentran publicados los distintos servicios web de @Firma.
- **callTimeout:** Tiempo de espera máximo, en milisegundos, para una solicitud de servicio.
- **authorizationMethod:** Método de autorización de ejecución de servicios Web empleado. Los valores posibles son:
  - **none** → No se utilizará ningún método de autorización.
  - **UsernameToken** → Se utilizará autorización mediante usuario y contraseña.
  - **BinarySecurity** → Se utilizará autorización mediante certificado digital.
- **authorizationMethod.user:** Nombre de usuario para el método de autorización **UsernameToken**, o alias del certificado (*.p12 o *.pfx) empleado para el método de autorización **BinarySecurity**.
- **authorizationMethod.password:** Contraseña del usuario para el método de autorización **UsernameToken**, o contraseña del certificado (*.p12 o *.pfx) empleado para el método de autorización **BinarySecurity**.
- **authorizationMethod.passwordType:** Modo en que se transmitirá la contraseña del usuario indicado en el método de autorización **UsernameToken**. Los valores posibles son:
  - **clear** → Se envía la contraseña en formato de texto plano.
  - **digest** → Se envía el resumen de la contraseña.
- **authorizationMethod.userKeystore:** Ruta al almacén de certificados donde se encuentra almacenado el certificado empleado para el método de autorización **BinarySecurity**.
- **authorizationMethod.userKeystorePassword:** Contraseña del almacén de certificados donde se encuentra almacenado el certificado empleado para el método de autorización **BinarySecurity**.
- **authorizationMethod.userKeystoreType:** Tipo del almacén de certificados donde se encuentra almacenado el certificado empleado para el método de autorización **BinarySecurity**. Los valores posibles son:
  - **JKS**
  - **PKCS12**
  - **JCEKS**

- **response.validate:** Indicador para validar la firma de la respuesta de los servicios de @Firma, en caso de encontrarse firmada. Los valores posibles son:

  - **true** → Se validará la firma de las respuestas de los servicios de @Firma que se encuentren firmadas.

  - **false** → No se validará la firma de las respuestas de los servicios de @Firma que se encuentren firmadas.

- **response.certificateAlias:** Alias empleado para identificar la clave pública del certificado usado en la firma de la respuesta de los servicios de @Firma. La clave pública estará almacenada en el almacén de claves indicado en la propiedad **authorizationMethod.userKeystore**.

8.17 Archivo evisorXXXXX.properties

Archivo con configuraciones de acceso a eVisor. El nombre del fichero queda a elección del integrador y se definirá en el fichor mappingFiles.properties. Se propone como nombre evisorXXXXX.properties donde XXXXX se refiere a la aplicación a configurar y habría que mapearlo de la siguiente forma

```
evisorNombreAplicacion = evisorXXXXX.properties.
```

Este archivo define las propiedades asociadas a la invocación de los servicios de @Firma. Es necesario para el uso de las clases definidas en el paquete **afirma5ServiceInvoker** en lo referente al acceso a los servicios de eVisor.

8.17.1 Parámetros Específicos a Cada Una de las Aplicaciones Configuradas en la Plataforma de eVisor:

- **secureMode:** Indicador de invocación del servicio web mediante un canal seguro. Los valores posibles son:

  - **true** → La invocación al servicio web se hará mediante un canal seguro.

  - **false** → La invocación al servicio web se hará mediante un canal no seguro.

- **endPoint:** IP y puerto donde se encuentran publicados los distintos WS. Sigue el formato **IP:Puerto** donde:

  - **IP** es la IP donde se encuentran publicados los distintos WS

- **Puerto** es el número del puerto donde se encuentran publicados los distintos WS

- **servicePath:** Ruta donde se encuentran publicados los distintos servicios web de @Firma.

- **callTimeout:** Tiempo de espera máximo, en milisegundos, para una solicitud de servicio.

- **authorizationMethod:** Método de autorización de ejecución de servicios Web empleado. Los valores posibles son:

  - **none** → No se utilizará ningún método de autorización.

  - **UsernameToken** → Se utilizará autorización mediante usuario y contraseña.

  - **BinarySecurity** → Se utilizará autorización mediante certificado digital.

- **authorizationMethod.user:** Nombre de usuario para el método de autorización **UsernameToken**, o alias del certificado (*.p12 o *.pfx) empleado para el método de autorización **BinarySecurity**.

- **authorizationMethod.password:** Contraseña del usuario para el método de autorización **UsernameToken**, o contraseña del certificado (*.p12 o *.pfx) empleado para el método de autorización **BinarySecurity**.

- **authorizationMethod.passwordType:** Modo en que se transmitirá la contraseña del usuario indicado en el método de autorización **UsernameToken**. Los valores posibles son:

  - **clear** → Se envía la contraseña en formato de texto plano.

  - **digest** → Se envía el resumen de la contraseña.

- **authorizationMethod.userKeystore:** Ruta al almacén de certificados donde se encuentra almacenado el certificado empleado para el método de autorización **BinarySecurity**.

- **authorizationMethod.userKeystorePassword:** Contraseña del almacén de certificados donde se encuentra almacenado el certificado empleado para el método de autorización **BinarySecurity**.

- **authorizationMethod.userKeystoreType:** Tipo del almacén de certificados donde se encuentra almacenado el certificado empleado para el método de autorización **BinarySecurity**. Los valores posibles son:

  - **JKS**

  - **PKCS12**

  - **JCEKS**

- **response.validate:** Indicador para validar la firma de la respuesta de los servicios de @Firma, en caso de encontrarse firmada. Los valores posibles son:

  - **true** → Se validará la firma de las respuestas de los servicios de @Firma que se encuentren firmadas.

  - **false** → No se validará la firma de las respuestas de los servicios de @Firma que se encuentren firmadas.

- **response.certificateAlias:** Alias empleado para identificar la clave pública del certificado usado en la firma de la respuesta de los servicios de @Firma. La clave pública estará almacenada en el almacén de claves indicado en la propiedad **authorizationMethod.userKeystore**.

8.18 Archivo hsm.properties

Este archivo define las propiedades asociadas al uso de almacenes de claves de tipo HSM. Es necesario para el uso de las clases definidas en el paquete **hsm**.

- **HSM_CONFIG_PATH:** Ruta absoluta al fichero de configuración PKCS11 donde se establece la ruta absoluta a la librería nativa del HSM entre otras configuraciones adicionales. Por ejemplo "/opt/users/HSM/config/pkcs11.cfg" indica la ruta al fichero "pkcs11.cfg" que contiene la configuración PKCS11 del HSM, cuyo contenido podría ser:

```
name = HSM_RealSec
library = /opt/users/HSM/lib/libcryptosec.so
description = Dispositivo HSM RealSec
disabledMechanism = {
CKM_SHA512
}
```

- **HSM_PASSWORD:** Contraseña de acceso al HSM. Esta propiedad podría estar vacía en caso de no ser necesaria dicha contraseña.

8.19 Archivo Language.properties

Este archivo permite configurar el idioma asociado a los mensajes de la API Integr@. Admite seleccionar los mensajes en inglés o en español.

- **LANGUAGE:** Idioma asociado a los mensajes de la API Integr@. Los valores posibles son:

  - **es_ES** → Idioma español.

  - **en_US** → Idioma inglés.

8.20 Archivo integra-log4j2.xml

Archivo de configuración de la API Log4j2.

8.21 Archivo parserParameters.properties

Fichero de propiedades que contiene los identificadores empleados como atajo o acceso directo a las rutas de los nodos extraídos en el procesado de las respuestas XML de los servicios de @Firma, TS@ y eVisor. Su uso es útil para el paquete **afirma5ServiceInvoker**.

Cada clave se corresponderá con el nombre del atajo o acceso directo al elemento, y el valor será la ruta xPath del nodo que obtener de la respuesta XML. Por ejemplo:

```
RFC3161Timestamp = dss:SignatureObject/dss:Timestamp/dss:RFC3161TimeStampToken
```

En este caso, el atajo con nombre **RFC3161Timestamp** se utilizaría para obtener, de una respuesta del *Servicio de Generación de Sello de Tiempo* de TS@ o del *Servicio de Renovación de Sello de Tiempo* de TS@, el valor del elemento **dss:RFC3161TimeStampToken** contenido dentro del elemento **dss:Timestamp**, que a su vez está contenido dentro del elemento **dss:SignatureObject**.

8.22 Archivo transformers.properties

Fichero de propiedades donde se definen las propiedades necesarias para la conversión de parámetros a XML en las peticiones a los WS de @Firma, TS@ y eVisor, así como las propiedades necesarias para el procesamiento de las respuestas de dichos WS.

8.22.1 Parámetros de Uso Común a los Servicios de @Firma, eVisor y TS@

- **TransformersTemplatesPath:** Ruta al directorio donde se encuentran las plantillas que contienen la información para generar las interfaces XML a generar y las plantillas que contienen la información para extraer los nodos de las respuestas XML y convertirlos a parámetros.

8.22.2 Parámetros Específicos a Cada Servicio de @Firma, eVisor y TS@

Cada propiedad tendrá el formato **NOMBRE_PETICION.NOMBRE_SERVICIO.VERSION_MENSAJE.NOMBRE_PARAMETRO** donde

- **NOMBRE_PETICION** es el nombre asociado a la petición y respuesta del servicio
- **NOMBRE_SERVICIO** es el nombre del servicio
- **VERSION_MENSAJE** es la versión del mensaje
- **NOMBRE_PARAMETRO** es el nombre del parámetro específico

Por ejemplo:

```
ValidarCertificado.ValidarCertificado.1_0.request.transformerClass=es.gob.afirma.t
ransformers.xmlTransformers.CommonXmlTransformer
```

Los parámetros específicos son:

- **request.transformerClass:** Nombre completo de la clase java que genera la interfaz XML para el servicio.
- **request.template:** Nombre de la plantilla XML empleada en la generación de la petición XML. Debe corresponderse con el nombre de alguno de los ficheros XML indicados en 8.11.
- **parser.transformerClass:** Nombre completo de la clase java que realizará el procesado de la respuesta XML.
- **parser.rootElement:** Ruta XPATH del elemento padre de la información relevante de la respuesta a partir del tag *<mensajeSalida>.*
- **parser.template:** Nombre de la plantilla XML empleada en el procesamiento de la respuesta XML. Debe corresponderse con el nombre de alguno de los ficheros XML indicados en 8.10.

8.23 Archivo staticTsl.properties

Archivo con configuraciones necesarias para la validación de certificados mediante una TSL. Este fichero de propiedades se encuentra incluido en la misma librería **Integra-tsl-2.2.3_001.jar.**

9 Integración

En función del tipo de integración que se realice se dispondrá de unos paquetes u otros, asi mismo, se dispondrá de forma parcial de alguno de los paquetes en función del tipo de integración que se realice.

9.1 Tipo de integración 1. Acceso a servicios OCSP y RFC 3161

9.1.1 Paquete es.gob.afirma.rfc3161TSAServiceInvoker

Este paquete contiene clases que gestionan la comunicación con los servicios RFC 3161 de TS@. Para poder hacer uso de los métodos definidos en este paquete debe estar configurado correctamente el archivo **tsaXXXXX.properties e integra.properties**. De este paquete se describirán aquellas clases más destacadas.

Clase RFC3161TSAServiceInvoker

Esta clase gestiona la obtención de sellos de tiempo a través de los servicios RFC 3161 de TS@. Está compuesta por los métodos:

```
public final byte[ ] generateTimeStampToken(String protocol, String
applicationParam, byte[ ] dataParam) throws TSAServiceInvokerException
```

Método que obtiene un sello de tiempo ASN.1 a través de los servicios RFC 3161 de TS@. En caso de que se produzca algún error durante el proceso, el método lanzará una excepción *TSAServiceInvokerException*. Como parámetros recibe:

- **protocol:** Parámetro que representa el protocolo a usar para la comunicación con TS@. Los valores permitidos son:
  - **TCP**
  - **HTTPS**
  - **SSL**
- **applicationParam:** Parámetro que representa el identificador de la aplicación cliente que lleva a cabo la invocación. El caso de que se desee atacar a una TSA que no requiere identificador de aplicación cliente, este parámetro se utilizará solo para identificar la configuración del acceso a TSA en el fichero de propiedades tsaServiceInvoker.properties, y no se enviará en la petición.
- **dataParam:** Parámetro que representa los datos a sellar.

9.1.2 Paquete es.gob.afirma.ocsp

Este paquete contiene todas las clases asociadas a la validación de certificados contra servidores OCSP. De este paquete se describirán aquellas clases más destacadas.

Clase OCSPEnhancedResponse

Clase que representa una respuesta OCSP, incluyendo la fecha en que la respuesta OCSP cacheada expira, siguiendo las recomendaciones indicadas en la RFC 5019. Está compuesta por los métodos:

```
public final Date getMaxAge()
```

Método que devuelve la fecha en que la respuesta OCSP cacheada expira en el servidor OCSP al que se ha realizado la petición de validación de certificado.

```
public final int getStatus()
```

Método que devuelve el estado asociado a la respuesta OCSP. Los valores que devuelven pueden ser:

- **0** → La respuesta de validación de certificado es correcta.
- **1** → La petición de validación de certificado es incorrecta.
- **2** → Se ha producido un error interno en el emisor.
- **3** → Intentar la validación del certificado más tarde.
- **5** → La petición de validación de certificado debe ir firmada.
- **6** → La petición no está autorizada.

```
public final Date getRevocationDate()
```

Método que devuelve la fecha de revocación del certificado, en caso de que se encuentre revocado.

```
public final String getErrorMsg()
```

Método que devuelve el mensaje de error en el caso de que la validación del certificado no haya sido correcta.

```
public final void setMaxAge(Date maxAgeParam)
```

Método que establece la fecha en que la respuesta OCSP cacheada expira en el servidor OCSP al que se ha realizado la petición de validación de certificado. Como parámetros recibe:

- **maxAgeParam:** Parámetro que representa la fecha en que la respuesta OCSP cacheada expira en el servidor OCSP al que se ha realizado la petición de validación de certificado.

```
public final void setStatus(int statusParam)
```

Método que establece el estado asociado a la respuesta OCSP. Como parámetros recibe:

- **statusParam:** Parámetro que representa el estado asociado a la respuesta OCSP. Los valores permitidos son:
  - **0** → La respuesta de validación de certificado es correcta.
  - **1** → La petición de validación de certificado es incorrecta.
  - **2** → Se ha producido un error interno en el emisor.
  - **3** → Intentar la validación del certificado más tarde.
  - **5** → La petición de validación de certificado debe ir firmada.
  - **6** → La petición no está autorizada.

```
public final void setRevocationDate(Date revocationDateParam)
```

Método que establece la fecha de revocación del certificado. Como parámetros recibe:

- **revocationDateParam:** Parámetro que representa la fecha de revocación del certificado.

```
public final void setErrorMsg(String errorMsgParam)
```

Método que establece el mensaje de error en el caso de que la validación del certificado no haya sido correcta. Como parámetros recibe:

- **errorMsgParam:** Parámetro que representa el mensaje de error en el caso de que la validación del certificado no haya sido correcta.

Clase OCSPClient

Esta clase representa un cliente OCSP conforme a las RFC 2560 y 5019. Está compuesta por los métodos:

```
public static OCSPEnhancedResponse validateCertificate(X509Certificate
certificateToValidate) throws OCSPClientException
```

Método que permite validar un certificado respecto a un servidor OCSP. Para poder llevar a cabo la validación es necesario que el archivo **integra.properties** esté correctamente configurado con los datos de configuración ocsp. En caso de que se produzca un error durante el proceso de validación el método lanzará una excepción *OCSPClientException*. Devuelve un objeto java que representa la información asociada a la respuesta OCSP, descrito en el punto 0.Como parámetros recibe:

- **certificateToValidate:** Parámetro que representa el certificado a validar.

9.1.3 Paquete es.gob.afirma.utils

Este paquete contiene clases que representan utilidades, así como interfaces con constantes. De este paquete se describirán aquellas clases más destacadas. Teniendo en cuenta que para este tipo de integración no funcionarán todas las clases y métodos del paquete es.gob.afirma.utils al no necesitarse inluir ciertas librerías de terceros innecesarias para este tipo de integración.

Clase Base64Coder

Esta clase define métodos que permiten la codificación y decodificación de datos en Base 64. Está compuesta por los métodos:

```
public static byte[ ] encodeBase64(byte[ ] data) throws TransformersException
```

Método que permite codificar en Base 64 una colección de bytes. Devuelve una colección de bytes codificados en Base 64. En caso de que se produzca algún error durante el proceso, el método lanzará una excepción *TransformersException*. Como parámetros recibe:

- **data:** Parámetro que representa la colección de bytes a codificar en Base 64.

```
public static byte[ ] encodeBase64(byte[ ] data, int offset, int len) throws
TransformersException
```

Método que permite codificar en Base 64 una colección de bytes, indicando la posición inicial y final de los bytes a codificar. Devuelve una colección de bytes codificados en Base 64. En caso de que se produzca algún error durante el proceso, el método lanzará una excepción *TransformersException*. Como parámetros recibe:

- **data:** Parámetro que representa la colección de bytes a codificar en Base 64.

- **offset:** Parámetro que representa la posición dentro de la colección de bytes para iniciar la codificación en Base 64.

- **len:** Parámetro que representa la posición dentro de la colección de bytes para finalizar la codificación en Base 64.

```
public static byte[ ] decodeBase64(byte[ ] data) throws TransformersException
```

Método que permite decodificar una colección de bytes codificados en Base 64. Devuelve una colección de bytes decodificados en Base 64. En caso de que se produzca algún error durante el proceso, el método lanzará una excepción *TransformersException*. Como parámetros recibe:

- **data:** Parámetro que representa la colección de bytes codificados en Base 64 a decodificar.

```
public static byte[ ] decodeBase64(byte[ ] data, int offset, int len) throws
TransformersException
```

Método que permite decodificar una colección de bytes codificados en Base 64, indicando la posición inicial y final de los bytes a decodificar. Devuelve una colección de bytes decodificada en Base 64. En caso de que se produzca algún error durante el proceso, el método lanzará una excepción *TransformersException*. Como parámetros recibe:

- **data:** Parámetro que representa la colección de bytes codificados en Base 64.

- **offset:** Parámetro que representa la posición dentro de la colección de bytes para iniciar la decodificación en Base 64.

- **len:** Parámetro que representa la posición dentro de la colección de bytes para finalizar la decodificación en Base 64.

```
public static boolean isBase64Encoded(byte[ ] data) throws TransformersException
```

Método que indica si una colección de bytes se encuentran codificados en Base 64. Devuelve un valor lógico que indica si la colección de bytes está codificada en Base 64 (verdadero) o no (falso). En caso de que se produzca algún error durante el proceso, el método lanzará una excepción *TransformersException*. Como parámetros recibe:

- **data:** Parámetro que representa la colección de bytes a comprobar.

```
public static String encodeBase64(String data) throws TransformersException
```

Método que permite codificar en Base 64 una cadena de texto. Devuelve una cadena de texto codificada en Base 64. En caso de que se produzca algún error durante el proceso, el método lanzará una excepción *TransformersException.* Como parámetros recibe:

- **data:** Parámetro que representa la cadena de texto a codificar en Base 64.

```
public static String decodeBase64(String data) throws TransformersException
```

Método que permite decodificar una cadena de texto codificada en Base 64. Devuelve una cadena de texto decodificada en Base 64. En caso de que se produzca algún error durante el proceso, el método lanzará una excepción *TransformersException.* Como parámetros recibe:

- **data:** Parámetro que representa la cadena de texto codificada en Base 64.

Clase CryptoUtil

Esta clase define métodos asociados a operaciones criptográficas de cálculo de resumen de datos o con funciones de hash. Está compuesta por los métodos:

```
public static byte[ ] digest(String algorithm, byte[ ] data) throws
SigningException
```

Método que permite calcular el resumen de un conjunto de datos a partir de un determinado algoritmo de hash. Devuelve una colección de bytes que se corresponde con el resumen calculado. En caso de que no sea posible llevar a cabo la operación se devolverá un valor nulo. Si la implementación asociada al algoritmo de resumen indicado como parámetro no existe para el proveedor criptográfico usado, el método lanzará una excepción *SigningException*. Como parámetros recibe:

- **algorithm:** Parámetro que representa el nombre del algoritmo de hash.
- **data:** Parámetro que representa la colección de datos que procesar.

```
public static AlgorithmIdentifier getAlgorithmIdentifierByName(String
hashAlgorithm)
```

Método que obtiene el OID de un algoritmo de hash a partir de su nombre. En caso de que no sea posible obtener el OID se devolverá un valor nulo. Como parámetros recibe:

- **hashAlgorithm:** Parámetro que representa el nombre del algoritmo de hash.

```
public static String getDigestAlgorithmName(final String pseudoName)
```

Método que obtiene el nombre de un algoritmo de hash a partir de su alias. Por ejemplo, si el alias del algoritmo de hash fuera “sHa1” el método devolvería como nombre del algoritmo “SHA-1”. En

caso de que no sea posible obtener el nombre del algoritmo se devolverá un valor nulo. Como parámetros recibe:

- **pseudoName:** Parámetro que representa el alias del algoritmo de hash.

Clase GenericUtils

Esta clase contiene utilidades de uso genérico. Está compuesta por los métodos:

```
public static boolean assertStringValue(String value)
```

Método que comprueba si una cadena de texto no es vacía ni nula. Devuelve un valor lógico que indica si la cadena de texto no es vacía ni nula (verdadero) o por el contrario es vacía o nula (falso). Como parámetros recibe:

- **value:** Parámetro que representa la cadena de texto a comprobar.

```
public static boolean assertArrayValid(byte[ ] data)
```

Método que comprueba si una colección de bytes no es vacía ni nula. Devuelve un valor lógico que indica si la colección de bytes no es vacía ni nula (verdadero) o por el contrario es vacía o nula (falso). Como parámetros recibe:

- **data:** Parámetro que representa la colección de bytes a comprobar.

```
public static String getValueFromMapsTree(String path, Map<String, Object>
treeValues)
```

Método que obtiene un valor concreto de un árbol de mapas a partir de una ruta indicada. Devuelve una cadena de texto que se corresponde con el valor buscado. Como parámetros recibe:

- **path:** Parámetro que representa la ruta en el árbol de mapas, utilizando como separador el carácter '/'.

- **treeValues:** Parámetro que representa el árbol de mapas que procesar.

```
public static byte[ ] getDataFromInputStream(final InputStream input) throws
IOException
```

Método que obtiene una colección de bytes a partir de una secuencia de datos. Devuelve la colección de bytes leídos de la secuencia de datos de entrada. En caso de que se produjese algún error durante el proceso, el método lanzará una excepción *IOException*. Como parámetros recibe:

- **input:** Parámetro que representa la secuencia de datos de entrada que procesar.

```
public static boolean checkNullValues(Object... values)
```

Método que comprueba si ninguno de los valores de un conjunto es nulo. Devuelve un valor lógico que indica si ningún valor es nulo (verdadero) o bien alguno es nulo (falso). Como parámetros recibe:

- **values:** Parámetro que representa el conjunto de valores a procesar.

```
public static void printResult(byte[ ] result, Logger logger)
```

Método que codifica en Base 64 y escribe en el log un conjunto de datos. Como parámetros recibe:

- **result:** Parámetro que representa el conjunto de datos que codificar en Base 64 y escribir en el log.
- **logger:** Parámetro que representa el elemento que gestiona el log.

Clase UtilsCertificate

Esta clase contiene define utilidades asociadas a la gestión de certificados. Está compuesta por los métodos:

```
public static String canonicalizeX500Principal(String x500PrincipalName)
```

Método que canonicaliza un elemento X.500 Principal de un certificado. Devuelve una cadena de texto que se corresponde con el elemento canonicalizado. Como parámetros recibe:

- **x500PrincipalName:** Parámetro que representa un elemento X.500 Principal de un certificado.

```
public static X509Certificate generateCertificate(byte[ ] certificateBytes) throws
CertificateException
```

Método que genera un certificado a partir de una colección de bytes. Devuelve un objeto X.509 que representa el certificado. En caso de que no sea posible obtener el certificado a partir de la colección de bytes, el método lanzará una excepción *CertificateException*. Como parámetros recibe:

- **certificateBytes:** Parámetro que representa el certificado.

```
public static boolean equals(X509Certificate cert1, X509Certificate cert2)
```

Método que compara la clave pública, el emisor y el número de serie de dos certificados. Devuelve un valor lógico que indica si ambos certificados son iguales (verdadero) o no (falso). Como parámetros recibe:

- **cert1:** Parámetro que representa el primer certificado a comparar.
- **cert2:** Parámetro que representa el segundo certificado a comparar.

Clase UtilsFileSystem

Esta clase contiene utilidades de uso genérico. Está compuesta por los métodos:

```
public static synchronized String readFileBase64Encoded(String path, boolean
isRelativePath)
```

Método que lee un archivo y lo codifica en Base 64. Devuelve una cadena de texto codificada en Base 64 que representa el contenido del archivo. Como parámetros recibe:

- **path:** Parámetro que representa la ruta al fichero. Puede ser una ruta completa o relativa.
- **isRelativePath:** Parámetro que indica si la ruta al fichero es relativa (verdadero) o completa (falso).

```
public static synchronized byte[ ] getArrayByteFileBase64Encoded(String path,
boolean isRelativePath)
```

Método que lee un archivo y lo codifica en Base 64. Devuelve una colección de bytes codificada en Base 64 que representa el contenido del archivo. Como parámetros recibe:

- **path:** Parámetro que representa la ruta al fichero. Puede ser una ruta completa o relativa.
- **isRelativePath:** Parámetro que indica si la ruta al fichero es relativa (verdadero) o completa (falso).

```
public static synchronized byte[ ] readFile(String path, boolean isRelativePath)
```

Método que lee un archivo. Devuelve una colección de bytes que representa el contenido del archivo. Como parámetros recibe:

- **path:** Parámetro que representa la ruta al fichero. Puede ser una ruta completa o relativa.

- **isRelativePath:** Parámetro que indica si la ruta al fichero es relativa (verdadero) o completa (falso).

```
public static void writeFile(byte[ ] data, String filename) throws IOException
```

Método que genera un archivo a partir de un conjunto de datos. En caso de producirse algún error durante el proceso, el método lanzará una excepción *IOException*. Como parámetros recibe:

- **data:** Parámetro que representa el conjunto de datos que se corresponden con el archivo.
- **filename:** Parámetro que representa la ruta completa al archivo a generar.

Clase UtilsKeystore

Esta clase contiene utilidades que permiten el manejo de almacenes de claves. Está compuesta por los métodos:

```
public static KeyStore loadKeystore(String path, String password, String type)
throws KeyStoreException, NoSuchAlgorithmException, CertificateException,
IOException
```

Método que lee un almacén de claves. En caso de que el tipo de almacén de claves a instanciar no esté reconocido, el método lanzará una excepción *KeyStoreException*. En caso de que el algoritmo usado para comprobar la integración del almacén de claves no esté reconocido, el método lanzará una excepción *NoSuchAlgorithmException*. En caso de que alguno de los certificados del almacén de claves no pueda ser cargado, el método lanzará una excepción *CertificateException*. En caso de que se produzca algún error de lectura del almacén de claves, el método lanzará una excepción *IOException*. Devuelve un objeto java que representa el almacén de claves. Como parámetros recibe:

- **path:** Parámetro que representa la ruta completa al almacén de claves.
- **password:** Parámetro que representa la contraseña para acceder al almacén de claves.
- **type:** Parámetro que representa el tipo del almacén de claves.

```
public static byte[ ] getCertificateEntry(byte[ ] keystore, String
keystoreDecodedPass, String alias, String keystoreType) throws KeyStoreException,
NoSuchAlgorithmException, CertificateException, IOException
```

Método que obtiene un certificado almacenado en un almacén de claves. Devuelve una colección de bytes que se corresponden con el objeto X.509 del certificado. En caso de que el tipo de almacén de claves a instanciar no esté reconocido, el método lanzará una excepción *KeyStoreException*. En caso de que el algoritmo usado para comprobar la integración del almacén de claves no esté reconocido, el método lanzará una excepción *NoSuchAlgorithmException*. En caso de que alguno de los

certificados del almacén de claves no pueda ser cargado, el método lanzará una excepción *CertificateException*. En caso de que se produzca algún error de lectura del almacén de claves, el método lanzará una excepción *IOException.* Como parámetros recibe:

- **keystore:** Parámetro que representa la colección de bytes que se corresponden con el almacén de claves.
- **keystoreDecodedPass:** Parámetro que representa la contraseña para acceder al almacén de claves.
- **alias:** Parámetro que representa el alias del certificado a obtener.
- **keystoreType:** Parámetro que representa el tipo del almacén de claves.

```
public static PrivateKey getPrivateKeyEntry(byte[ ] keystore, String
keystoreDecodedPass, String alias, String keystoreType, String
privateKeyDecodedPass) throws KeyStoreException, NoSuchAlgorithmException,
CertificateException, IOException, UnrecoverableKeyException
```

Método que obtiene una clave privada almacenada en un almacén de claves. Devuelve un objeto java que representa la clave privada. En caso de que el tipo de almacén de claves a instanciar no esté reconocido, el método lanzará una excepción *KeyStoreException*. En caso de que el algoritmo usado para comprobar la integración del almacén de claves no esté reconocido, el método lanzará una excepción *NoSuchAlgorithmException*. En caso de que alguno de los certificados del almacén de claves no pueda ser cargado, el método lanzará una excepción *CertificateException*. En caso de que se produzca algún error de lectura del almacén de claves, el método lanzará una excepción *IOException*. En caso de que la clave privada no pueda ser recuperada, el método lanzará una excepción *UnrecoverableKeyException*. Como parámetros recibe:

- **keystore:** Parámetro que representa la colección de bytes que se corresponden con el almacén de claves.
- **keystoreDecodedPass:** Parámetro que representa la contraseña para acceder al almacén de claves.
- **alias:** Parámetro que representa el alias de la clave privada a obtener.
- **keystoreType:** Parámetro que representa el tipo del almacén de claves.
- **privateKeyDecodedPass:** Parámetro que representa la contraseña para acceder a la clave privada.

```
public static List<X509Certificate> getListCertificates(byte[ ] keystore, String
keystoreDecodedPass, String keystoreType) throws KeyStoreException,
NoSuchAlgorithmException, CertificateException, IOException
```

Método que obtiene una lista con todos los certificados almacenados en un almacén de claves. En caso de que el tipo de almacén de claves a instanciar no esté reconocido, el método lanzará una excepción *KeyStoreException*. En caso de que el algoritmo usado para comprobar la integración del almacén de claves no esté reconocido, el método lanzará una excepción *NoSuchAlgorithmException*. En caso de que alguno de los certificados del almacén de claves no pueda ser cargado, el método lanzará una excepción *CertificateException*. En caso de que se produzca algún error de lectura del almacén de claves, el método lanzará una excepción *IOException*. Como parámetros recibe:

- **keystore:** Parámetro que representa la colección de bytes que se corresponden con el almacén de claves.
- **keystoreDecodedPass:** Parámetro que representa la contraseña para acceder al almacén de claves.
- **keystoreType:** Parámetro que representa el tipo del almacén de claves.

Clase UtilsResources

Esta clase proporciona funcionalidades para controlar el cierre de recursos. Está compuesta por los métodos:

```
public static void safeCloseInputStream(InputStream is)
```

Método que gestiona el cierre de un flujo de entrada. Como parámetros recibe:

- **is:** Parámetro que representa el flujo de entrada.

```
public static void safeCloseOutputStream(OutputStream os)
```

Método que gestiona el cierre de un flujo de salida. Como parámetros recibe:

- **os:** Parámetro que representa el flujo de salida.

```
public static void safeCloseSocket(Socket socket)
```

Método que gestiona el cierre de un *socket*. Como parámetros recibe:

- **socket:** Parámetro que representa el *socket*.

```
public static void safeClosePDFStamper(PdfStamper stamper)
```

Método que gestiona el cierre de un objeto que permite modificar un documento PDF. Como parámetros recibe:

- **stamper:** Parámetro que permite modificar un documento PDF.

Clase UtilsSignature

Esta clase proporciona funcionalidades criptográficas relacionadas con firmas electrónicas. Está compuesta por los métodos:

```
public static void validateCertificate(X509Certificate certificate, Date
validationDate, boolean isUpgradeOperation) throws SigningException
```

Método que valida el periodo de validez y el estado de revocación de un certificado. Si el certificado no es válido o se ha producido algún error durante la validación, se lanzará una excepción *SigningException.* Como parámetros recibe:

- **certificate:** Parámetro que representa el certificado a validar.
- **validationDate:** Paramétro que indica la fecha de validación.
- **isUpgradeOperation:** Bandera que indica si la operación original es de actualización o no.

```
public static PDFSignatureDictionary obtainLatestSignatureFromPDF(PdfReader
reader)
```

Método que recupera el diccionario de firma con el número de revisión mayor en una firma de tipo PAdES. Los parámetros recibidos son:

- **reader:** Parámetro que representa el objeto Reader del documento PDF.

```
public static boolean isNotPAdESEnhancedPDF(PdfDictionary pdfDic)
```

Método que indica si un diccionario de firma hace referencia a una firma PDF/PAdES-Basic o a una firma PAdES-BES/PAdES-EPES. Los parámetros recibidos son:

- **pdfDic:** Parámetro que representa el diccionario de firma.

```
public static CMSSignedData getCMSSignature(PDFSignatureDictionary
signatureDictionary) throws SigningException
```

Método que recupera el elemento signedData contenido en un diccionario de firma de un documento PDF. Si se produce algún error se lanzará una excepción *SigningException.* Los parámetros recibidos son:

- **signatureDictionary:** Parámetro que representa el diccionario de firma.

```
public static boolean isImplicit(CMSSignedData cmsSignedData)
```

Método que comprueba si la firma incluye el documento original o no.

- **cmsSignedData:** Parámetro que representa el mensaje de tipo pkcs7-signature.

```
public static boolean equalsHash(PdfArray pdfArrayByteRange, MessageDigest
messageDigestSignature, byte[ ] pdfDocument, byte[ ] hashSignature)
```

Método que compara dos bytes de arrays y comprueba si el hash de cada uno de ellos son iguales o no. Como parámetros recibe:

- **pdfArrayByteRange:** Array de bytes que representa los datos originales sobre los que se calculará el hash.
- **messageDigestSignature:** Algoritmo de resumen a utilizar.
- **pdfDocument:** Conjunto de bytes que representa el primer hash a comparar.
- **hashSignature:** Conjunto de bytes que representa el segundo hash a comparar.

```
public static void checkSubFilterConditionsISO320001(PDFSignatureDictionary
dictionarySignature, CMSSignedData signedData)
```

Método que comprueba que se cumpla la condición especificada en la sección 12.8.3.3.1 de la ISO 32000-1:

adbe.pkcs7.detached: The original signed message digest over the document's byte range shall be incorporated as the normal PKCS#7 SignedData field. No data shall be encapsulated in the PKCS#7 SignedData field.

adbe.pkcs7.sha1: The SHA1 digest of the document's byte range shall be encapsulated in the PKCS#7 SignedData field with ContentInfo of type Data. The digest of that SignedData shall be incorporated as the normal PKCS#7 digest.

Como parámetros recibe:

- **dictionarySignature:** Parámetro que representa el diccionario de firma.
- **signedData:** Parámetro que representa los datos firmados.

```
public static void validatePAdESEnhancedMandatoryAttributes(PDFSignatureDictionary
signatureDictionary, CMSSignedData signedData) throws SigningException
```

Método que valida los atributos obligatorios de una firma *PAdES enhanced*. Si se produce algún error durante el proceso el método lanzará una excepción *SigningException*. Como parámetros recibe:

- **signatureDictionary:** Parámetro que representa el diccionario de firma.
- **signedData:** Parámetro que representa los datos firmados.

```
public static void validatePAdESOptionalAttributes(PDFSignatureDictionary
signatureDictionary, CMSSignedData signedData, boolean isEPES, boolean isBasic)
throws SigningException
```

Método que valida los atributos opcionales de una firma PAdES. Si se produce algún error durante el proceso el método lanzará una excepción *SigningException*. Como parámetros recibe:

- **signatureDictionary:** Parámetro que representa el diccionario de firma.
- **signedData:** Parámetro que representa los datos firmados.
- **isEPES:** Bandera que indica si la firma es de tipo PAdES-EPES o PAdES-BES.
- **isBasic:** Bandera que indica si la firma es de tipo PAdES-Basic o PAdES enhanced.

```
public static X509CertificateHolder getX509CertificateHolderBySignerId(Store
certificatesStore, SignerId signerId)
```

Método que obtiene la estructura de un certificado almacenado. Como parámetros recibe:

- **certificatesStore:** Parámetro que representa el certificado almacenado.
- **signerId:** Parámetro que representa el identificador del firmante usado para buscar el certificado.

```
public static void validatePDFSigner(CMSSignedData signedData, SignerInformation
signerInformation, PdfDictionary pdfSignatureDictionary, Date validationDate)
throws SigningException
```

Método que valida la firma contenida en un documento PDF. Si se produce algún error durante el proceso el método lanzará una excepción *SigningException*. Como parámetros recibe:

- **signedData:** Parámetro que representa los datos firmados.
- **signerInformation:** Parámetro que representa la información del firmante.

- **pdfSignatureDictionary:** Parámetro que representa el diccionario de firma PDF.
- **validationDate:** Parámetro que representa la fecha de validación.

```
public static List<XAdESSignerInfo> getXAdESListSigners(Document doc)
```

Método que obtiene una lista con la principal información relativa a los firmantes de una firma XAdES. Como parámetros recibe:

- **doc:** Parámetro que representa el documento XML.

```
public static List<CAdESSignerInfo> getCAdESListSigners(CMSSignedData signedData)
```

Método que obtiene una lista con la principal información relativa a los firmantes de una firma CAdES. Como parámetros recibe:

- **signedData:** Parámetro que representa los datos firmados.

```
public static void
validateXAdESSigner(org.apache.xml.security.signature.XMLSignature xmlSignature,
X509Certificate signingCertificate, TimeStampToken tst, Element xmlTst, String
signingMode, byte[ ] signedFile, String signedFileName) throws SigningException
```

Método que valida el firmante de una firma XAdES. Si se produce algún error durante el proceso el método lanzará una excepción *SigningException*. Como parámetros recibe:

- **xmlSignature:** Parámetro que representa la firma XML.
- **signingCertificate**: Parámetro que representa el certificado firmante.
- **tst**: Parámetro que representa el sello de tiempo RFC3161 asociado al firmante.
- **xmlTst**: Parámetro que representa el sello de tiempo XML asociado al firmante.
- **signingMode**: Parámetro que representa el modo en el que se ha realizado la firma XAdES (detached, enveloped o enveloping).
- **signedFile**: Parámetro que representa el documento firmado cuando éstos no están incluidos en el XML.
- **signedFileName**: Parámetro que representa el nombre del documento firmado cuando éstos no están incluidos en el XML.

```
public static X509Certificate getSigningCertificate(CMSSignedData signedData,
SignerInformation signerInformation) throws SigningException
```

Método que obtiene el certificado firmante de un firmante incluido dentro de una forma. Si se produce algún error durante el proceso el método lanzará una excepción *SigningException*. Como parámetros recibe:

- **signedData:** Parámetro que representa los datos firmados.
- **signerInformation**: Parámetro que representa la información relativa al firmante sobre el que se quiere obtener el certificado.

```
public static Document getDocumentFromXML(byte[ ] xmlDocument) throws
SigningException
```

Método que obtiene un objeto java como representación de un documento XML. Si se produce algún error durante el proceso el método lanzará una excepción *SigningException*. Como parámetros recibe:

- **xmlDocument:** Parámetro que representa el documento XML.

```
public static PdfReader obtainLatestRevision(PdfReader reader) throws
SigningException
```

Método que obtiene el diccionario de firma con el número de revisión más reciente. Si se produce algún error durante el proceso el método lanzará una excepción SigningException. Como parámetros recibe:

- **reader:** Parámetro que representa el *reader* del documento PDF.

```
public static void checkPDFCertificationLevel(Map<Integer, InputStream>
mapRevisions) throws SigningException
```

Método que comprueba si se ha añadido alguna firma al documento PDF tras haber sido certificado. Si se produce algún error durante el proceso el método lanzará una excepción SigningException. Como parámetros recibe:

- **mapRevisions:** Parámetro que representa el conjunto de revisiones del documento PDF. Cada revisión representa un diccionario de firma.

Clase UtilsTimestamp

Esta clase proporciona funcionalidades criptográficas relacionadas con sellos de tiempo. Está compuesta por los métodos:

```
public static TimeStampToken getTimestampFromRFC3161Service(byte[ ] dataToStamp,
String applicationID, String tsaCommunicationMode) throws SigningException
```

Método que obtiene un sello de tiempo ASN.1 del servicio RFC 3161 de TS@. Para poder obtener el sello de tiempo de TS@ será necesario que el fichero **tsaXXXXX.properties e integra.properties** estén configurados correctamente respecto a las propiedades asociadas a la comunicación con TS@. En caso de que se produzca un error durante el proceso y no sea posible obtener el sello de tiempo el método lanzará una excepción *SigningException*. Como parámetros recibe:

- **dataToStamp:** Parámetro que representa los datos a sellar.
- **applicationID:** Parámetro que representa el identificador de aplicación cliente para la comunicación con TS@.
- **tsaCommunicationMode:** Parámetro que representa el protocolo definido para la comunicación con TS@. Los valores permitidos son:
  - **RFC3161-TCP** → Comunicación TCP.
  - **RFC3161-HTTPS** → Comunicación HTTPS.
  - **RFC3161-SSL** → Comunicación SSL.

```
public static X509Certificate getSigningCertificate(TimeStampToken tst) throws
SigningException
```

Método que obtiene el certificado firmante de un sello de tiempo ASN.1. En caso de que se produzca un error durante el proceso o no sea posible obtener el certificado el método lanzará una excepción *SigningException*. Como parámetros recibe:

- **tst:** Parámetro que representa el sello de tiempo ASN.1.

```
public static void validateASN1Timestamp(TimeStampToken tst) throws
SigningException
```

Método que valida estructuralmente un sello de tiempo ASN.1 (no valida el certificado firmante). En caso de que se produzca un error durante el proceso o si el sello de tiempo no es válido, el método lanzará una excepción *SigningException*. Como parámetros recibe:

- **tst:** Parámetro que representa el sello de tiempo ASN.1 que validar.

```
public static TimeStampToken getTimeStampToken(SignerInformation
signerInformation) throws SigningException
```

Método que obtiene un sello de tiempo ASN.1 de la información asociada al firmante de una firma, si es que ésta contiene un sello de tiempo. Si no contuviese un sello de tiempo se devolvería un valor nulo. En caso de que el sello de tiempo se encuentre mal formado el método lanzará una excepción *SigningException*. Como parámetros recibe:

- **signerInformation:** Parámetro que representa la información asociada al firmante.

```
public static Date getGenTimeXMLTimestamp(Element xmlTimestamp) throws
SigningException
```

Método que obtiene la fecha de generación de un sello de tiempo XML. En caso de que la fecha de generación del sello de tiempo no tenga un formato UTC el método lanzará una excepción *SigningException*. Como parámetros recibe:

- **xmlTimestamp:** Parámetro que representa el sello de tiempo XML.

9.1.4 Paquete es.gob.afirma.hsm

Este paquete contiene clases que gestionan el manejo de dispositivos HSM. Para poder hacer uso de los métodos definidos en este paquete debe estar configurado correctamente el archivo **hsm.properties**. De este paquete se describirán aquellas clases más destacadas.

Clase HSMKeystore

Esta clase maneja todas las operaciones relacionadas con almacenes de claves HSM. Está compuesta por los métodos:

```
public static PrivateKey getPrivateKey(String alias) throws HSMException
```

Método que obtiene una clave privada almacenada en un HSM a partir de su alias. En caso de que se produzca algún error durante el proceso, el método lanzará una excepción *HSMException*. Como parámetros recibe:

- **alias:** Parámetro que representa el alias de la clave privada a obtener.

```
public static X509Certificate getCertificate(String alias) throws HSMException
```

Método que obtiene un certificado almacenado en un HSM a partir de su alias. En caso de que se produzca algún error durante el proceso, el método lanzará una excepción *HSMException*. Como parámetros recibe:

- **alias:** Parámetro que representa el alias del certificado a obtener.

```
public static PrivateKeyEntry getPrivateKeyEntry(String alias) throws HSMException
```

Método que obtiene una entrada asociada a una clave privada almacenado en un HSM a partir de su alias. En caso de que se produzca algún error durante el proceso, el método lanzará una excepción *HSMException*. Como parámetros recibe:

- **alias:** Parámetro que representa el alias de la entrada a obtener.

9.2 Tipo de integración 2. Acceso a servicios WS y DSS

9.2.1 Paquete es.gob.afirma.afirma5ServiceInvoker

Este paquete contiene clases que permiten la comunicación con los WS de @Firma y eVisor. Para poder hacer uso de los métodos definidos en este paquete deben estar configurados correctamente los archivos **afirmaXXXXX.properties e integra.properties**. De este paquete se describirán aquellas clases más destacadas.

Clase Afirma5ServiceInvokerFacade

Esta clase implementa el patrón fachada y permite independizar la invocación de los servicios publicados en las plataformas de @Firma y eVisor. Está compuesta por los métodos:

```
public String invokeService(String xmlInput, String service, String method, String
applicationName) throws Afirma5ServiceInvokerException
```

Método que permite invocar a servicios de @Firma y eVisor. Devuelve una cadena de texto en formato XML con la respuesta ofrecida por el servicio. En caso de que se produzca algún error durante el proceso, el método lanzará una excepción *Afirma5ServiceInvokerException*. Como parámetros recibe:

- **xmlInput:** Parámetro que representa la petición en formato XML.
- **service:** Parámetro que representa el nombre del servicio que invocar. Los valores permitidos son:
  - **AlmacenarDocumento** → Servicio de Almacenamiento de Documentos de @Firma.
  - **StoreDocument** → Servicio de Almacenar Documento, en inglés, de @Firma.

- **ValidarCertificado** → Servicio de Validación de Certificados de @Firma.
- **ValidateCertificate** → Servicio de Validación de Certificados, en inglés, de @Firma.
- **ObtenerInfoCertificado** → Servicio de Obtención de Información de Certificado de @Firma.
- **GetInfoCertificate** → Servicio de Obtención de Información de Certificado, en inglés, de @Firma.
- **ValidarFirma** → Servicio de Validación de Firmas de @Firma.
- **SignatureValidation** → Servicio de Validación de Firmas, en inglés, de @Firma.
- **ServerSignature** → Servicio de Firma Servidor, en inglés, de @Firma.
- **FirmaServidor** → Servicio de Firma Servidor de @Firma.
- **FirmaServidorCoSign** → Servicio de Firma Servidor CoSign de @Firma.
- **ServerSignatureCoSign** → Servicio de Firma Servidor CoSign, en inglés, de @Firma.
- **FirmaServidorCounterSign** → Servicio de Firma Servidor CounterSign de @Firma.
- **ServerSignatureCounterSign** → Servicio de Firma Servidor CounterSign, en inglés, de @Firma.
- **DSSAfirmaSign** → Servicios de Firma Delegada de @Firma.
- **ThreePhaseUserSignatureF1** → Servicio Fase 1 de Firma Usuario 3 Fases, en inglés, de @Firma.
- **FirmaUsuario3FasesF1** → Servicio Fase 1 de Firma Usuario 3 Fases de @Firma.
- **ThreePhaseUserSignatureF1CoSign** → Servicio Fase 1 de Firma Usuario 3 Fases CoSign, en inglés, de @Firma.
- **FirmaUsuario3FasesF1CoSign** → Servicio Fase 1 de Firma Usuario 3 Fases CoSign de @Firma.
- **ThreePhaseUserSignatureF1CounterSign** → Servicio Fase 1 de Firma Usuario 3 Fases CounterSign, en inglés, de @Firma.
- **FirmaUsuario3FasesF1CounterSign** → Servicio Fase 1 de Firma Usuario 3 Fases CounterSign de @Firma.

- **ThreePhaseUserSignatureF3** → Servicio Fase 3 de Firma Usuario 3 Fases, en inglés, de @Firma.
- **FirmaUsuario3FasesF3** → Servicio Fase 3 de Firma Usuario 3 Fases de @Firma.
- **TwoPhaseUserSignatureF2** → Servicio Fase 2 de Firma Usuario 2 Fases, en inglés, de @Firma.
- **FirmaUsuario2FasesF2** → Servicio Fase 2 de Firma Usuario 2 Fases de @Firma.
- **DSSAfirmaVerify** → Servicio de Validación y Actualización de Firmas de @Firma.
- **DSSAfirmaVerifyCertificate** → Servicio de Validación de Certificados de @Firma.
- **DSSBatchVerifyCertificate** → Servicio de Validación de Certificados por Lote de @Firma.
- **DSSBatchVerifySignature** → Servicio de Validación de Firmas por Lote de @Firma.
- **DSSAsyncRequestStatus** → Servicio de Consulta de Peticiones Asíncronas de @Firma.
- **SignatureReportService** → Servicios de Validación y Generación de Informes de eVisor.

- **method:** Parámetro que representa el nombre del método asociado al servicio que invocar.
- **applicationName:** Parámetro que representa el identificador de la aplicación cliente que lleva a cabo la petición.

```
public String invokeService(String xmlInput, String service, String method,
Properties serviceProperties) throws Afirma5ServiceInvokerException
```

Método que permite invocar a servicios de @Firma y eVisor en base a un conjunto de propiedades ajenas a las definidas para Integr@. Devuelve una cadena de texto en formato XML con la respuesta ofrecida por el servicio. En caso de que se produzca algún error durante el proceso, el método lanzará una excepción *Afirma5ServiceInvokerException*. Como parámetros recibe:

- **xmlInput:** Parámetro que representa la petición en formato XML.
- **service:** Parámetro que representa el nombre del servicio que invocar. Los valores permitidos son:
  - **AlmacenarDocumento** → Servicio de Almacenamiento de Documentos de @Firma.

- **StoreDocument** → Servicio de Almacenar Documento, en inglés, de @Firma.
- **ValidarCertificado** → Servicio de Validación de Certificados de @Firma.
- **ValidateCertificate** → Servicio de Validación de Certificados, en inglés, de @Firma.
- **ObtenerInfoCertificado** → Servicio de Obtención de Información de Certificado de @Firma.
- **GetInfoCertificate** → Servicio de Obtención de Información de Certificado, en inglés, de @Firma.
- **ValidarFirma** → Servicio de Validación de Firmas de @Firma.
- **SignatureValidation** → Servicio de Validación de Firmas, en inglés, de @Firma.
- **ServerSignature** → Servicio de Firma Servidor, en inglés, de @Firma.
- **FirmaServidor** → Servicio de Firma Servidor de @Firma.
- **FirmaServidorCoSign** → Servicio de Firma Servidor CoSign de @Firma.
- **ServerSignatureCoSign** → Servicio de Firma Servidor CoSign, en inglés, de @Firma.
- **FirmaServidorCounterSign** → Servicio de Firma Servidor CounterSign de @Firma.
- **ServerSignatureCounterSign** → Servicio de Firma Servidor CounterSign, en inglés, de @Firma.
- **DSSAfirmaSign** → Servicios de Firma Delegada de @Firma.
- **ThreePhaseUserSignatureF1** → Servicio Fase 1 de Firma Usuario 3 Fases, en inglés, de @Firma.
- **FirmaUsuario3FasesF1** → Servicio Fase 1 de Firma Usuario 3 Fases de @Firma.
- **ThreePhaseUserSignatureF1CoSign** → Servicio Fase 1 de Firma Usuario 3 Fases CoSign, en inglés, de @Firma.
- **FirmaUsuario3FasesF1CoSign** → Servicio Fase 1 de Firma Usuario 3 Fases CoSign de @Firma.
- **ThreePhaseUserSignatureF1CounterSign** → Servicio Fase 1 de Firma Usuario 3 Fases CounterSign, en inglés, de @Firma.

- **FirmaUsuario3FasesF1CounterSign** → Servicio Fase 1 de Firma Usuario 3 Fases CounterSign de @Firma.
- **ThreePhaseUserSignatureF3** → Servicio Fase 3 de Firma Usuario 3 Fases, en inglés, de @Firma.
- **FirmaUsuario3FasesF3** → Servicio Fase 3 de Firma Usuario 3 Fases de @Firma.
- **TwoPhaseUserSignatureF2** → Servicio Fase 2 de Firma Usuario 2 Fases, en inglés, de @Firma.
- **FirmaUsuario2FasesF2** → Servicio Fase 2 de Firma Usuario 2 Fases de @Firma.
- **DSSAfirmaVerify** → Servicio de Validación y Actualización de Firmas de @Firma.
- **DSSAfirmaVerifyCertificate** → Servicio de Validación de Certificados de @Firma.
- **DSSBatchVerifyCertificate** → Servicio de Validación de Certificados por Lote de @Firma.
- **DSSBatchVerifySignature** → Servicio de Validación de Firmas por Lote de @Firma.
- **DSSAsyncRequestStatus** → Servicio de Consulta de Peticiones Asíncronas de @Firma.
- **SignatureReportService** → Servicios de Validación y Generación de Informes de eVisor.

- **method:** Parámetro que representa el nombre del método asociado al servicio que invocar.
- **serviceProperties:** Parámetro que representa el conjunto de propiedades configuradas para llevar a cabo la petición al WS.

9.2.2 Paquete es.gob.afirma.transformers

Este paquete proporciona clases relacionadas con el procesamiento de peticiones y respuestas asociadas a los WS de @Firma, TS@ y eVisor. De este paquete se describirán aquellas clases más destacadas.

Clase TransformersFacade

Esta clase proporciona métodos para transformar parámetros de entrada y salida relacionados con los WS de @Firma, TS@ y eVisor. Para que el funcionamiento de la clase sea correcto el archivo **transformers.properties** debe estar correctamente configurado. La clase está compuesta por los métodos:

```
public static TransformersFacade getInstance()
```

Método que obtiene una instancia de la clase.

```
public Document getXmlRequestFileByRequestType(String serviceReq, String method,
String type, String version) throws TransformersException
```

Método que genera una estructura XML a partir de uno de los documentos XML definidos en el punto 8.1.2. En caso de que se produzca algún error durante el proceso, el método lanzará una excepción *TransformersException*. Como parámetros recibe:

- **serviceReq:** Parámetro que representa el nombre del servicio para el que construir la estructura XML. Los valores permitidos son:
    - **AlmacenarDocumento** → Servicio de Almacenamiento de Documentos de @Firma.
    - **StoreDocument** → Servicio de Almacenar Documento, en inglés, de @Firma.
    - **ValidarCertificado** → Servicio de Validación de Certificados de @Firma.
    - **ValidateCertificate** → Servicio de Validación de Certificados, en inglés, de @Firma.
    - **ObtenerInfoCertificado** → Servicio de Obtención de Información de Certificado de @Firma.
    - **GetInfoCertificate** → Servicio de Obtención de Información de Certificado, en inglés, de @Firma.
    - **ValidarFirma** → Servicio de Validación de Firmas de @Firma.
    - **SignatureValidation** → Servicio de Validación de Firmas, en inglés, de @Firma.
    - **ServerSignature** → Servicio de Firma Servidor, en inglés, de @Firma.
    - **FirmaServidor** → Servicio de Firma Servidor de @Firma.
    - **FirmaServidorCoSign** → Servicio de Firma Servidor CoSign de @Firma.

- **ServerSignatureCoSign** → Servicio de Firma Servidor CoSign, en inglés, de @Firma.
- **FirmaServidorCounterSign** → Servicio de Firma Servidor CounterSign de @Firma.
- **ServerSignatureCounterSign** → Servicio de Firma Servidor CounterSign, en inglés, de @Firma.
- **DSSAfirmaSign** → Servicios de Firma Delegada de @Firma.
- **ThreePhaseUserSignatureF1** → Servicio Fase 1 de Firma Usuario 3 Fases, en inglés, de @Firma.
- **FirmaUsuario3FasesF1** → Servicio Fase 1 de Firma Usuario 3 Fases de @Firma.
- **ThreePhaseUserSignatureF1CoSign** → Servicio Fase 1 de Firma Usuario 3 Fases CoSign, en inglés, de @Firma.
- **FirmaUsuario3FasesF1CoSign** → Servicio Fase 1 de Firma Usuario 3 Fases CoSign de @Firma.
- **ThreePhaseUserSignatureF1CounterSign** → Servicio Fase 1 de Firma Usuario 3 Fases CounterSign, en inglés, de @Firma.
- **FirmaUsuario3FasesF1CounterSign** → Servicio Fase 1 de Firma Usuario 3 Fases CounterSign de @Firma.
- **ThreePhaseUserSignatureF3** → Servicio Fase 3 de Firma Usuario 3 Fases, en inglés, de @Firma.
- **FirmaUsuario3FasesF3** → Servicio Fase 3 de Firma Usuario 3 Fases de @Firma.
- **TwoPhaseUserSignatureF2** → Servicio Fase 2 de Firma Usuario 2 Fases, en inglés, de @Firma.
- **FirmaUsuario2FasesF2** → Servicio Fase 2 de Firma Usuario 2 Fases de @Firma.
- **DSSAfirmaVerify** → Servicio de Validación y Actualización de Firmas de @Firma.
- **DSSAfirmaVerifyCertificate** → Servicio de Validación de Certificados de @Firma.
- **DSSBatchVerifyCertificate** → Servicio de Validación de Certificados por Lote de @Firma.
- **DSSBatchVerifySignature** → Servicio de Validación de Firmas por Lote de @Firma.

- **DSSAsyncRequestStatus** → Servicio de Consulta de Peticiones Asíncronas de @Firma.
- **SignatureReportService** → Servicios de Validación y Generación de Informes de eVisor.
- **DSSTSA** → Servicios de Generación, Renovación y Validación de Sello de Tiempo de TS@.

- **method:** Parámetro que representa el nombre del método asociado al servicio.
- **type:** Parámetro que representa el tipo de estructura XML que generar. Los valores permitidos son:
  - **request** → Para generar una estructura XML de petición.
  - **response** → Para generar una estructura XML de respuesta.
- **version:** Parámetro que representa la versión del servicio.

```
public Document getParserTemplateByRequestType(String serviceReq, String method,
String version) throws TransformersException
```

Método que genera una estructura XML a partir de uno de los documentos XML definidos en el punto 8.10. En caso de que se produzca algún error durante el proceso, el método lanzará una excepción *Afirma5ServiceInvokerException*. Como parámetros recibe:

- **serviceReq:** Parámetro que representa el nombre del servicio para el que construir la estructura XML. Los valores permitidos son:
  - **AlmacenarDocumento** → Servicio de Almacenamiento de Documentos de @Firma.
  - **StoreDocument** → Servicio de Almacenar Documento, en inglés, de @Firma.
  - **ValidarCertificado** → Servicio de Validación de Certificados de @Firma.
  - **ValidateCertificate** → Servicio de Validación de Certificados, en inglés, de @Firma.
  - **ObtenerInfoCertificado** → Servicio de Obtención de Información de Certificado de @Firma.
  - **GetInfoCertificate** → Servicio de Obtención de Información de Certificado, en inglés, de @Firma.
  - **ValidarFirma** → Servicio de Validación de Firmas de @Firma.

- **SignatureValidation** → Servicio de Validación de Firmas, en inglés, de @Firma.
- **ServerSignature** → Servicio de Firma Servidor, en inglés, de @Firma.
- **FirmaServidor** → Servicio de Firma Servidor de @Firma.
- **FirmaServidorCoSign** → Servicio de Firma Servidor CoSign de @Firma.
- **ServerSignatureCoSign** → Servicio de Firma Servidor CoSign, en inglés, de @Firma.
- **FirmaServidorCounterSign** → Servicio de Firma Servidor CounterSign de @Firma.
- **ServerSignatureCounterSign** → Servicio de Firma Servidor CounterSign, en inglés, de @Firma.
- **DSSAfirmaSign** → Servicios de Firma Delegada de @Firma.
- **ThreePhaseUserSignatureF1** → Servicio Fase 1 de Firma Usuario 3 Fases, en inglés, de @Firma.
- **FirmaUsuario3FasesF1** → Servicio Fase 1 de Firma Usuario 3 Fases de @Firma.
- **ThreePhaseUserSignatureF1CoSign** → Servicio Fase 1 de Firma Usuario 3 Fases CoSign, en inglés, de @Firma.
- **FirmaUsuario3FasesF1CoSign** → Servicio Fase 1 de Firma Usuario 3 Fases CoSign de @Firma.
- **ThreePhaseUserSignatureF1CounterSign** → Servicio Fase 1 de Firma Usuario 3 Fases CounterSign, en inglés, de @Firma.
- **FirmaUsuario3FasesF1CounterSign** → Servicio Fase 1 de Firma Usuario 3 Fases CounterSign de @Firma.
- **ThreePhaseUserSignatureF3** → Servicio Fase 3 de Firma Usuario 3 Fases, en inglés, de @Firma.
- **FirmaUsuario3FasesF3** → Servicio Fase 3 de Firma Usuario 3 Fases de @Firma.
- **TwoPhaseUserSignatureF2** → Servicio Fase 2 de Firma Usuario 2 Fases, en inglés, de @Firma.
- **FirmaUsuario2FasesF2** → Servicio Fase 2 de Firma Usuario 2 Fases de @Firma.
- **DSSAfirmaVerify** → Servicio de Validación y Actualización de Firmas de @Firma.
- **DSSAfirmaVerifyCertificate** → Servicio de Validación de Certificados de @Firma.

- **DSSBatchVerifyCertificate** → Servicio de Validación de Certificados por Lote de @Firma.
- **DSSBatchVerifySignature** → Servicio de Validación de Firmas por Lote de @Firma.
- **DSSAsyncRequestStatus** → Servicio de Consulta de Peticiones Asíncronas de @Firma.
- **SignatureReportService** → Servicios de Validación y Generación de Informes de eVisor.
- **DSSTSA** → Servicios de Generación, Renovación y Validación de Sello de Tiempo de TS@.

- **method:** Parámetro que representa el nombre del método asociado al servicio.
- **version:** Parámetro que representa la versión del servicio.

```
public String generateXml(Map<String, Object> parameters, String service, String
version) throws TransformersException
```

Método que obtiene una cadena de texto con formato XML usada para invocar servicios web a partir de un mapa de parámetros. En caso de que se produzca algún error durante el proceso, el método lanzará una excepción *TransformersException*. Como parámetros recibe:

- **parameters:** Mapa con los parámetros de entrada relacionados con la petición XML al servicio web.
- **service:** Parámetro que representa el nombre del servicio web que invocar. Los valores permitidos son:
  - **AlmacenarDocumento** → Servicio de Almacenamiento de Documentos de @Firma.
  - **StoreDocument** → Servicio de Almacenar Documento, en inglés, de @Firma.
  - **ValidarCertificado** → Servicio de Validación de Certificados de @Firma.
  - **ValidateCertificate** → Servicio de Validación de Certificados, en inglés, de @Firma.
  - **ObtenerInfoCertificado** → Servicio de Obtención de Información de Certificado de @Firma.
  - **GetInfoCertificate** → Servicio de Obtención de Información de Certificado, en inglés, de @Firma.
  - **ValidarFirma** → Servicio de Validación de Firmas de @Firma.

- **SignatureValidation** → Servicio de Validación de Firmas, en inglés, de @Firma.
- **ServerSignature** → Servicio de Firma Servidor, en inglés, de @Firma.
- **FirmaServidor** → Servicio de Firma Servidor de @Firma.
- **FirmaServidorCoSign** → Servicio de Firma Servidor CoSign de @Firma.
- **ServerSignatureCoSign** → Servicio de Firma Servidor CoSign, en inglés, de @Firma.
- **FirmaServidorCounterSign** → Servicio de Firma Servidor CounterSign de @Firma.
- **ServerSignatureCounterSign** → Servicio de Firma Servidor CounterSign, en inglés, de @Firma.
- **DSSAfirmaSign** → Servicios de Firma Delegada de @Firma.
- **ThreePhaseUserSignatureF1** → Servicio Fase 1 de Firma Usuario 3 Fases, en inglés, de @Firma.
- **FirmaUsuario3FasesF1** → Servicio Fase 1 de Firma Usuario 3 Fases de @Firma.
- **ThreePhaseUserSignatureF1CoSign** → Servicio Fase 1 de Firma Usuario 3 Fases CoSign, en inglés, de @Firma.
- **FirmaUsuario3FasesF1CoSign** → Servicio Fase 1 de Firma Usuario 3 Fases CoSign de @Firma.
- **ThreePhaseUserSignatureF1CounterSign** → Servicio Fase 1 de Firma Usuario 3 Fases CounterSign, en inglés, de @Firma.
- **FirmaUsuario3FasesF1CounterSign** → Servicio Fase 1 de Firma Usuario 3 Fases CounterSign de @Firma.
- **ThreePhaseUserSignatureF3** → Servicio Fase 3 de Firma Usuario 3 Fases, en inglés, de @Firma.
- **FirmaUsuario3FasesF3** → Servicio Fase 3 de Firma Usuario 3 Fases de @Firma.
- **TwoPhaseUserSignatureF2** → Servicio Fase 2 de Firma Usuario 2 Fases, en inglés, de @Firma.
- **FirmaUsuario2FasesF2** → Servicio Fase 2 de Firma Usuario 2 Fases de @Firma.
- **DSSAfirmaVerify** → Servicio de Validación y Actualización de Firmas de @Firma.
- **DSSAfirmaVerifyCertificate** → Servicio de Validación de Certificados de @Firma.

- **DSSBatchVerifyCertificate** → Servicio de Validación de Certificados por Lote de @Firma.
- **DSSBatchVerifySignature** → Servicio de Validación de Firmas por Lote de @Firma.
- **DSSAsyncRequestStatus** → Servicio de Consulta de Peticiones Asíncronas de @Firma.
- **SignatureReportService** → Servicios de Validación y Generación de Informes de eVisor.
- **DSSTSA** → Servicios de Generación, Renovación y Validación de Sello de Tiempo de TS@.

- **version:** Parámetro que representa la versión del servicio web.

```
public String generateXml(Map<String, Object> parameters, String service, String
method, String version) throws TransformersException
```

Método que obtiene una cadena de texto con formato XML usada para invocar servicios web a partir de un mapa de parámetros. En caso de que se produzca algún error durante el proceso, el método lanzará una excepción *TransformersException*. Como parámetros recibe:

- **parameters:** Mapa con los parámetros de entrada relacionados con la petición XML al servicio web.
- **service:** Parámetro que representa el nombre del servicio web que invocar. Los valores permitidos son:
  - **AlmacenarDocumento** → Servicio de Almacenamiento de Documentos de @Firma.
  - **StoreDocument** → Servicio de Almacenar Documento, en inglés, de @Firma.
  - **ValidarCertificado** → Servicio de Validación de Certificados de @Firma.
  - **ValidateCertificate** → Servicio de Validación de Certificados, en inglés, de @Firma.
  - **ObtenerInfoCertificado** → Servicio de Obtención de Información de Certificado de @Firma.
  - **GetInfoCertificate** → Servicio de Obtención de Información de Certificado, en inglés, de @Firma.
  - **ValidarFirma** → Servicio de Validación de Firmas de @Firma.

- **SignatureValidation** → Servicio de Validación de Firmas, en inglés, de @Firma.
- **ServerSignature** → Servicio de Firma Servidor, en inglés, de @Firma.
- **FirmaServidor** → Servicio de Firma Servidor de @Firma.
- **FirmaServidorCoSign** → Servicio de Firma Servidor CoSign de @Firma.
- **ServerSignatureCoSign** → Servicio de Firma Servidor CoSign, en inglés, de @Firma.
- **FirmaServidorCounterSign** → Servicio de Firma Servidor CounterSign de @Firma.
- **ServerSignatureCounterSign** → Servicio de Firma Servidor CounterSign, en inglés, de @Firma.
- **DSSAfirmaSign** → Servicios de Firma Delegada de @Firma.
- **ThreePhaseUserSignatureF1** → Servicio Fase 1 de Firma Usuario 3 Fases, en inglés, de @Firma.
- **FirmaUsuario3FasesF1** → Servicio Fase 1 de Firma Usuario 3 Fases de @Firma.
- **ThreePhaseUserSignatureF1CoSign** → Servicio Fase 1 de Firma Usuario 3 Fases CoSign, en inglés, de @Firma.
- **FirmaUsuario3FasesF1CoSign** → Servicio Fase 1 de Firma Usuario 3 Fases CoSign de @Firma.
- **ThreePhaseUserSignatureF1CounterSign** → Servicio Fase 1 de Firma Usuario 3 Fases CounterSign, en inglés, de @Firma.
- **FirmaUsuario3FasesF1CounterSign** → Servicio Fase 1 de Firma Usuario 3 Fases CounterSign de @Firma.
- **ThreePhaseUserSignatureF3** → Servicio Fase 3 de Firma Usuario 3 Fases, en inglés, de @Firma.
- **FirmaUsuario3FasesF3** → Servicio Fase 3 de Firma Usuario 3 Fases de @Firma.
- **TwoPhaseUserSignatureF2** → Servicio Fase 2 de Firma Usuario 2 Fases, en inglés, de @Firma.
- **FirmaUsuario2FasesF2** → Servicio Fase 2 de Firma Usuario 2 Fases de @Firma.
- **DSSAfirmaVerify** → Servicio de Validación y Actualización de Firmas de @Firma.
- **DSSAfirmaVerifyCertificate** → Servicio de Validación de Certificados de @Firma.

- **DSSBatchVerifyCertificate** → Servicio de Validación de Certificados por Lote de @Firma.
- **DSSBatchVerifySignature** → Servicio de Validación de Firmas por Lote de @Firma.
- **DSSAsyncRequestStatus** → Servicio de Consulta de Peticiones Asíncronas de @Firma.
- **SignatureReportService** → Servicios de Validación y Generación de Informes de eVisor.
- **DSSTSA** → Servicios de Generación, Renovación y Validación de Sello de Tiempo de TS@.

- **method:** Parámetro que representa el nombre del método asociado al servicio.
- **version:** Parámetro que representa la versión del servicio web.

```
public Map<String, Object> parseResponse(String response, String service, String
version) throws TransformersException
```

Método que procesa una respuesta XML de un servicio web para generar un mapa con los elementos contenidos en la respuesta. En caso de que se produzca algún error durante el proceso, el método lanzará una excepción *TransformersException*. Como parámetros recibe:

- **response:** Parámetro que representa la respuesta XML.
- **service:** Parámetro que representa el nombre del servicio web que invocar. Los valores permitidos son:
  - **AlmacenarDocumento** → Servicio de Almacenamiento de Documentos de @Firma.
  - **StoreDocument** → Servicio de Almacenar Documento, en inglés, de @Firma.
  - **ValidarCertificado** → Servicio de Validación de Certificados de @Firma.
  - **ValidateCertificate** → Servicio de Validación de Certificados, en inglés, de @Firma.
  - **ObtenerInfoCertificado** → Servicio de Obtención de Información de Certificado de @Firma.
  - **GetInfoCertificate** → Servicio de Obtención de Información de Certificado, en inglés, de @Firma.
  - **ValidarFirma** → Servicio de Validación de Firmas de @Firma.
  - **SignatureValidation** → Servicio de Validación de Firmas, en inglés, de @Firma.

- **ServerSignature** → Servicio de Firma Servidor, en inglés, de @Firma.
- **FirmaServidor** → Servicio de Firma Servidor de @Firma.
- **FirmaServidorCoSign** → Servicio de Firma Servidor CoSign de @Firma.
- **ServerSignatureCoSign** → Servicio de Firma Servidor CoSign, en inglés, de @Firma.
- **FirmaServidorCounterSign** → Servicio de Firma Servidor CounterSign de @Firma.
- **ServerSignatureCounterSign** → Servicio de Firma Servidor CounterSign, en inglés, de @Firma.
- **DSSAfirmaSign** → Servicios de Firma Delegada de @Firma.
- **ThreePhaseUserSignatureF1** → Servicio Fase 1 de Firma Usuario 3 Fases, en inglés, de @Firma.
- **FirmaUsuario3FasesF1** → Servicio Fase 1 de Firma Usuario 3 Fases de @Firma.
- **ThreePhaseUserSignatureF1CoSign** → Servicio Fase 1 de Firma Usuario 3 Fases CoSign, en inglés, de @Firma.
- **FirmaUsuario3FasesF1CoSign** → Servicio Fase 1 de Firma Usuario 3 Fases CoSign de @Firma.
- **ThreePhaseUserSignatureF1CounterSign** → Servicio Fase 1 de Firma Usuario 3 Fases CounterSign, en inglés, de @Firma.
- **FirmaUsuario3FasesF1CounterSign** → Servicio Fase 1 de Firma Usuario 3 Fases CounterSign de @Firma.
- **ThreePhaseUserSignatureF3** → Servicio Fase 3 de Firma Usuario 3 Fases, en inglés, de @Firma.
- **FirmaUsuario3FasesF3** → Servicio Fase 3 de Firma Usuario 3 Fases de @Firma.
- **TwoPhaseUserSignatureF2** → Servicio Fase 2 de Firma Usuario 2 Fases, en inglés, de @Firma.
- **FirmaUsuario2FasesF2** → Servicio Fase 2 de Firma Usuario 2 Fases de @Firma.
- **DSSAfirmaVerify** → Servicio de Validación y Actualización de Firmas de @Firma.
- **DSSAfirmaVerifyCertificate** → Servicio de Validación de Certificados de @Firma.

- **DSSBatchVerifyCertificate** → Servicio de Validación de Certificados por Lote de @Firma.
- **DSSBatchVerifySignature** → Servicio de Validación de Firmas por Lote de @Firma.
- **DSSAsyncRequestStatus** → Servicio de Consulta de Peticiones Asíncronas de @Firma.
- **SignatureReportService** → Servicios de Validación y Generación de Informes de eVisor.
- **DSSTSA** → Servicios de Generación, Renovación y Validación de Sello de Tiempo de TS@.

- **version:** Parámetro que representa la versión del servicio web.

```
public Map<String, Object> parseResponse(String response, String service, String
method, String version) throws TransformersException
```

Método que procesa una respuesta XML de un servicio web para generar un mapa con los elementos contenidos en la respuesta. En caso de que se produzca algún error durante el proceso, el método lanzará una excepción *TransformersException*. Como parámetros recibe:

- **response:** Parámetro que representa la respuesta XML.
- **service:** Parámetro que representa el nombre del servicio web que invocar. Los valores permitidos son:
  - **AlmacenarDocumento** → Servicio de Almacenamiento de Documentos de @Firma.
  - **StoreDocument** → Servicio de Almacenar Documento, en inglés, de @Firma.
  - **ValidarCertificado** → Servicio de Validación de Certificados de @Firma.
  - **ValidateCertificate** → Servicio de Validación de Certificados, en inglés, de @Firma.
  - **ObtenerInfoCertificado** → Servicio de Obtención de Información de Certificado de @Firma.
  - **GetInfoCertificate** → Servicio de Obtención de Información de Certificado, en inglés, de @Firma.
  - **ValidarFirma** → Servicio de Validación de Firmas de @Firma.
  - **SignatureValidation** → Servicio de Validación de Firmas, en inglés, de @Firma.

- **ServerSignature** → Servicio de Firma Servidor, en inglés, de @Firma.
- **FirmaServidor** → Servicio de Firma Servidor de @Firma.
- **FirmaServidorCoSign** → Servicio de Firma Servidor CoSign de @Firma.
- **ServerSignatureCoSign** → Servicio de Firma Servidor CoSign, en inglés, de @Firma.
- **FirmaServidorCounterSign** → Servicio de Firma Servidor CounterSign de @Firma.
- **ServerSignatureCounterSign** → Servicio de Firma Servidor CounterSign, en inglés, de @Firma.
- **DSSAfirmaSign** → Servicios de Firma Delegada de @Firma.
- **ThreePhaseUserSignatureF1** → Servicio Fase 1 de Firma Usuario 3 Fases, en inglés, de @Firma.
- **FirmaUsuario3FasesF1** → Servicio Fase 1 de Firma Usuario 3 Fases de @Firma.
- **ThreePhaseUserSignatureF1CoSign** → Servicio Fase 1 de Firma Usuario 3 Fases CoSign, en inglés, de @Firma.
- **FirmaUsuario3FasesF1CoSign** → Servicio Fase 1 de Firma Usuario 3 Fases CoSign de @Firma.
- **ThreePhaseUserSignatureF1CounterSign** → Servicio Fase 1 de Firma Usuario 3 Fases CounterSign, en inglés, de @Firma.
- **FirmaUsuario3FasesF1CounterSign** → Servicio Fase 1 de Firma Usuario 3 Fases CounterSign de @Firma.
- **ThreePhaseUserSignatureF3** → Servicio Fase 3 de Firma Usuario 3 Fases, en inglés, de @Firma.
- **FirmaUsuario3FasesF3** → Servicio Fase 3 de Firma Usuario 3 Fases de @Firma.
- **TwoPhaseUserSignatureF2** → Servicio Fase 2 de Firma Usuario 2 Fases, en inglés, de @Firma.
- **FirmaUsuario2FasesF2** → Servicio Fase 2 de Firma Usuario 2 Fases de @Firma.
- **DSSAfirmaVerify** → Servicio de Validación y Actualización de Firmas de @Firma.
- **DSSAfirmaVerifyCertificate** → Servicio de Validación de Certificados de @Firma.

- **DSSBatchVerifyCertificate** → Servicio de Validación de Certificados por Lote de @Firma.
- **DSSBatchVerifySignature** → Servicio de Validación de Firmas por Lote de @Firma.
- **DSSAsyncRequestStatus** → Servicio de Consulta de Peticiones Asíncronas de @Firma.
- **SignatureReportService** → Servicios de Validación y Generación de Informes de eVisor.
- **DSSTSA** → Servicios de Generación, Renovación y Validación de Sello de Tiempo de TS@.

- **method:** Parámetro que representa el nombre del método asociado al servicio.
- **version:** Parámetro que representa la versión del servicio web.

```
public String getParserParameterValue(String parameterName)
```

Método que obtiene el valor de un elemento definido en el archivo **parserParameters.properties** a partir de su clave. Como parámetros recibe:

- **parameterName:** Parámetro que representa la clave del elemento que obtener.

9.2.3 Paquete es.gob.afirma.integraFacade

Paquete que contiene todas las clases asociadas a la fachada de invocación. Esta fachada facilita a las aplicaciones la integración con los servicios de @Firma. De este paquete se describirán aquellas clases más destacadas.

Clase IntegraFacadeWSNative

Esta clase representa la fachada para la invocación de servicios nativos de @Firma. Está compuesta por los métodos:

```
public static IntegraFacadeWSNative getInstance()
```

Método que obtiene una instancia de la clase.

```
public DocumentResponse storingDocument(DocumentRequest docReq)
```

Método que realiza una petición a @Firma para almacenar documentos. Devuelve un objeto que representa la respuesta del servicio. Para poder llevar a cabo la petición al servicio es necesario tener configurado todo lo referente a la comunicación con @Firma en los ficheros **afirmaXXXXX.properties e integra.properties**. Como parámetros recibe:

- **docReq:** Parámetro que representa la petición al servicio.

```
public ContentResponse deleteDocumentContent(ContentRequest conReq)
```

Método que realiza una petición a @Firma para eliminar el contenido de documentos almacenados. Devuelve un objeto que representa la respuesta del servicio. Para poder llevar a cabo la petición al servicio es necesario tener configurado todo lo referente a la comunicación con @Firma en los ficheros **afirmaXXXXX.properties e integra.properties**. Como parámetros recibe:

- **conReq:** Parámetro que representa la petición al servicio.

```
public ContentResponse getDocumentContent(ContentRequest conReq)
```

Método que realiza una petición al servicio de @Firma que obtiene el contenido de documentos almacenados. Devuelve un objeto que representa la respuesta del servicio. Para poder llevar a cabo la petición al servicio es necesario tener configurado todo lo referente a la comunicación con @Firma en los ficheros **afirmaXXXXX.properties e integra.properties**. Como parámetros recibe:

- **conReq:** Parámetro que representa la petición al servicio.

```
public ContentResponse getContentDocumentId(ContentRequest conReq)
```

Método que realiza una petición a @Firma para obtener el contenido de documentos almacenados a partir de su identificador. Devuelve un objeto que representa la respuesta del servicio. Para poder llevar a cabo la petición al servicio es necesario tener configurado todo lo referente a la comunicación con @Firma en los ficheros **afirmaXXXXX.properties e integra.properties**. Como parámetros recibe:

- **conReq:** Parámetro que representa la petición al servicio.

```
public ContentResponse getDocumentId(ContentRequest conReq)
```

Método que realiza una petición a @Firma para obtener el identificador de documentos almacenados a partir de su identificador de transacción. Devuelve un objeto que representa la respuesta del servicio. Para poder llevar a cabo la petición al servicio es necesario tener configurado todo lo referente a la comunicación con @Firma en los ficheros **afirmaXXXXX.properties e integra.properties**. Como parámetros recibe:

- **conReq:** Parámetro que representa la petición al servicio.

```
public SignatureTransactionResponse getSignatureTransaction(ContentRequest conReq)
```

Método que realiza una petición a @Firma para obtener el contenido de una firma a partir de su identificador de transacción. Devuelve un objeto que representa la respuesta del servicio. Para poder llevar a cabo la petición al servicio es necesario tener configurado todo lo referente a la comunicación con @Firma en los ficheros **afirmaXXXXX.properties e integra.properties**. Como parámetros recibe:

- **conReq:** Parámetro que representa la petición al servicio.

```
public CertificateInfoResponse getCertificateInfo(CertificateInfoRequest
cerInfReq)
```

Método que realiza una petición a @Firma para obtener la información asociada a un certificado. Devuelve un objeto que representa la respuesta del servicio. Para poder llevar a cabo la petición al servicio es necesario tener configurado todo lo referente a la comunicación con @Firma en los ficheros **afirmaXXXXX.properties e integra.properties**. Como parámetros recibe:

- **cerInfReq:** Parámetro que representa la petición al servicio.

Clase IntegraFacadeWSDSS

Esta clase representa la fachada para la invocación de servicios DSS de @Firma. Está compuesta por los métodos:

```
public static IntegraFacadeWSDSS getInstance()
```

Método que obtiene una instancia de la clase.

```
public ServerSignerResponse sign(ServerSignerRequest serSigReq)
```

Método que realiza una petición a @Firma para realizar una firma. Devuelve un objeto que representa la respuesta del servicio. Para poder llevar a cabo la petición al servicio es necesario tener configurado todo lo referente a la comunicación con @Firma en los ficheros **afirmaXXXXX.properties e integra.properties**. Como parámetros recibe:

- **serSigReq:** Parámetro que representa la petición al servicio.

```
public ServerSignerResponse coSign(CoSignRequest coSigReq)
```

Método que realiza una petición a @Firma para realizar una co-firma. Devuelve un objeto que representa la respuesta del servicio. Para poder llevar a cabo la petición al servicio es necesario tener configurado todo lo referente a la comunicación con @Firma en los ficheros **afirmaXXXXX.properties e integra.properties**. Como parámetros recibe:

- **coSigReq:** Parámetro que representa la petición al servicio.

```
public ServerSignerResponse counterSign(CounterSignRequest couSigReq)
```

Método que realiza una petición a @Firma para realizar una contra-firma servidor. Devuelve un objeto que representa la respuesta del servicio. Para poder llevar a cabo la petición al servicio es necesario tener configurado todo lo referente a la comunicación con @Firma en los ficheros **afirmaXXXXX.properties e integra.properties**. Como parámetros recibe:

- **couSigReq:** Parámetro que representa la petición al servicio.

```
public VerifySignatureResponse verifySignature(VerifySignatureRequest verSigReq)
```

Método que realiza una petición a @Firma para validar una firma. Devuelve un objeto que representa la respuesta del servicio. Para poder llevar a cabo la petición al servicio es necesario tener configurado todo lo referente a la comunicación con @Firma en los ficheros **afirmaXXXXX.properties e integra.properties**. Como parámetros recibe:

- **verSigReq:** Parámetro que representa la petición al servicio.

```
public ServerSignerResponse upgradeSignature(UpgradeSignatureRequest upgSigReq)
```

Método que realiza una petición a @Firma para actualizar una firma. Devuelve un objeto que representa la respuesta del servicio. Para poder llevar a cabo la petición al servicio es necesario tener configurado todo lo referente a la comunicación con @Firma en los ficheros **afirmaXXXXX.properties e integra.properties**. Como parámetros recibe:

- **upgSigReq:** Parámetro que representa la petición al servicio.

```
public VerifyCertificateResponse verifyCertificate(VerifyCertificateRequest
verCerReq)
```

Método que realiza una petición a @Firma para validar un certificado. Devuelve un objeto que representa la respuesta del servicio. Para poder llevar a cabo la petición al servicio es necesario tener configurado todo lo referente a la comunicación con @Firma en los ficheros **afirmaXXXXX.properties e integra.properties**. Como parámetros recibe:

- **verCerReq:** Parámetro que representa la petición al servicio.

```
public BatchVerifySignatureResponse
batchVerifySignature(BatchVerifySignatureRequest batVerSigReq)
```

Método que realiza una petición a @Firma para validar firmas en lote. Devuelve un objeto que representa la respuesta del servicio. Para poder llevar a cabo la petición al servicio es necesario tener configurado todo lo referente a la comunicación con @Firma en los ficheros **afirmaXXXXX.properties e integra.properties**. Como parámetros recibe:

- **batVerSigReq:** Parámetro que representa la petición al servicio.

```
public BatchVerifyCertificateResponse
batchVerifyCertificate(BatchVerifyCertificateRequest batVerCerReq)
```

Método que realiza una petición a @Firma para validar certificados en lote. Devuelve un objeto que representa la respuesta del servicio. Para poder llevar a cabo la petición al servicio es necesario tener configurado todo lo referente a la comunicación con @Firma en los ficheros **afirmaXXXXX.properties e integra.properties**. Como parámetros recibe:

- **batVerCerReq:** Parámetro que representa la petición al servicio.

```
public AsynchronousResponse asynchronousRequest(PendingRequest pendingRequest)
```

Método que realiza una petición a @Firma para realizar consultas sobre peticiones asíncronas. Devuelve un objeto que representa la respuesta del servicio. Para poder llevar a cabo la petición al servicio es necesario tener configurado todo lo referente a la comunicación con @Firma en los ficheros **afirmaXXXXX.properties e integra.properties**. Como parámetros recibe:

- **pendingRequest:** Parámetro que representa la petición al servicio.

```
public ArchiveResponse getArchiveRetrieval(ArchiveRequest archiveRequest)
```

Método que realiza una petición a @Firma para obtener firmas almacenadas. Devuelve un objeto que representa la respuesta del servicio. Para poder llevar a cabo la petición al servicio es necesario tener configurado todo lo referente a la comunicación con @Firma en los ficheros **afirmaXXXXX.properties e integra.properties**. Como parámetros recibe:

- **archiveRequest:** Parámetro que representa la petición al servicio.

Clase TsaIntegraFacadeWSDSS

Esta clase representa la fachada para la invocación de servicios DSS de TS@. Está compuesta por los métodos:

```
public static TsaIntegraFacadeWSDSS getInstance()
```

Método que obtiene una instancia de la clase.

```
public TimestampResponse generateTimestamp(TimestampRequest timestampReq)
```

Método que realiza una petición a TS@ para realizar un sello de tiempo sobre un documento. Devuelve un objeto que representa la respuesta del servicio. Para poder llevar a cabo la petición al servicio es necesario tener configurado todo lo referente a la comunicación con TS@ en los ficheros **tsaXXXXX.properties e integra properties** y para las transformadas el fichero **transformers.properties**. Como parámetros recibe:

- **timestampReq:** Parámetro que representa la petición al servicio.

```
public TimestampResponse verifyTimestamp(TimestampRequest timestampReq)
```

Método que realiza una petición a TS@ para validar un sello de tiempo. Devuelve un objeto que representa la respuesta del servicio. Para poder llevar a cabo la petición al servicio es necesario tener configurado todo lo referente a la comunicación con TS@ en los ficheros **tsaXXXXX.properties e integra properties** y para las transformadas el fichero **transformers.properties**. Como parámetros recibe:

- **timestampReq:** Parámetro que representa la petición al servicio.

```
public TimestampResponse renewTimestamp(TimestampRequest timestampReq)
```

Método que realiza una petición a TS@ para realizar un resellado o renovación de sello de tiempo. Devuelve un objeto que representa la respuesta del servicio. Para poder llevar a cabo la petición al servicio es necesario tener configurado todo lo referente a la comunicación con TS@ en los ficheros **tsaXXXXX.properties e integra properties** y para las transformadas el fichero **transformers.properties**. Como parámetros recibe:

- **timestampReq:** Parámetro que representa la petición al servicio.

9.2.4 Paquete es.gob.afirma.integraFacade.pojo

Paquete que contiene todas las clases que representan parámetros de entrada y de salida para la invocación de servicios de @Firma a través de la fachada de servicios nativos (representada por la Clase IntegraFacadeWSNative), y de la fachada de servicios DSS (representada por la Clase IntegraFacadeWSDSS).

Clase ArchiveRequest

Esta clase es un POJO que representa una petición para el servicio de @Firma que permite obtener firmas almacenadas. Está compuesta por los métodos:

```
public final String getApplicationId()
```

Método que devuelve el identificador de la aplicación cliente.

```
public final void setApplicationId(String applicationIdParam)
```

Método que establece el identificador de la aplicación cliente. Como parámetros recibe:

- **applicationIdParam:** Parámetro que representa el identificador de la aplicación cliente.

```
public final String getTransactionId()
```

Método que devuelve el identificador de transacción de la firma a recuperar.

```
public final void setTransactionId(String transactionIdParam)
```

Método que establece el identificador de transacción de la firma a recuperar. Como parámetros recibe:

- **transactionIdParam:** Parámetro que representa el identificador de transacción de la firma a recuperar.

Clase ArchiveResponse

Esta clase es un POJO que representa una respuesta del servicio de @Firma que permite obtener firmas almacenadas. Está compuesta por los métodos:

```
public final Result getResult()
```

Método que devuelve el resultado del proceso.

```
public final void setResult(Result resultParam)
```

Método que establece el resultado del proceso. Como parámetros recibe:

- **resultParam:** Parámetro que representa el resultado del proceso.

```
public final byte[ ] getSignature()
```

Método que devuelve la firma.

```
public final void setSignature(byte[ ] signatureParam)
```

Método que establece la firma. Como parámetros recibe:

- **signatureParam:** Parámetro que representa la firma.

Clase AsynchronousResponse

Esta clase es un POJO que representa una respuesta del servicio de @Firma que permite consultar peticiones asíncronas. Está compuesta por los métodos:

```
public final ServerSignerResponse getSerSigRes()
```

Método que devuelve la respuesta del servicio de firma servidor.

```
public final void setSerSigRes(ServerSignerResponse serSigResParam)
```

Método que establece la respuesta del servicio de firma servidor. Como parámetros recibe:

- **serSigResParam:** Parámetro que representa la respuesta del servicio de firma servidor.

```
public final BatchVerifyCertificateResponse getBatVerCerRes()
```

Método que devuelve la respuesta del servicio de validación de certificados en lote.

```
public final void setBatVerCerRes(BatchVerifyCertificateResponse
batVerCerResParam)
```

Método que establece la respuesta del servicio de validación de certificados en lote. Como parámetros recibe:

- **batVerCerResParam:** Parámetro que representa la respuesta del servicio de validación de certificados en lote.

```
public final BatchVerifySignatureResponse getBatVerSigRes()
```

Método que devuelve la respuesta del servicio de validación de firmas en lote.

```
public final void setBatVerSigRes(BatchVerifySignatureResponse batVerSigResParam)
```

Método que establece la respuesta del servicio de validación de firmas en lote. Como parámetros recibe:

- **batVerSigResParam:** Parámetro que representa la respuesta del servicio de validación de firmas en lote.

```
public final InvalidAsyncResponse getInvAsyRes()
```

Método que devuelve la respuesta incorrecta del servicio.

```
public final void setInvAsyRes(InvalidAsyncResponse invAsyResParam)
```

Método que establece la respuesta incorrecta del servicio. Como parámetros recibe:

- **invAsyResParam:** Parámetro que representa la respuesta incorrecta del servicio.

Clase BatchVerifyCertificateRequest

Esta clase es un POJO que representa una petición al servicio de @Firma que permite validar certificados en lote. Está compuesta por los métodos:

```
public final List<VerifyCertificateRequest> getListVerifyCertificate()
```

Método que devuelve la lista de peticiones para validar certificados.

```
public final void setListVerifyCertificate(List<VerifyCertificateRequest>
listVerifyCertificateParam)
```

Método que establece la lista de peticiones para validar certificados. Como parámetros recibe:

- **listVerifyCertificateParam:** Parámetro que representa la lista de peticiones para validar certificados.

```
public final String getApplicationId()
```

Método que devuelve el identificador de la aplicación cliente.

```
public final void setApplicationId(String applicationIdParam)
```

Método que establece el identificador de la aplicación cliente. Como parámetros recibe:

- **applicationIdParam:** Parámetro que representa el identificador de la aplicación cliente.

Clase BatchVerifyCertificateResponse

Esta clase es un POJO que representa una respuesta del servicio de @Firma que permite validar certificados en lote. Está compuesta por los métodos:

```
public final Result getResult()
```

Método que devuelve el resultado del proceso.

```
public final void setResult(Result resultParam)
```

Método que establece el resultado del proceso. Como parámetros recibe:

- **resultParam:** Parámetro que representa el resultado del proceso.

```
public final String getAsyncResponse()
```

Método que devuelve la respuesta asíncrona.

```
public final void setAsyncResponse(String asyncResponseParam)
```

Método que establece la respuesta asíncrona. Como parámetros recibe:

- **asyncResponseParam:** Parámetro que representa la respuesta asíncrona.

```
public final List<VerifyCertificateResponse> getListVerifyResponse()
```

Método que devuelve la lista con las respuestas de la validación de cada uno de los certificados.

```
public final void setListVerifyResponse(List<VerifyCertificateResponse>
listVerifyResponseParam)
```

Método que establece la lista con las respuestas de la validación de cada uno de los certificados. Como parámetros recibe:

- **listVerifyResponseParam:** Parámetro que representa la lista con las respuestas de la validación de cada uno de los certificados.

Clase BatchVerifySignatureRequest

Esta clase es un POJO que representa una petición al servicio de @Firma que permite validar firmas en lote. Está compuesta por los métodos:

```
public final List<VerifySignatureRequest> getListVerifySignature()
```

Método que devuelve la lista de peticiones para validar firmas.

```
public final void setListVerifySignature(List<VerifySignatureRequest>
listVerifySignatureParam)
```

Método que establece la lista de peticiones para validar firmas. Como parámetros recibe:

- **listVerifySignatureParam:** Parámetro que representa la lista de peticiones para validar firmas.

```
public final String getApplicationId()
```

Método que devuelve el identificador de la aplicación cliente.

```
public final void setApplicationId(String applicationIdParam)
```

Método que establece el identificador de la aplicación cliente. Como parámetros recibe:

- **applicationIdParam:** Parámetro que representa el identificador de la aplicación cliente.

Clase BatchVerifySignatureResponse

Esta clase es un POJO que representa una respuesta del servicio de @Firma que permite validar firmas en lote. Está compuesta por los métodos:

```
public final Result getResult()
```

Método que devuelve el resultado del proceso.

```
public final void setResult(Result resultParam)
```

Método que establece el resultado del proceso. Como parámetros recibe:

- **resultParam:** Parámetro que representa el resultado del proceso.

```
public final String getAsyncResponse()
```

Método que devuelve la respuesta asíncrona.

```
public final void setAsyncResponse(String asyncResponseParam)
```

Método que establece la respuesta asíncrona. Como parámetros recibe:

- **asyncResponseParam:** Parámetro que representa la respuesta asíncrona.

```
public final List<VerifySignatureResponse> getListVerifyResponse()
```

Método que devuelve la lista con las respuestas de la validación de cada una de las firmas.

```
public final void setListVerifyResponse(List<VerifySignatureResponse>
listVerifyResponseParam)
```

Método que establece la lista con las respuestas de la validación de cada una de las firmas. Como parámetros recibe:

- **listVerifyResponseParam:** Parámetro que representa la lista con las respuestas de la validación de cada una de las firmas.

Clase CertificateInfoRequest

Esta clase es un POJO que representa una petición al servicio de @Firma que permite obtener la información de un certificado. Está compuesta por los métodos:

```
public final String getApplicationId()
```

Método que devuelve el identificador de la aplicación cliente.

```
public final void setApplicationId(String applicationIdParam)
```

Método que establece el identificador de la aplicación cliente. Como parámetros recibe:

- **applicationIdParam:** Parámetro que representa el identificador de la aplicación cliente.

```
public final byte[ ] getCertificate()
```

Método que devuelve el certificado.

```
public final void setCertificate(byte[ ] certificateParam)
```

Método que establece el certificado. Como parámetros recibe:

- **certificateParam:** Parámetro que representa el certificado.

Clase CertificateInfoResponse

Esta clase es un POJO que representa una respuesta del servicio de @Firma que permite obtener la información de un certificado. Está compuesta por los métodos:

```
public final Map<String, Object> getMapInfoCertificate()
```

Método que devuelve el mapa con cada uno de los elementos que componen la información asociada al certificado.

```
public final void setMapInfoCertificate(Map<String, Object>
mapInfoCertificateParam)
```

Método que establece el mapa con cada uno de los elementos que componen la información asociada al certificado. Como parámetros recibe:

- **mapInfoCertificateParam:** Parámetro que representa el mapa con cada uno de los elementos que componen la información asociada al certificado.

```
public final ErrorResponse getError()
```

Método que devuelve el mensaje asociado a una respuesta de error.

```
public final void setError(ErrorResponse errorParam)
```

Método que establece el mensaje asociado a una respuesta de error. Como parámetros recibe:

- **errorParam:** Parámetro que representa el mensaje asociado a una respuesta de error.

Clase CertificatePathValidity

Esta clase es un POJO que representa la información asociada a la validación de la cadena de certificación de un certificado. Está compuesta por los métodos:

```
public final Detail getSummary()
```

Método que devuelve el resultado global de verificar el certificado firmante.

```
public final void setSummary(Detail summaryParam)
```

Método que establece el resultado global de verificar el certificado firmante. Como parámetros recibe:

- **summaryParam:** Parámetro que representa el resultado global de verificar el certificado firmante.

```
public final String getIdentifier()
```

Método que devuelve el identificador asociado al certificado (emisor y número de serie).

```
public final void setIdentifier(String identifierParam)
```

Método que establece el identificador asociado al certificado (emisor y número de serie). Como parámetros recibe:

- **identifierParam:** Parámetro que representa el identificador asociado al certificado (emisor y número de serie).

```
public final List<CertificateValidity> getDetail()
```

Método que devuelve la lista con la información de verificación de cada uno de los certificados que componen la cadena de certificación.

```
public final void setDetail(List<CertificateValidity> detailParam)
```

Método que establece la lista con la información de verificación de cada uno de los certificados que componen la cadena de certificación. Como parámetros recibe:

- **detailParam:** Parámetro que representa la lista con la información de verificación de cada uno de los certificados que componen la cadena de certificación.

Clase CertificateValidity

Esta clase es un POJO que representa la información asociada a la validación de un certificado. Está compuesta por los métodos:

```
public Map<String, String> getInfoMap()
```

Método que devuelve el mapa con todos los aspectos asociados a la validación del certificado.

```
public void setInfoMap(Map<String, String> infoMapParam)
```

Método que establece el mapa con todos los aspectos asociados a la validación del certificado. Como parámetros recibe:

- **infoMapParam:** Parámetro que representa el mapa con todos los aspectos asociados a la validación del certificado.

Clase ContentRequest

Esta clase es un POJO que representa una petición a los servicios de @Firma que permiten eliminar el contenido de un documento, obtener un documento, u obtener una firma. Está compuesta por los métodos:

```
public final String getApplicationId()
```

Método que devuelve el identificador de la aplicación cliente.

```
public final void setApplicationId(String applicationIdParam)
```

Método que establece el identificador de la aplicación cliente. Como parámetros recibe:

- **applicationIdParam:** Parámetro que representa el identificador de la aplicación cliente.

```
public final String getTransactionId()
```

Método que devuelve el identificador de la transacción asociada o el identificador del documento asociado.

```
public final void setTransactionId(String idTransactionParam)
```

Método que establece el identificador de la transacción asociada o el identificador del documento asociado. Como parámetros recibe:

- **idTransactionParam:** Parámetro que representa el identificador de la transacción asociada o el identificador del documento asociado.

Clase ContentResponse

Esta clase es un POJO que representa una respuesta de los servicios de @Firma que permiten eliminar el contenido de un documento, obtener un documento, u obtener una firma. Está compuesta por los métodos:

```
public final boolean isState()
```

Método que devuelve un valor lógico que indica si la operación finalizó correctamente (verdadero) o no (falso).

```
public final void setState(boolean stateParam)
```

Método que establece si la operación finalizó correctamente (verdadero) o no (falso). Como parámetros recibe:

- **stateParam:** Parámetro que indica si la operación finalizó correctamente (verdadero) o no (falso).

```
public final String getDescription()
```

Método que devuelve la descripción del error o de la excepción producida durante el proceso.

```
public final void setDescription(String descriptionParam)
```

Método que establece la descripción del error o de la excepción producida durante el proceso. Como parámetros recibe:

- **descriptionParam:** Parámetro que representa la descripción del error o de la excepción producida durante el proceso.

```
public final byte[ ] getContent()
```

Método que devuelve el documento o la firma obtenido.

```
public final void setContent(byte[ ] contentParam)
```

Método que establece el documento o la firma obtenido. Como parámetros recibe:

- **contentParam:** Parámetro que representa el documento o la firma obtenido.

```
public final ErrorResponse getError()
```

Método que devuelve la información asociada al error que se haya producido durante el proceso.

```
public final void setError(ErrorResponse errorParam)
```

Método que establece la información asociada al error que se haya producido durante el proceso. Como parámetros recibe:

- **errorParam:** Parámetro que representa la información asociada al error que se haya producido durante el proceso.

Clase CoSignRequest

Esta clase es un POJO que representa una petición al servicio de @Firma que permite generar una co-firma. Está compuesta por los métodos:

```
public final String getTransactionId()
```

Método que devuelve el identificador de transacción asociado a la generación de la firma que co-firmar.

```
public final void setTransactionId(String transactionIdParam)
```

Método que establece el identificador de transacción asociado a la generación de la firma que co-firmar. Como parámetros recibe:

- **transactionIdParam:** Parámetro que representa el identificador de transacción asociado a la generación de la firma que co-firmar.

```
public final Repository getDocumentRepository()
```

Método que devuelve la ubicación del documento original en un gestor de documentos o repositorio.

```
public final void setDocumentRepository(Repository documentRepositoryParam)
```

Método que establece la ubicación del documento original en un gestor de documentos o repositorio. Como parámetros recibe:

- **documentRepositoryParam:** Parámetro que representa la ubicación del documento original en un gestor de documentos o repositorio.

```
public final Repository getSignatureRepository()
```

Método que devuelve la ubicación de la firma que co-firmar en un gestor de documentos o repositorio.

```
public final void setSignatureRepository(Repository signatureRepositoryParam)
```

Método que establece la ubicación de la firma que co-firmar en un gestor de documentos o repositorio. Como parámetros recibe:

- **signatureRepositoryParam:** Parámetro que representa la ubicación de la firma que co-firmar en un gestor de documentos o repositorio.

```
public final String getApplicationId()
```

Método que devuelve el identificador de la aplicación cliente.

```
public final void setApplicationId(String applicationIdParam)
```

Método que establece el identificador de la aplicación cliente. Como parámetros recibe:

- **applicationIdParam:** Parámetro que representa el identificador de la aplicación cliente.

```
public final String getKeySelector()
```

Método que devuelve el identificador de la clave usada para generar la co-firma delegada.

```
public final void setKeySelector(String keySelectorParam)
```

Método que establece el identificador de la clave usada para generar la co-firma delegada. Como parámetros recibe:

- **keySelectorParam:** Parámetro que representa el identificador de la clave usada para generar la co-firma delegada.

```
public final HashAlgorithmEnum getHashAlgorithm()
```

Método que devuelve el algoritmo de hash definido para calcular la co-firma.

```
public final void setHashAlgorithm(HashAlgorithmEnum hashAlgorithmParam)
```

Método que establece el algoritmo de hash definido para calcular la co-firma. Como parámetros recibe:

- **hashAlgorithmParam:** Parámetro que representa el algoritmo de hash definido para calcular la co-firma.

```
public final String getSignaturePolicyIdentifier()
```

Método que devuelve el identificador de la política de firma a utilizar en la generación de la co-firma.

```
public final void setSignaturePolicyIdentifier(String
signaturePolicyIdentifierParam)
```

Método que establece el identificador de la política de firma a utilizar en la generación de la co-firma. Como parámetros recibe:

- **signaturePolicyIdentifierParam:** Parámetro que representa el identificador de la política de firma a utilizar en la generación de la co-firma.

```
public final boolean isIgnoreGracePeriod()
```

Método que devuelve un valor lógico que indica si se debe ignorar el periodo de gracia (verdadero) o no (falso).

```
public final void setIgnoreGracePeriod(boolean ignoreGracePeriodParam)
```

Método que establece si se debe ignorar el periodo de gracia (verdadero) o no (falso). Como parámetros recibe:

- **ignoreGracePeriodParam:** Parámetro que indica si se debe ignorar el periodo de gracia (verdadero) o no (falso).

```
public final byte[] getSignature()
```

Método que devuelve la firma a utilizar en la generación de la co-firma.

```
public final void setSignature(byte[] signatureParam)
```

Método que establece la firma a utilizar en la generación de la co-firma. Como parámetros recibe:

- **signatureParam:** Parámetro que representa la firma a utilizar en la generación de la co-firma.

```
public final byte[] getDocument()
```

Método que devuelve el documento original para la generación de la co-firma.

```
public final void setDocument(byte[] documentParam)
```

Método que establece el documento original a utilizar en la generación de la co-firma. Como parámetros recibe:

- **documentParam:** Parámetro que representa el documento a utilizar en la generación de la co-firma.

Clase CounterSignRequest

Esta clase es un POJO que representa una petición al servicio de @Firma que permite generar una contra-firma. Está compuesta por los métodos:

```
public final String getTransactionId()
```

Método que devuelve el identificador de transacción asociado a la generación de la firma que contra-firmar.

```
public final void setTransactionId(String transactionIdParam)
```

Método que establece el identificador de transacción asociado a la generación de la firma que contra-firmar. Como parámetros recibe:

- **transactionIdParam:** Parámetro que representa el identificador de transacción asociado a la generación de la firma que contra-firmar.

```
public final Repository getSignatureRepository()
```

Método que devuelve la ubicación de la firma que contra-firmar en un gestor de documentos o repositorio.

```
public final void setSignatureRepository(Repository signatureRepositoryParam)
```

Método que establece la ubicación de la firma que contra-firmar en un gestor de documentos o repositorio. Como parámetros recibe:

- **signatureRepositoryParam:** Parámetro que representa la ubicación de la firma que contra-firmar en un gestor de documentos o repositorio.

```
public final String getApplicationId()
```

Método que devuelve el identificador de la aplicación cliente.

```
public final void setApplicationId(String applicationIdParam)
```

Método que establece el identificador de la aplicación cliente. Como parámetros recibe:

- **applicationIdParam:** Parámetro que representa el identificador de la aplicación cliente.

```
public final String getKeySelector()
```

Método que devuelve el identificador de la clave usada para generar la contra-firma delegada.

```
public final void setKeySelector(String keySelectorParam)
```

Método que establece el identificador de la clave usada para generar la contra-firma delegada. Como parámetros recibe:

- **keySelectorParam:** Parámetro que representa el identificador de la clave usada para generar la contra -firma delegada.

```
public final HashAlgorithmEnum getHashAlgorithm()
```

Método que devuelve el algoritmo de hash definido para calcular la contra-firma.

```
public final void setHashAlgorithm(HashAlgorithmEnum hashAlgorithmParam)
```

Método que establece el algoritmo de hash definido para calcular la contra-firma. Como parámetros recibe:

- **hashAlgorithmParam:** Parámetro que representa el algoritmo de hash definido para calcular la contra -firma.

```
public final String getSignaturePolicyIdentifier()
```

Método que devuelve el identificador de la política de firma a utilizar en la generación de la contra-firma.

```
public final void setSignaturePolicyIdentifier(String
signaturePolicyIdentifierParam)
```

Método que establece el identificador de la política de firma a utilizar en la generación de la contra-firma. Como parámetros recibe:

- **signaturePolicyIdentifierParam:** Parámetro que representa el identificador de la política de firma a utilizar en la generación de la contra-firma.

```
public final boolean isIgnoreGracePeriod()
```

Método que devuelve un valor lógico que indica si se debe ignorar el periodo de gracia (verdadero) o no (falso).

```
public final void setIgnoreGracePeriod(boolean ignoreGracePeriodParam)
```

Método que establece si se debe ignorar el periodo de gracia (verdadero) o no (falso). Como parámetros recibe:

- **ignoreGracePeriodParam:** Parámetro que indica si se debe ignorar el periodo de gracia (verdadero) o no (falso).

```
public final byte[ ] getTargetSigner()
```

Método que devuelve el firmante objetivo de la contra-firma.

```
public final void setTargetSigner(byte[ ] targetSignerParam)
```

Método que establece el firmante objetivo de la contra-firma. Como parámetros recibe:

- **targetSignerParam:** Parámetro que representa el firmante objetivo de la contra-firma.

```
public final byte[] getSignature()
```

Método que devuelve la firma a utilizar en la generación de la contra-firma.

```
public final void setSignature(byte[] signatureParam)
```

Método que establece la firma a utilizar en la generación de la contra-firma. Como parámetros recibe:

- **signatureParam:** Parámetro que representa la firma a utilizar en la generación de la contra-firma.

Clase DataInfo

Esta clase es un POJO que representa la información de los datos firmados por un firmante particular de una firma verificada. Está compuesta por los métodos:

```
public final byte[ ] getContentData()
```

Método que devuelve los datos originalmente firmados.

```
public final void setContentData(byte[ ] contentDataParam)
```

Método que establece los datos originalmente firmados. Como parámetros recibe:

- **contentDataParam:** Parámetro que representa los datos originalmente firmados.

```
public final List<String> getSignedDataRefs()
```

Método que devuelve la lista con la información de las referencias firmadas por el firmante.

```
public final void setSignedDataRefs(List<String> signedDataRefsParam)
```

Método que establece la lista con la información de las referencias firmadas por el firmante. Como parámetros recibe:

- **signedDataRefsParam:** Parámetro que representa la lista con la información de las referencias firmadas por el firmante.

```
public final DocumentHash getDocumentHash()
```

Método que devuelve el resumen de los datos originalmente firmados.

```
public final void setDocumentHash(DocumentHash documentHashParam)
```

Método que establece el resumen de los datos originalmente firmados. Como parámetros recibe:

- **documentHashParam:** Parámetro que representa el resumen de los datos originalmente firmados.

Clase Detail

Esta clase es un POJO que representa la información detallada de la respuesta de un servicio web. Está compuesta por los métodos:

```
public String getType()
```

Método que devuelve la URI que identifica la tarea de validación ejecutada.

```
public void setType(String typeParam)
```

Método que establece la URI que identifica la tarea de validación ejecutada. Como parámetros recibe:

- **typeParam:** Parámetro que representa la URI que identifica la tarea de validación ejecutada.

```
public String getCode()
```

Método que devuelve la URI que identifica el código asociado al resultado.

```
public void setCode(String codeParam)
```

Método que establece la URI que identifica el código asociado al resultado. Como parámetros recibe:

- **codeParam:** Parámetro que representa la URI que identifica el código asociado al resultado.

```
public String getMessage()
```

Método que devuelve el mensaje descriptivo asociado al resultado del proceso.

```
public void setMessage(String messageParam)
```

Método que establece el mensaje descriptivo asociado al resultado del proceso. Como parámetros recibe:

- **messageParam:** Parámetro que representa el mensaje descriptivo asociado al resultado del proceso.

Clase DetailLevelEnum

Esta clase es un POJO que representa los diferentes niveles de detalle para la respuesta de un servicio web. Está compuesta por los métodos:

```
public String getUri()
```

Método que devuelve la URI que identifica el nivel de detalle.

Clase DocumentHash

Esta clase es un POJO que representa el resumen de los datos originales usados para su verificación con respecto a los datos incluidos en una firma. Está compuesta por los métodos:

```
public final String getDigestMethod()
```

Método que devuelve el nombre del algoritmo de resumen.

```
public final void setDigestMethod(String digestMethodParam)
```

Método que establece el nombre del algoritmo de resumen. Como parámetros recibe:

- **digestMethodParam:** Parámetro que representa el nombre del algoritmo de resumen.

```
public final byte[ ] getDigestValue()
```

Método que devuelve el valor del resumen.

```
public final void setDigestValue(byte[ ] digestValueParam)
```

Método que establece el valor del resumen. Como parámetros recibe:

- **digestValueParam:** Parámetro que representa el valor del resumen.

```
public final TransformData getTransform()
```

Método que devuelve el valor del objeto TransformData para la canonicalización.

```
public final void setTransform(TransformData transformParam)
```

Método que establece el valor del objeto TransformData. Como parámetros recibe:

- **transformParam:** Parámetro que representa el valor del objeto TransformData para la canonicalización.

Clase DocumentRequest

Esta clase es un POJO que representa una petición al servicio de @Firma que permite almacenar un documento. Está compuesta por los métodos:

```
public final String getApplicationId()
```

Método que devuelve el identificador de la aplicación cliente.

```
public final void setApplicationId(String applicationIdParam)
```

Método que establece el identificador de la aplicación cliente. Como parámetros recibe:

- **applicationIdParam:** Parámetro que representa el identificador de la aplicación cliente.

```
public final byte[ ] getDocument()
```

Método que devuelve el documento.

```
public final void setDocument(byte[ ] documentParam)
```

Método que establece el documento. Como parámetros recibe:

- **documentParam:** Parámetro que representa el documento.

```
public final String getName()
```

Método que devuelve el nombre del documento.

```
public final void setName(String nameParam)
```

Método que establece el nombre del documento. Como parámetros recibe:

- **nameParam:** Parámetro que representa el nombre del documento.

```
public final String getType()
```

Método que devuelve la extensión del documento.

```
public final void setType(String typeParam)
```

Método que establece la extensión del documento. Como parámetros recibe:

- **typeParam:** Parámetro que representa la extensión del documento.

Clase DocumentResponse

Esta clase es un POJO que representa una respuesta del servicio de @Firma que permite almacenar un documento. Está compuesta por los métodos:

```
public final boolean isState()
```

Método que devuelve un valor lógico que indica si la operación finalizó correctamente (verdadero) o no (falso).

```
public final void setState(boolean stateParam)
```

Método que establece si la operación finalizó correctamente (verdadero) o no (falso). Como parámetros recibe:

- **stateParam:** Parámetro que indica si la operación finalizó correctamente (verdadero) o no (falso).

```
public final String getDescription()
```

Método que devuelve la descripción del error o de la excepción producida durante el proceso.

```
public final void setDescription(String descriptionParam)
```

Método que establece la descripción del error o de la excepción producida durante el proceso. Como parámetros recibe:

- **descriptionParam:** Parámetro que representa la descripción del error o de la excepción producida durante el proceso.

```
public final String getDocumentId()
```

Método que devuelve el identificador asignado en la custodia del documento.

```
public final void setDocumentId(String documentIdParam)
```

Método que establece el identificador asignado en la custodia del documento. Como parámetros recibe:

- **documentIdParam:** Parámetro que representa el identificador asignado en la custodia del documento.

```
public final ErrorResponse getError()
```

Método que devuelve la información asociada al error que se haya producido durante el proceso.

```
public final void setError(ErrorResponse errorParam)
```

Método que establece la información asociada al error que se haya producido durante el proceso. Como parámetros recibe:

- **errorParam:** Parámetro que representa la información asociada al error que se haya producido durante el proceso.

Clase DocumentTypeEnum

Esta clase es un POJO que representa los diferentes tipos de documento admitidos para las peticiones a los servicios web de TS@. Está compuesta por los métodos:

```
public String getType()
```

Método que devuelve el tipo de documento.

Clase ErrorResponse

Esta clase es un POJO que contiene la información asociada a un error producido durante la invocación de un servicio nativo de @Firma. Está compuesta por los métodos:

```
public final String getCodeError()
```

Método que devuelve el código de error.

```
public final void setCodeError(String codeErrorParam)
```

Método que establece el código de error. Como parámetros recibe:

- **codeErrorParam:** Parámetro que representa el código de error.

```
public final String getDescription()
```

Método que devuelve la descripción del error.

```
public final void setDescription(String descriptionParam)
```

Método que establece la descripción del error. Como parámetros recibe:

- **descriptionParam:** Parámetro que representa la descripción del error.

Clase HashAlgorithmEnum

Esta clase es un POJO que representa los diferentes algoritmos de hash admitidos para las peticiones a los servicios web de @Firma. Está compuesta por los métodos:

```
public String getUri()
```

Método que devuelve la URI que identifica el algoritmo de hash.

Clase IndividualSignatureReport

Esta clase es un POJO que contiene la información detallada sobre el procesamiento de una firma concreta contenida en la firma original. Está compuesta por los métodos:

```
public final Result getResult()
```

Método que devuelve el resultado del procesamiento de una firma en particular.

```
public final void setResult(Result resultParam)
```

Método que establece el resultado del procesamiento de una firma en particular. Como parámetros recibe:

- **resultParam:** Parámetro que representa el resultado del procesamiento de una firma en particular.

```
public final Map<String, Object> getReadableCertificateInfo()
```

Método que devuelve el mapa con la información asociada al certificado firmante.

```
public final void setReadableCertificateInfo(Map<String, Object>
readableCertificateInfoParam)
```

Método que establece el mapa con la información asociada al certificado firmante. Como parámetros recibe:

- **readableCertificateInfoParam:** Parámetro que representa el mapa con la información asociada al certificado firmante.

```
public final String getSignaturePolicyIdentifier()
```

Método que devuelve el identificador de la política de firma asociada a la firma.

```
public final void setSignaturePolicyIdentifier(String
signaturePolicyIdentifierParam)
```

Método que establece el identificador de la política de firma asociada a la firma. Como parámetros recibe:

- **signaturePolicyIdentifierParam:** Parámetro que representa el identificador de la política de firma asociada a la firma.

```
public final byte[ ] getSigPolicyDocument()
```

Método que devuelve el documento que especifica la política de firma asociada a la firma.

```
public final void setSigPolicyDocument(byte[ ] sigPolicyDocumentParam)
```

Método que establece el documento que especifica la política de firma asociada a la firma. Como parámetros recibe:

- **sigPolicyDocumentParam:** Parámetro que representa el documento que especifica la política de firma asociada a la firma.

```
public final ProcessingDetail getProcessingDetails()
```

Método que devuelve el resultado de los diferentes pasos involucrados en el proceso de verificación de la firma.

```
public final void setProcessingDetails(ProcessingDetail processingDetailsParam)
```

Método que establece el resultado de los diferentes pasos involucrados en el proceso de verificación de la firma. Como parámetros recibe:

- **processingDetailsParam:** Parámetro que representa el resultado de los diferentes pasos involucrados en el proceso de verificación de la firma.

```
public final String getDetailedReport()
```

Método que devuelve la información adicional solicitada en la petición.

```
public final void setDetailedReport(String detailedReportParam)
```

Método que establece la información adicional solicitada en la petición. Como parámetros recibe:

- **detailedReportParam:** Parámetro que representa la información adicional solicitada en la petición.

Clase InvalidAsyncResponse

Esta clase es un POJO que representa una respuesta de error procedente de los servicios asíncronos de validación de certificados y de firmas. Está compuesta por los métodos:

```
public final Result getResult()
```

Método que devuelve el resultado erróneo del proceso.

```
public final void setResult(Result resultParam)
```

Método que establece el resultado erróneo del proceso. Como parámetros recibe:

- **resultParam:** Parámetro que representa el resultado erróneo del proceso.

Clase OptionalParameters

Esta clase es un POJO que representa los parámetros adicionales que admiten las peticiones a los servicios web de @Firma para solicitar información adicional en la respuesta de dichos servicios. Está compuesta por los métodos:

```
public final boolean isReturnReadableCertificateInfo()
```

Método que devuelve un valor lógico que indica si se debe devolver información detallada de los certificados validados (verdadero) o no (falso).

```
public final void setReturnReadableCertificateInfo(boolean
returnReadableCertificateInfoParam)
```

Método que establece si se debe devolver información detallada de los certificados validados (verdadero) o no (falso). Como parámetros recibe:

- **returnReadableCertificateInfoParam:** Parámetro que indica si se debe devolver información detallada de los certificados validados (verdadero) o no (falso).

```
public final boolean isAdditionalReportOption()
```

Método que devuelve un valor lógico que indica si se debe incluir en la respuesta del servicio cierta información adicional sobre el proceso de validación, referente a atributos firmados y no firmados incluidos en la firma (verdadero), o no (falso).

```
public final void setAdditionalReportOption(boolean additionalReportOptionParam)
```

Método que establece si se debe incluir en la respuesta del servicio cierta información adicional sobre el proceso de validación, referente a atributos firmados y no firmados incluidos en la firma (verdadero), o no (falso). Como parámetros recibe:

- **additionalReportOptionParam:** Parámetro que indica si se debe incluir en la respuesta del servicio cierta información adicional sobre el proceso de validación, referente a atributos firmados y no firmados incluidos en la firma (verdadero), o no (falso).

```
public final boolean isReturnProcessingDetails()
```

Método que devuelve un valor lógico que indica si se debe incluir en la respuesta del servicio el resultado de cada tarea de verificación que forma el proceso de validación de la firma (verdadero), o no (falso).

```
public final void setReturnProcessingDetails(boolean returnProcessingDetailsParam)
```

Método que establece si se debe incluir en la respuesta del servicio el resultado de cada tarea de verificación que forma el proceso de validación de la firma (verdadero), o no (falso). Como parámetros recibe:

- **returnProcessingDetailsParam:** Parámetro que indica si se debe incluir en la respuesta del servicio el resultado de cada tarea de verificación que forma el proceso de validación de la firma (verdadero), o no (falso).

```
public final boolean isReturnSignedDataInfo()
```

Método que devuelve un valor lógico que indica si se debe incluir en la respuesta del servicio información acerca de los datos firmados (verdadero), o no (falso).

```
public final void setReturnSignedDataInfo(boolean returnSignedDataInfoParam)
```

Método que establece si se debe incluir en la respuesta del servicio información acerca de los datos firmados (verdadero), o no (falso). Como parámetros recibe:

- **returnSignedDataInfoParam:** Parámetro que indica si se debe incluir en la respuesta del servicio información acerca de los datos firmados (verdadero), o no (falso).

```
public final boolean isReturnSignPolicyDocument()
```

Método que devuelve un valor lógico que indica si se debe incluir en la respuesta del servicio el documento de la política de firma asociada (verdadero), o no (falso).

```
public final void setReturnSignPolicyDocument(boolean
returnSignPolicyDocumentParam)
```

Método que establece si se debe incluir en la respuesta del servicio el documento de la política de firma asociada (verdadero), o no (falso). Como parámetros recibe:

- **returnSignPolicyDocumentParam:** Parámetro que indica si se debe incluir en la respuesta del servicio el documento de la política de firma asociada (verdadero), o no (falso).

```
public final String getCertificateValidationLevel()
```

Método que obtiene el nivel de validación requerido para el certificado.

```
public final void setCertificateValidationLevel(String
certificateValidationLevelParam)
```

Método que establece el nivel de validación del certificado. Como parámetros recibe:

- **certificateValidationLevelParam:** Parámetro que representa el nivel de validación requerido para el certificado.

```
public boolean isReturnNextUpdate()
```

Método que devuelve un valor lógico que indica si se debe incluir en la respuesta del servicio la fecha de experición de la firma (verdadero), o no (falso).

```
public void setReturnNextUpdate(boolean returnNextUpdateParam)
```

Método que establece si se debe incluir en la respuesta del servicio la fecha de expiración de la firma (verdadero), o no (falso). Como parámetros recibe:

- **returnNextUpdateParam:** Parámetro que indica si se debe incluir en la respuesta del servicio la fecha de expiración de la firma (verdadero), o no (falso).

```
public boolean isProcessAsNotBaseline()
```

Método que devuelve un valor lógico que indica si la petición debe ser procesada como no baseline (verdadero), o no (falso).

```
public void setProcessAsNotBaseline(boolean processAsNotBaseline)
```

Método que establece si la petición se debe procesar como no baseline (verdadero), o no (falso). Como parámetros recibe:

- **processAsNotBaseline:** Parámetro que indica si la petición se debe procesar como no baseline (verdadero), o no (falso).

Clase PendingRequest

Esta clase es un POJO que representa una petición de consulta de proceso asíncrono. Está compuesta por los métodos:

```
public final String getApplicationId()
```

Método que devuelve el identificador de la aplicación cliente.

```
public final void setApplicationId(String applicationIdParam)
```

Método que establece el identificador de la aplicación cliente. Como parámetros recibe:

- **applicationIdParam:** Parámetro que representa el identificador de la aplicación cliente.

```
public final String getResponseId()
```

Método que devuelve el identificador del proceso asíncrono.

```
public final void setResponseId(String responseIdParam)
```

Método que establece el identificador del proceso asíncrono. Como parámetros recibe:

- **responseIdParam:** Parámetro que representa el identificador del proceso asíncrono.

Clase ProcessingDetail

Esta clase es un POJO que representa el resultado de los distintos pasos que forman el proceso de verificación de una firma. Está compuesta por los métodos:

```
public List<Detail> getListValidDetail()
```

Método que devuelve la lista con las tareas de validación que han arrojado un resultado satisfactorio.

```
public void setListValidDetail(List<Detail> listValidDetailParam)
```

Método que establece la lista con las tareas de validación que han arrojado un resultado satisfactorio. Como parámetros recibe:

- **listValidDetailParam:** Parámetro que representa la lista con las tareas de validación que han arrojado un resultado satisfactorio.

```
public List<Detail> getListInvalidDetail()
```

Método que devuelve la lista con las tareas de validación que han arrojado un resultado no satisfactorio.

```
public void setListInvalidDetail(List<Detail> listInvalidDetailParam)
```

Método que establece la lista con las tareas de validación que han arrojado un resultado no satisfactorio. Como parámetros recibe:

- **listInvalidDetailParam:** Parámetro que representa la lista con las tareas de validación que han arrojado un resultado no satisfactorio.

```
public List<Detail> getListIndeterminateDetail()
```

Método que devuelve la lista con las tareas de validación que han arrojado un resultado indeterminado.

```
public void setListIndeterminateDetail(List<Detail> listIndeterminateDetailParam)
```

Método que establece la lista con las tareas de validación que han arrojado un resultado indeterminado. Como parámetros recibe:

- **listIndeterminateDetailParam:** Parámetro que representa la lista con las tareas de validación que han arrojado un resultado indeterminado.

Clase Repository

Esta clase es un POJO que representa un repositorio o gestor documental que almacena documentos y firmas. Está compuesta por los métodos:

```
public final String getId()
```

Método que devuelve el identificador del repositorio o gestor documental.

```
public final void setId(String idParam)
```

Método que establece el identificador del repositorio o gestor documental. Como parámetros recibe:

- **idParam:** Parámetro que representa el identificador del repositorio o gestor documental.

```
public final String getObject()
```

Método que devuelve el UUID del objeto almacenado en el repositorio o gestor documental.

```
public final void setObject(String objectParam)
```

Método que establece el UUID del objeto almacenado en el repositorio o gestor documental. Como parámetros recibe:

- **objectParam:** Parámetro que representa el UUID del objeto almacenado en el repositorio o gestor documental.

Clase Result

Esta clase es un POJO que representa la información asociada al resultado de un proceso. Está compuesta por los métodos:

```
public final String getResultMajor()
```

Método que devuelve la URI del resultado global del proceso.

```
public final void setResultMajor(String resultMajorParam)
```

Método que establece la URI del resultado global del proceso. Como parámetros recibe:

- **resultMajorParam:** Parámetro que representa la URI del resultado global del proceso.

```
public final String getResultMinor()
```

Método que devuelve la URI del resultado concreto.

```
public final void setResultMinor(String resultMinorParam)
```

Método que establece la URI del resultado concreto. Como parámetros recibe:

- **resultMinorParam:** Parámetro que representa la URI del resultado concreto.

```
public final String getResultMessage()
```

Método que devuelve el mensaje descriptivo del resultado del proceso.

```
public final void setResultMessage(String resultMessageParam)
```

Método que establece el mensaje descriptivo del resultado del proceso. Como parámetros recibe:

- **resultMessageParam:** Parámetro que representa el mensaje descriptivo del resultado del proceso.

Clase ServerSignerRequest

Esta clase es un POJO que representa una petición al servicio de @Firma que permite generar una firma. Está compuesta por los métodos:

```
public final byte[ ] getDocument()
```

Método que devuelve los datos a firmar.

```
public final void setDocument(byte[ ] documentParam)
```

Método que establece los datos a firmar. Como parámetros recibe:

- **documentParam:** Parámetro que representa los datos a firmar.

```
public final String getDocumentId()
```

Método que devuelve el identificador único del documento a firmar.

```
public final void setDocumentId(String documentIdParam)
```

Método que establece el identificador único del documento a firmar. Como parámetros recibe:

- **documentIdParam:** Parámetro que representa el identificador único del documento a firmar.

```
public final Repository getDocumentRepository()
```

Método que devuelve la localización del documento a firmar dentro de un repositorio o gestor de documentos.

```
public final void setDocumentRepository(Repository documentRepositoryParam)
```

Método que establece la localización del documento a firmar dentro de un repositorio o gestor de documentos. Como parámetros recibe:

- **documentRepositoryParam:** Parámetro que representa la localización del documento a firmar dentro de un repositorio o gestor de documentos.

```
public final String getApplicationId()
```

Método que devuelve el identificador de la aplicación cliente.

```
public final void setApplicationId(String applicationIdParam)
```

Método que establece el identificador de la aplicación cliente. Como parámetros recibe:

- **applicationIdParam:** Parámetro que representa el identificador de la aplicación cliente.

```
public final String getKeySelector()
```

Método que devuelve el identificador de la clave usada para generar la firma delegada.

```
public final void setKeySelector(String keySelectorParam)
```

Método que establece el identificador de la clave usada para generar la firma delegada. Como parámetros recibe:

- **keySelectorParam:** Parámetro que representa el identificador de la clave usada para generar la firma delegada.

```
public final SignatureFormatEnum getSignatureFormat()
```

Método que devuelve el formato de la firma a generar.

```
public final void setSignatureFormat(SignatureFormatEnum signatureFormatParam)
```

Método que establece el formato de la firma a generar. Como parámetros recibe:

- **signatureFormatParam:** Parámetro que representa el formato de la firma a generar.

```
public final HashAlgorithmEnum getHashAlgorithm()
```

Método que devuelve el algoritmo de hash definido para calcular la firma.

```
public final void setHashAlgorithm(HashAlgorithmEnum hashAlgorithmParam)
```

Método que establece el algoritmo de hash definido para calcular la firma. Como parámetros recibe:

- **hashAlgorithmParam:** Parámetro que representa el algoritmo de hash definido para calcular la firma.

```
public final XmlSignatureModeEnum getXmlSignatureMode()
```

Método que devuelve el modo de firma en formato XML.

```
public final void setXmlSignatureMode(XmlSignatureModeEnum xmlSignatureModeParam)
```

Método que establece el modo de firma en formato XML. Como parámetros recibe:

- **xmlSignatureModeParam:** Parámetro que representa el modo de firma en formato XML.

```
public final String getSignaturePolicyIdentifier()
```

Método que devuelve el identificador de la política de firma asociada.

```
public final void setSignaturePolicyIdentifier(String
signaturePolicyIdentifierParam)
```

Método que establece el identificador de la política de firma asociada. Como parámetros recibe:

- **signaturePolicyIdentifierParam:** Parámetro que representa el identificador de la política de firma asociada.

```
public final boolean isIgnoreGracePeriod()
```

Método que devuelve un valor lógico que indica si se debe ignorar el periodo de gracia (verdadero) o no (falso).

```
public final void setSignaturePolicyIdentifier(String
signaturePolicyIdentifierParam)
```

Método que establece si se debe ignorar el periodo de gracia (verdadero) o no (falso). Como parámetros recibe:

- **signaturePolicyIdentifierParam:** Parámetro que indica si se debe ignorar el periodo de gracia (verdadero) o no (falso).

```
public final DocumentHash getDocumentHash()
```

Método que devuelve el hash del documento a firmar.

```
public final void setDocumentHash(DocumentHash documentHashParam)
```

Método que establece el hash del documento a firmar. Como parámetros recibe:

- **documentHashParam:** Parámetro que representa el identificador de la política de firma asociada.

Clase ServerSignerResponse

Esta clase es un POJO que representa una respuesta de los servicios de @Firma que permiten generar o actualizar una firma. Está compuesta por los métodos:

```
public final Result getResult()
```

Método que devuelve el resultado del proceso.

```
public final void setResult(Result resultParam)
```

Método que establece el resultado del proceso. Como parámetros recibe:

- **resultParam:** Parámetro que representa el resultado del proceso.

```
public final String getSignatureFormat()
```

Método que devuelve el formato de la firma generada o actualizada.

```
public final void setSignatureFormat(String signatureFormatParam)
```

Método que establece el formato de la firma generada o actualizada. Como parámetros recibe:

- **signatureFormatParam:** Parámetro que representa el formato de la firma generada o actualizada.

```
public final String getIdTransaction()
```

Método que devuelve el identificador único de la transacción generada.

```
public final void setTransactionId(String transactionIdParam)
```

Método que establece el identificador único de la transacción generada. Como parámetros recibe:

- **transactionIdParam:** Parámetro que representa el identificador único de la transacción generada.

```
public final String getAsyncResponse()
```

Método que devuelve el identificador de proceso asíncrono que identifica la petición en el caso de que se haya realizado una petición de generación o actualización de firma con periodo de gracia.

```
public final void setAsyncResponse(String asyncResponseParam)
```

Método que establece el identificador de proceso asíncrono que identifica la petición en el caso de que se haya realizado una petición de generación o actualización de firma con periodo de gracia. Como parámetros recibe:

- **asyncResponseParam:** Parámetro que representa el identificador de proceso asíncrono que identifica la petición en el caso de que se haya realizado una petición de generación o actualización de firma con periodo de gracia.

```
public final byte[ ] getSignature()
```

Método que devuelve la firma generada, en el caso de que se haya realizado una petición de generación de firma.

```
public final void setSignature(byte[ ] signatureParam)
```

Método que establece la firma generada, en el caso de que se haya realizado una petición de generación de firma. Como parámetros recibe:

- **signatureParam:** Parámetro que representa la firma generada, en el caso de que se haya realizado una petición de generación de firma.

```
public final byte[ ] getUpdatedSignature()
```

Método que devuelve la firma actualizada, en el caso de que se haya realizado una petición de actualización de firma.

```
public final void setUpdatedSignature(byte[ ] updatedSignatureParam)
```

Método que establece la firma actualizada, en el caso de que se haya realizado una petición de actualización de firma. Como parámetros recibe:

- **updatedSignatureParam:** Parámetro que representa la firma actualizada, en el caso de que se haya realizado una petición de actualización de firma.

Clase SignatureFormatEnum

Esta clase es un POJO que representa los diferentes formatos de firma admitidos por @Firma. Está compuesta por los métodos:

```
public String getUriType()
```

Método que devuelve la URI que identifica el tipo de la firma.

```
public String getUriFormat()
```

Método que devuelve la URI que identifica la forma de la firma.

Clase SignatureTransactionResponse

Esta clase es un POJO que representa una respuesta del servicio de @Firma que permite obtener una firma a partir de su identificador de transacción. Está compuesta por los métodos:

```
public final boolean isState()
```

Método que devuelve un valor lógico que indica si la operación finalizó correctamente (verdadero) o no (falso).

```
public final void setState(boolean stateParam)
```

Método que establece si la operación finalizó correctamente (verdadero) o no (falso). Como parámetros recibe:

- **stateParam:** Parámetro que indica si la operación finalizó correctamente (verdadero) o no (falso).

```
public final String getDescription()
```

Método que devuelve la descripción del error o de la excepción producida durante el proceso.

```
public final void setDescription(String descriptionParam)
```

Método que establece la descripción del error o de la excepción producida durante el proceso. Como parámetros recibe:

- **descriptionParam:** Parámetro que representa la descripción del error o de la excepción producida durante el proceso.

```
public final byte[ ] getSignature()
```

Método que devuelve la firma obtenida.

```
public final void setSignature(byte[ ] signatureParam)
```

Método que establece la firma obtenida. Como parámetros recibe:

- **signatureParam:** Parámetro que representa la firma obtenida.

```
public final String getSignatureFormat()
```

Método que devuelve el formato de la firma generada o actualizada.

```
public final void setSignatureFormat(String signatureFormatParam)
```

Método que establece el formato de la firma generada o actualizada. Como parámetros recibe:

- **signatureFormatParam:** Parámetro que representa el formato de la firma generada o actualizada.

```
public final ErrorResponse getError()
```

Método que devuelve la información asociada al error que se haya producido durante el proceso.

```
public final void setError(ErrorResponse errorParam)
```

Método que establece la información asociada al error que se haya producido durante el proceso. Como parámetros recibe:

- **errorParam:** Parámetro que representa la información asociada al error que se haya producido durante el proceso.

Clase TimestampRequest

Esta clase es un POJO que representa una petición al servicio de TS@ que permite generar un sello de tiempo. Está compuesta por los métodos:

```
public final TimestampTypeEnum getTimestampType()
```

Método que devuelve el tipo de sello de tiempo que vamos a generar. Podrá ser "XML" o "RFC 3161".

```
public final void setTimestampType(TimestampTypeEnum timestampTypeParam)
```

Método que establece el tipo de sello de tiempo a generar. Como parámetros recibe:

- **timestampTypeParam:** Parámetro que representa el tipo de sello de tiempo.

```
public final byte[ ] getDataToStamp()
```

Método que devuelve los datos del fichero al que vamos a generar el sello de tiempo.

```
public final void setDataToStamp(byte[ ] dataToStampParam)
```

Método que establece los datos del fichero al que vamos a generar el sello de tiempo. Como parámetros recibe:

- **dataToStampParam:** Parámetro que representa los datos del fichero.

```
public final String getApplicationId()
```

Método que devuelve el identificador de la aplicación cliente.

```
public final void setApplicationId(String applicationIdParam)
```

Método que establece el identificador de la aplicación cliente. Como parámetros recibe:

- **applicationIdParam:** Parámetro que representa el identificador de la aplicación cliente.

```
public final DocumentHash getDocumentHash()
```

Método que devuelve el hash del documento a generar el sello de tiempo.

```
public final void setDocumentHash(DocumentHash documentHashParam)
```

Método que establece el hash del documento a generar el sello de tiempo. Como parámetros recibe:

- **documentHashParam:** Parámetro que representa el identificador de la política de firma asociada.

```
public final TransformData getTransformData()
```

Método que devuelve el valor del objeto TransformData para la canonicalización.

```
public final void setTransformData(TransformData transformDataParam)
```

Método que establece el valor del objeto TransformData. Como parámetros recibe:

- **transformDataParam:** Parámetro que representa el valor del objeto TransformData para la canonicalización.

```
public final DocumentTypeEnum getDocumentType()
```

Método que devuelve el tipo de documento al que vamos a generar el sello de tiempo. Podrá ser: "DocumentHash", "DocumentHashTransformedData", "TransformedData", "Base64Data", "Base64XML", "InlineXML" y "EscapedXML".

```
public final void setDocumentType(DocumentTypeEnum documentTypeParam)
```

Método que establece el tipo de documento. Como parámetros recibe:

- **documentTypeParam:** Parámetro que representa el tipo documento.

```
public final byte[ ] getTimestampTimestampToken()
```

Método que devuelve los datos del sello de tiempo que vamos a validar.

```
public final void setTimestampTimestampToken(byte[ ] timestampTimestampTokenParam)
```

Método que establece los datos del sello de tiempo que vamos a validar. Como parámetro recibe:

- **timestampTimestampTokenParam:** Parámetro que representa los datos del sello de tiempo.

```
public final byte[ ] getTimestampPreviousTimestampToken()
```

Método que devuelve los datos del sello de tiempo al que vamos a hacer el resellado.

```
public final void setTimestampPreviousTimestampToken(byte[ ]
timestampPreviousTimestampTokenParam)
```

Método que establece los datos del sello de tiempo al que vamos a realizar el resellado. Como parámetro recibe:

- **timestampPreviousTimestampTokenParam:** Parámetro que representa los datos del sello de tiempo.

Clase TimestampResponse

Esta clase es un POJO que representa una respuesta de los servicios de TS@ que permiten generar, validar o renovar un sello de tiempo. Está compuesta por los métodos:

```
public final Result getResult()
```

Método que devuelve el resultado del proceso.

```
public final void setResult(Result resultParam)
```

Método que establece el resultado del proceso. Como parámetros recibe:

- **resultParam:** Parámetro que representa el resultado del proceso.

```
public final byte[ ] getTimestamp()
```

Método que devuelve el sello de tiempo generado, en el caso de que se haya realizado una petición de generación de sello de tiempo o de renovación.

```
public final void setTimestamp(byte[ ] timestampParam)
```

Método que establece el sello de tiempo generado, en el caso de que se haya realizado una petición de generación o renovación de sello de tiempo. Como parámetros recibe:

- **timestampParam:** Parámetro que representa el sello de tiempo generado.

Clase TimestampTypeEnum

Esta clase es un POJO que representa los diferentes tipos de sello de tiempo admitidos para las peticiones a los servicios web de TS@. Está compuesta por los métodos:

```
public String getType()
```

Método que devuelve el tipo de sello de tiempo.

Clase TransformData

Esta clase es un POJO que representa los diferentes métodos de canonicalización que puede sufrir un fichero a firmar. Está compuesta por los métodos:

```
public TransformData(String algorithm)
```

Método que devuelve un objeto TransformData donde el algoritmo de codificación usado viene definido por el parámetro de entrada. Como parámetro recibe:

- **algorithm:** Parámetro que indica el algoritmo de codificación usado para los métodos de la canonicalización.

```
public TransformData(String algorithm, List<String> xpathList)
```

Método que devuelve un objeto TransformData donde el algoritmo de codificación usado viene definido por el parámetro de entrada Como parámetros recibe:

- **algorithm:** Parámetro que indica el algoritmo de codificación usado para los métodos de la canonicalización.
- **xpathList:** Parámetro que muestra la lista de algoritmos de canonicalización que se aplican al fichero indicado.

```
public final String getAlgorithm()
```

Método que devuelve el algoritmo de codificación usado para los métodos de la canonicalización.

```
public final List<String> getXPath()
```

Método que devuelve la lista de algoritmos de canonicalización que se aplican al fichero indicado

Clase UpgradeSignatureRequest

Esta clase es un POJO que representa una petición al servicio de @Firma que permite actualizar una firma. Está compuesta por los métodos:

```
public final byte[ ] getSignature()
```

Método que devuelve la firma a actualizar.

```
public final void setSignature(byte[ ] signatureParam)
```

Método que establece la firma a actualizar. Como parámetros recibe:

- **signatureParam:** Parámetro que representa la firma a actualizar.

```
public final Repository getSignatureRepository()
```

Método que devuelve la ubicación en un gestor de documentos o repositorio de la firma a actualizar.

```
public final void setSignatureRepository(Repository signatureRepositoryParam)
```

Método que establece la ubicación en un gestor de documentos o repositorio de la firma a actualizar. Como parámetros recibe:

- **signatureRepositoryParam:** Parámetro que representa la ubicación en un gestor de documentos o repositorio de la firma a actualizar.

```
public final String getTransactionId()
```

Método que devuelve el identificador de transacción asociado a la generación de la firma que actualizar.

```
public final void setTransactionId(String transactionIdParam)
```

Método que establece el identificador de transacción asociado a la generación de la firma que actualizar. Como parámetros recibe:

- **transactionIdParam:** Parámetro que representa el identificador de transacción asociado a la generación de la firma que actualizar.

```
public final SignatureFormatEnum getSignatureFormat()
```

Método que devuelve el formato al que actualizar la firma.

```
public final void setSignatureFormat(SignatureFormatEnum signatureFormatParam)
```

Método que establece el formato al que actualizar la firma. Como parámetros recibe:

- **signatureFormatParam:** Parámetro que representa el formato al que actualizar la firma.

```
public final byte[ ] getTargetSigner()
```

Método que devuelve el firmante objetivo de la actualización.

```
public final void setTargetSigner(byte[ ] targetSignerParam)
```

Método que establece el firmante objetivo de la actualización. Como parámetros recibe:

- **targetSignerParam:** Parámetro que representa el firmante objetivo de la actualización.

```
public final boolean isIgnoreGracePeriod()
```

Método que devuelve un valor lógico que indica si se debe ignorar el periodo de gracia (verdadero) o no (falso).

```
public final void setIgnoreGracePeriod(boolean ignoreGracePeriodParam)
```

Método que establece el valor de la bandera que indica si se debe ignorar el periodo de gracia. Como parámetros recibe:

- **ignoreGracePeriodParam:** Parámetro que representa el valor booleano que indica si se debe ignorar el periodo de gracia.

```
public final void setSignaturePolicyIdentifier(String
signaturePolicyIdentifierParam)
```

Método que establece si se debe ignorar el periodo de gracia (verdadero) o no (falso). Como parámetros recibe:

- **signaturePolicyIdentifierParam:** Parámetro que indica si se debe ignorar el periodo de gracia (verdadero) o no (falso).

```
public final String getApplicationId()
```

Método que devuelve el identificador de la aplicación cliente.

```
public final void setApplicationId(String applicationIdParam)
```

Método que establece el identificador de la aplicación cliente. Como parámetros recibe:

- **applicationIdParam:** Parámetro que representa el identificador de la aplicación cliente.

```
public boolean isProcessAsNotBaseline()
```

Método que devuelve un valor lógico que indica si se debe procesar la petición como no baseline (verdadero) o no (falso).

```
public void setProcessAsNotBaseline(boolean processAsNotBaseline)
```

Método que establece el valor de la bandera que indica si la petición se debe procesar como no baseline (por defecto será falso). Como parámetros recibe:

- **processAsNotBaseline:** Parámetro que representa el booleano que indica si la petición se debe procesar como no baseline.

Clase VerificationReport

Esta clase es un POJO que representa el resultado e información adicional que se desea incluir en las respuestas generadas durante el proceso de verificación de una firma o de un certificado. Está compuesta por los métodos:

```
public final Boolean getCheckCertificateStatus()
```

Método que devuelve un valor lógico que indica si se desea verificar el estado de revocación del certificado (verdadero), o no (falso).

```
public final void setCheckCertificateStatus(Boolean checkCertificateStatusParam)
```

Método que establece si se desea verificar el estado de revocación del certificado (verdadero), o no (falso). Como parámetros recibe:

- **checkCertificateStatusParam:** Parámetro que indica si se desea verificar el estado de revocación del certificado (verdadero), o no (falso).

```
public final Boolean getIncludeCertificateValues()
```

Método que devuelve un valor lógico que indica si se desea que la respuesta incluya los certificados validados (verdadero), o no (falso).

```
public final void setIncludeCertificateValues(Boolean
includeCertificateValuesParam)
```

Método que establece si se desea que la respuesta incluya los certificados validados (verdadero), o no (falso). Como parámetros recibe:

- **includeCertificateValuesParam:** Parámetro que indica si se desea que la respuesta incluya los certificados validados (verdadero), o no (falso).

```
public final Boolean getIncludeRevocationValues()
```

Método que devuelve un valor lógico que indica si se desea que la respuesta incluya los elementos de consulta de estado de revocación CRL u OCSP utilizados en la validación de los certificados que forman la cadena validada (verdadero), o no (falso).

```
public final void setIncludeRevocationValues(Boolean includeRevocationValuesParam)
```

Método que establece si se desea que la respuesta incluya los elementos de consulta de estado de revocación CRL u OCSP utilizados en la validación de los certificados que forman la cadena validada (verdadero), o no (falso). Como parámetros recibe:

- **includeRevocationValuesParam:** Parámetro que indica si se desea que la respuesta incluya los elementos de consulta de estado de revocación CRL u OCSP utilizados en la validación de los certificados que forman la cadena validada (verdadero), o no (falso).

```
public final DetailLevelEnum getReportDetailLevel()
```

Método que devuelve el nivel de detalle que se desea obtener en la respuesta del servicio.

```
public final void setReportDetailLevel(DetailLevelEnum reportDetailLevelParam)
```

Método que establece el nivel de detalle que se desea obtener en la respuesta del servicio. Como parámetros recibe:

- **reportDetailLevelParam:** Parámetro que representa el nivel de detalle que se desea obtener en la respuesta del servicio.

Clase VerifyCertificateRequest

Esta clase es un POJO que representa una petición al servicio de @Firma que permite validar un certificado. Está compuesta por los métodos:

```
public final byte[ ] getCertificate()
```

Método que devuelve el certificado a validar.

```
public final void setCertificate(byte[ ] certificateParam)
```

Método que establece el certificado a validar. Como parámetros recibe:

- **certificateParam:** Parámetro que representa el certificado a validar.

```
public final Repository getCertificateRepository()
```

Método que devuelve la ubicación del certificado a validar en un gestor de documentos o repositorio.

```
public final void setCertificateRepository(Repository certificateRepositoryParam)
```

Método que establece la ubicación del certificado a validar en un gestor de documentos o repositorio. Como parámetros recibe:

- **certificateRepositoryParam:** Parámetro que representa la ubicación del certificado a validar en un gestor de documentos o repositorio.

```
public final String getApplicationId()
```

Método que devuelve el identificador de la aplicación cliente.

```
public final void setApplicationId(String applicationIdParam)
```

Método que establece el identificador de la aplicación cliente. Como parámetros recibe:

- **applicationIdParam:** Parámetro que representa el identificador de la aplicación cliente.

```
public final Boolean getReturnReadableCertificateInfo()
```

Método que devuelve un valor lógico que indica si se solicitará información acerca de campos del certificado en la petición (verdadero), o no (falso).

```
public final void setReturnReadableCertificateInfo(Boolean
returnReadableCertificateInfoParam)
```

Método que establece si se solicitará información acerca de campos del certificado en la petición (verdadero), o no (falso). Como parámetros recibe:

- **returnReadableCertificateInfoParam:** Parámetro que indica si se solicitará información acerca de campos del certificado en la petición (verdadero), o no (falso).

```
public final VerificationReport getReturnVerificationReport()
```

Método que devuelve las validaciones a realizar sobre el certificado, así como la información que debe ser devuelta en la respuesta.

```
public final void setReturnVerificationReport(VerificationReport
returnVerificationReportParam)
```

Método que establece las validaciones a realizar sobre el certificado, así como la información que debe ser devuelta en la respuesta. Como parámetros recibe:

- **returnVerificationReportParam:** Parámetro que representa las validaciones a realizar sobre el certificado, así como la información que debe ser devuelta en la respuesta.

Clase VerifyCertificateResponse

Esta clase es un POJO que representa una respuesta del servicio de @Firma que permite validar un certificado. Está compuesta por los métodos:

```
public final Result getResult()
```

Método que devuelve el resultado del proceso.

```
public final void setResult(Result resultParam)
```

Método que establece el resultado del proceso. Como parámetros recibe:

- **resultParam:** Parámetro que representa el resultado del proceso.

```
public final Map<String, String> getReadableCertificateInfo()
```

Método que devuelve la información detallada del certificado firmante. Esta información será un mapa de campos del tipo atributo/valor con el resultado de haber procesado el certificado según la configuración del sistema.

```
public final void setReadableCertificateInfo(Map<String, String>
readableCertificateInfoParam)
```

Método que establece la información detallada del certificado firmante. Esta información será un mapa de campos del tipo atributo/valor con el resultado de haber procesado el certificado según la configuración del sistema. Como parámetros recibe:

- **readableCertificateInfoParam:** Parámetro que representa la información detallada del certificado firmante. Esta información será un mapa de campos del tipo atributo/valor con el resultado de haber procesado el certificado según la configuración del sistema.

```
public final CertificatePathValidity getCertificatePathValidity()
```

Método que devuelve la información asociada a la validación del certificado firmante.

```
public final void setCertificatePathValidity(CertificatePathValidity
certificatePathValidityParam)
```

Método que establece la información asociada a la validación del certificado firmante. Como parámetros recibe:

- **certificatePathValidityParam:** Parámetro que representa la información asociada a la validación del certificado firmante.

Clase VerifySignatureRequest

Esta clase es un POJO que representa una petición al servicio de @Firma que permite validar una firma. Está compuesta por los métodos:

```
public final byte[ ] getDocument()
```

Método que devuelve el documento original que fue firmado.

```
public final void setDocument(byte[ ] documentParam)
```

Método que establece el documento original que fue firmado. Como parámetros recibe:

- **documentParam:** Parámetro que representa el documento original que fue firmado.

```
public final DocumentHash getDocumentHash()
```

Método que devuelve el hash del documento original que fue firmado.

```
public final void setDocumentHash(DocumentHash documentHashParam)
```

Método que establece el hash del documento original que fue firmado. Como parámetros recibe:

- **documentHashParam:** Parámetro que representa el hash del documento original que fue firmado.

```
public final Repository getDocumentRepository()
```

Método que devuelve la ubicación del documento original en un gestor de documentos o repositorio.

```
public final void setDocumentRepository(Repository documentRepositoryParam)
```

Método que establece la ubicación del documento original en un gestor de documentos o repositorio. Como parámetros recibe:

- **documentRepositoryParam:** Parámetro que representa la ubicación del documento original en un gestor de documentos o repositorio.

```
public final byte[ ] getSignature()
```

Método que devuelve la firma a validar.

```
public final void setSignature(byte[ ] signatureParam)
```

Método que establece la firma a validar. Como parámetros recibe:

- **signatureParam:** Parámetro que representa la firma a validar.

```
public final Repository getSignatureRepository()
```

Método que devuelve la ubicación de la firma que validar en un gestor de documentos o repositorio.

```
public final void setSignatureRepository(Repository signatureRepositoryParam)
```

Método que establece la ubicación de la firma que validar en un gestor de documentos o repositorio. Como parámetros recibe:

- **signatureRepositoryParam:** Parámetro que representa la ubicación de la firma que validar en un gestor de documentos o repositorio.

```
public final String getApplicationId()
```

Método que devuelve el identificador de la aplicación cliente.

```
public final void setApplicationId(String applicationIdParam)
```

Método que establece el identificador de la aplicación cliente. Como parámetros recibe:

- **applicationIdParam:** Parámetro que representa el identificador de la aplicación cliente.

```
public final VerificationReport getVerificationReport()
```

Método que devuelve el resultado e información adicional generada durante el proceso de verificación de la firma.

```
public final void setVerificationReport(VerificationReport
returnVerificationReportParam)
```

Método que establece el resultado e información adicional generada durante el proceso de verificación de la firma. Como parámetros recibe:

- **returnVerificationReportParam:** Parámetro que representa el resultado e información adicional generada durante el proceso de verificación de la firma.

```
public final OptionalParameters getOptionalParameters()
```

Método que devuelve el conjunto de parámetros adicionales que incluir en la petición.

```
public final void setOptionalParameters(OptionalParameters
optionalParametersParam)
```

Método que establece el conjunto de parámetros adicionales que incluir en la petición. Como parámetros recibe:

- **optionalParametersParam:** Parámetro que representa el conjunto de parámetros adicionales que incluir en la petición.

Clase VerifySignatureResponse

Esta clase es un POJO que representa una respuesta del servicio de @Firma que permite validar una firma. Está compuesta por los métodos:

```
public final Result getResult()
```

Método que devuelve el resultado del proceso.

```
public final void setResult(Result resultParam)
```

Método que establece el resultado del proceso. Como parámetros recibe:

- **resultParam:** Parámetro que representa el resultado del proceso.

```
public final List<IndividualSignatureReport> getVerificationReport()
```

Método que devuelve la lista con la información detallada sobre el procesamiento de cada firma contenida en la firma original.

```
public final void setVerificationReport(List<IndividualSignatureReport>
verificationReportParam)
```

Método que establece la lista con la información detallada sobre el procesamiento de cada firma contenida en la firma original. Como parámetros recibe:

- **verificationReportParam:** Parámetro que representa la lista con la información detallada sobre el procesamiento de cada firma contenida en la firma original.

```
public final String getSignatureFormat()
```

Método que devuelve el formato de la firma validada.

```
public final void setSignatureFormat(String signatureFormatParam)
```

Método que establece el formato de la firma validada. Como parámetros recibe:

- **signatureFormatParam:** Parámetro que representa el formato de la firma validada.

```
public final List<DataInfo> getSignedDataInfo()
```

Método que devuelve la lista con los datos firmados de por cada uno de los firmantes contenidos en la firma.

```
public final void setSignedDataInfo(List<DataInfo> signedDataInfoParam)
```

Método que establece la lista con los datos firmados de por cada uno de los firmantes contenidos en la firma. Como parámetros recibe:

- **signedDataInfoParam:** Parámetro que representa la lista con los datos firmados de por cada uno de los firmantes contenidos en la firma.

Clase XmlSignatureModeEnum

Esta clase es un POJO que representa los diferentes modos de firma para la petición de una firma servidor. Está compuesta por los métodos:

```
public String getMode()
```

Método que devuelve el modo de firma.

9.2.5 Paquete es.gob.afirma.tsaServiceInvoker

Este paquete contiene clases que permiten la comunicación con los WS de TS@. Para poder hacer uso de los métodos definidos en este paquete debe estar configurado correctamente los ficheros **tsaXXXXX.properties e integra properties**. De este paquete se describirán aquellas clases más destacadas.

Clase TSAServiceInvokerFacade

Esta clase implementa el patrón fachada y permite independizar la invocación de los servicios publicados en la plataforma TS@. Está compuesta por los métodos:

```
public String invokeService(String xmlInput, String service, String method, String
applicationName) throws Afirma5ServiceInvokerException
```

Método que permite invocar a servicios de TS@. Devuelve una cadena de texto en formato XML con la respuesta ofrecida por el servicio. En caso de que se produzca algún error durante el proceso, el método lanzará una excepción *Afirma5ServiceInvokerException*. Como parámetros recibe:

- **xmlInput:** Parámetro que representa la petición en formato XML.
- **service:** Parámetro que representa el nombre del servicio que invocar. Los valores permitidos son:
  - **CreateTimeStampWS** → Servicio de Generación de Sello de Tiempo de TS@.
  - **RenewTimeStampWS** → Servicio de Renovación de Sello de Tiempo de TS@.
  - **VerifyTimeStampWS** → Servicio de Validación de Sello de Sello de Tiempo de TS@.
- **method:** Parámetro que representa el nombre del método asociado al servicio que invocar.
- **applicationName:** Parámetro que representa el identificador de la aplicación cliente que lleva a cabo la petición.

9.2.6 Paquete es.gob.afirma.utils

Este paquete contiene clases que representan utilidades, así como interfaces con constantes. De este paquete se describirán aquellas clases más destacadas. Teniendo en cuenta que para este tipo de integración no funcionarán todas las clases y métodos del paquete es.gob.afirma.utils al no necesitarse inluir ciertas librerías de terceros innecesarias para este tipo de integración.

Clase Base64Coder

Esta clase define métodos que permiten la codificación y decodificación de datos en Base 64. Está compuesta por los métodos:

```
public static byte[ ] encodeBase64(byte[ ] data) throws TransformersException
```

Método que permite codificar en Base 64 una colección de bytes. Devuelve una colección de bytes codificados en Base 64. En caso de que se produzca algún error durante el proceso, el método lanzará una excepción *TransformersException*. Como parámetros recibe:

- **data:** Parámetro que representa la colección de bytes a codificar en Base 64.

```
public static byte[ ] encodeBase64(byte[ ] data, int offset, int len) throws
TransformersException
```

Método que permite codificar en Base 64 una colección de bytes, indicando la posición inicial y final de los bytes a codificar. Devuelve una colección de bytes codificados en Base 64. En caso de que se produzca algún error durante el proceso, el método lanzará una excepción *TransformersException*. Como parámetros recibe:

- **data:** Parámetro que representa la colección de bytes a codificar en Base 64.
- **offset:** Parámetro que representa la posición dentro de la colección de bytes para iniciar la codificación en Base 64.
- **len:** Parámetro que representa la posición dentro de la colección de bytes para finalizar la codificación en Base 64.

```
public static byte[ ] decodeBase64(byte[ ] data) throws TransformersException
```

Método que permite decodificar una colección de bytes codificados en Base 64. Devuelve una colección de bytes decodificados en Base 64. En caso de que se produzca algún error durante el proceso, el método lanzará una excepción *TransformersException*. Como parámetros recibe:

- **data:** Parámetro que representa la colección de bytes codificados en Base 64 a decodificar.

```
public static byte[ ] decodeBase64(byte[ ] data, int offset, int len) throws
TransformersException
```

Método que permite decodificar una colección de bytes codificados en Base 64, indicando la posición inicial y final de los bytes a decodificar. Devuelve una colección de bytes decodificada en Base 64. En caso de que se produzca algún error durante el proceso, el método lanzará una excepción *TransformersException*. Como parámetros recibe:

- **data:** Parámetro que representa la colección de bytes codificados en Base 64.
- **offset:** Parámetro que representa la posición dentro de la colección de bytes para iniciar la decodificación en Base 64.
- **len:** Parámetro que representa la posición dentro de la colección de bytes para finalizar la decodificación en Base 64.

```
public static boolean isBase64Encoded(byte[ ] data) throws TransformersException
```

Método que indica si una colección de bytes se encuentran codificados en Base 64. Devuelve un valor lógico que indica si la colección de bytes está codificada en Base 64 (verdadero) o no (falso). En caso de que se produzca algún error durante el proceso, el método lanzará una excepción *TransformersException*. Como parámetros recibe:

- **data:** Parámetro que representa la colección de bytes a comprobar.

```
public static String encodeBase64(String data) throws TransformersException
```

Método que permite codificar en Base 64 una cadena de texto. Devuelve una cadena de texto codificada en Base 64. En caso de que se produzca algún error durante el proceso, el método lanzará una excepción *TransformersException.* Como parámetros recibe:

- **data:** Parámetro que representa la cadena de texto a codificar en Base 64.

```
public static String decodeBase64(String data) throws TransformersException
```

Método que permite decodificar una cadena de texto codificada en Base 64. Devuelve una cadena de texto decodificada en Base 64. En caso de que se produzca algún error durante el proceso, el método lanzará una excepción *TransformersException.* Como parámetros recibe:

- **data:** Parámetro que representa la cadena de texto codificada en Base 64.

Clase CryptoUtil

Esta clase define métodos asociados a operaciones criptográficas de cálculo de resumen de datos o con funciones de hash. Está compuesta por los métodos:

```
public static String translateAlgorithmIdentifier(AlgorithmIdentifier
algorithmIdentifier)
```

Método que obtiene el nombre de un algoritmo de hash a partir de su OID. Si no es posible obtener el nombre del algoritmo se devolverá un valor nulo. Como parámetros recibe:

- **algorithmIdentifier:** Parámetro que representa el OID del algoritmo de hash.

```
public static String translateXmlDigestAlgorithm(String digestAlg)
```

Método que obtiene el nombre de un algoritmo de hash a partir de su URI. En caso de que no sea posible obtener el nombre del algoritmo se devolverá un valor nulo. Como parámetros recibe:

- **digestAlg:** Parámetro que representa la URI del algoritmo de hash.

```
public static MessageImprint generateMessageImprintFromXMLAlgorithm(String
hashAlgXML, byte[ ] data)
```

Método que calcula el "MessageImprint" de un conjunto de datos aplicando un determinado algoritmo de resumen. Como parámetros recibe:

- **hashAlgXML:** Parámetro que representa la URI del algoritmo de resumen.
- **data:** Parámetro que representa el conjunto de datos.

Clase EVisorUtil

Esta clase define utilidades relacionadas con los WS de eVisor. Está compuesta por los métodos:

```
public static Map<String, Object> newBarcodeMap(String message, String type,
Map<String, String> configParams)
```

Método que obtiene un mapa con todos los valores del nodo < *srsm:Barcode*>. Devuelve un mapa con tres elementos, el primero tiene como clave la etiqueta *srsm:Message* y como valor el mensaje del código de barras; el segundo tiene como clave la etiqueta *srsm:Type* y como valor el tipo del código de barras; el tercero tiene como clave la etiqueta con formato XPath *srsm:Configuration/srsm:Parameter* y como valor un mapa con los parámetros de configuración del código de barras. Como parámetros recibe:

- **message:** Parámetro que representa el mensaje del código de barras.
- **type:** Parámetro que representa el tipo del código de barras.
- **configParams:** Parámetros de configuración del código de barras.

```
public static Map<?, ?>[ ] newParameterMap(Map<String, String> configParams)
```

Método que obtiene una colección de mapas a partir de los parámetros de configuración de un código de barras. Devuelve una colección de mapas donde cada mapa se compone de dos registros, uno para la etiqueta *srsm:ParameterId*, y otro para la etiqueta *srsm:ParameterValue*. En caso de que no sea posible procesar los parámetros de entrada se devolverá un valor nulo. Como parámetros recibe:

- **configParams:** Parámetros de configuración del código de barras.

Clase GenericUtils

Esta clase contiene utilidades de uso genérico. Está compuesta por los métodos:

```
public static boolean assertStringValue(String value)
```

Método que comprueba si una cadena de texto no es vacía ni nula. Devuelve un valor lógico que indica si la cadena de texto no es vacía ni nula (verdadero) o por el contrario es vacía o nula (falso). Como parámetros recibe:

- **value:** Parámetro que representa la cadena de texto a comprobar.

```
public static boolean assertArrayValid(byte[ ] data)
```

Método que comprueba si una colección de bytes no es vacía ni nula. Devuelve un valor lógico que indica si la colección de bytes no es vacía ni nula (verdadero) o por el contrario es vacía o nula (falso). Como parámetros recibe:

- **data:** Parámetro que representa la colección de bytes a comprobar.

```
public static String getValueFromMapsTree(String path, Map<String, Object>
treeValues)
```

Método que obtiene un valor concreto de un árbol de mapas a partir de una ruta indicada. Devuelve una cadena de texto que se corresponde con el valor buscado. Como parámetros recibe:

- **path:** Parámetro que representa la ruta en el árbol de mapas, utilizando como separador el carácter '/'.
- **treeValues:** Parámetro que representa el árbol de mapas que procesar.

```
public static byte[ ] getDataFromInputStream(final InputStream input) throws
IOException
```

Método que obtiene una colección de bytes a partir de una secuencia de datos. Devuelve la colección de bytes leídos de la secuencia de datos de entrada. En caso de que se produjese algún error durante el proceso, el método lanzará una excepción *IOException*. Como parámetros recibe:

- **input:** Parámetro que representa la secuencia de datos de entrada que procesar.

```
public static boolean checkNullValues(Object... values)
```

Método que comprueba si ninguno de los valores de un conjunto es nulo. Devuelve un valor lógico que indica si ningún valor es nulo (verdadero) o bien alguno es nulo (falso). Como parámetros recibe:

- **values:** Parámetro que representa el conjunto de valores a procesar.

```
public static void printResult(byte[ ] result, Logger logger)
```

Método que codifica en Base 64 y escribe en el log un conjunto de datos. Como parámetros recibe:

- **result:** Parámetro que representa el conjunto de datos que codificar en Base 64 y escribir en el log.
- **logger:** Parámetro que representa el elemento que gestiona el log.

Clase UtilsCertificate

Esta clase contiene define utilidades asociadas a la gestión de certificados. Está compuesta por los métodos:

```
public static String canonicalizeX500Principal(String x500PrincipalName)
```

Método que canonicaliza un elemento X.500 Principal de un certificado. Devuelve una cadena de texto que se corresponde con el elemento canonicalizado. Como parámetros recibe:

- **x500PrincipalName:** Parámetro que representa un elemento X.500 Principal de un certificado.

```
public static X509Certificate generateCertificate(byte[ ] certificateBytes) throws
CertificateException
```

Método que genera un certificado a partir de una colección de bytes. Devuelve un objeto X.509 que representa el certificado. En caso de que no sea posible obtener el certificado a partir de la colección de bytes, el método lanzará una excepción *CertificateException*. Como parámetros recibe:

- **certificateBytes:** Parámetro que representa el certificado.

```
public static boolean equals(X509Certificate cert1, X509Certificate cert2)
```

Método que compara la clave pública, el emisor y el número de serie de dos certificados. Devuelve un valor lógico que indica si ambos certificados son iguales (verdadero) o no (falso). Como parámetros recibe:

- **cert1:** Parámetro que representa el primer certificado a comparar.
- **cert2:** Parámetro que representa el segundo certificado a comparar.

Clase UtilsFileSystem

Esta clase contiene utilidades de uso genérico. Está compuesta por los métodos:

```
public static synchronized String readFileBase64Encoded(String path, boolean
isRelativePath)
```

Método que lee un archivo y lo codifica en Base 64. Devuelve una cadena de texto codificada en Base 64 que representa el contenido del archivo. Como parámetros recibe:

- **path:** Parámetro que representa la ruta al fichero. Puede ser una ruta completa o relativa.
- **isRelativePath:** Parámetro que indica si la ruta al fichero es relativa (verdadero) o completa (falso).

```
public static synchronized byte[ ] getArrayByteFileBase64Encoded(String path,
boolean isRelativePath)
```

Método que lee un archivo y lo codifica en Base 64. Devuelve una colección de bytes codificada en Base 64 que representa el contenido del archivo. Como parámetros recibe:

- **path:** Parámetro que representa la ruta al fichero. Puede ser una ruta completa o relativa.
- **isRelativePath:** Parámetro que indica si la ruta al fichero es relativa (verdadero) o completa (falso).

```
public static synchronized byte[ ] readFile(String path, boolean isRelativePath)
```

Método que lee un archivo. Devuelve una colección de bytes que representa el contenido del archivo. Como parámetros recibe:

- **path:** Parámetro que representa la ruta al fichero. Puede ser una ruta completa o relativa.
- **isRelativePath:** Parámetro que indica si la ruta al fichero es relativa (verdadero) o completa (falso).

```
public static void writeFile(byte[ ] data, String filename) throws IOException
```

Método que genera un archivo a partir de un conjunto de datos. En caso de producirse algún error durante el proceso, el método lanzará una excepción *IOException*. Como parámetros recibe:

- **data:** Parámetro que representa el conjunto de datos que se corresponden con el archivo.

- **filename:** Parámetro que representa la ruta completa al archivo a generar.

Clase UtilsKeystore

Esta clase contiene utilidades que permiten el manejo de almacenes de claves. Está compuesta por los métodos:

```
public static KeyStore loadKeystore(String path, String password, String type)
throws KeyStoreException, NoSuchAlgorithmException, CertificateException,
IOException
```

Método que lee un almacén de claves. En caso de que el tipo de almacén de claves a instanciar no esté reconocido, el método lanzará una excepción *KeyStoreException*. En caso de que el algoritmo usado para comprobar la integración del almacén de claves no esté reconocido, el método lanzará una excepción *NoSuchAlgorithmException*. En caso de que alguno de los certificados del almacén de claves no pueda ser cargado, el método lanzará una excepción *CertificateException*. En caso de que se produzca algún error de lectura del almacén de claves, el método lanzará una excepción *IOException*. Devuelve un objeto java que representa el almacén de claves. Como parámetros recibe:

- **path:** Parámetro que representa la ruta completa al almacén de claves.

- **password:** Parámetro que representa la contraseña para acceder al almacén de claves.

- **type:** Parámetro que representa el tipo del almacén de claves.

```
public static byte[ ] getCertificateEntry(byte[ ] keystore, String
keystoreDecodedPass, String alias, String keystoreType) throws KeyStoreException,
NoSuchAlgorithmException, CertificateException, IOException
```

Método que obtiene un certificado almacenado en un almacén de claves. Devuelve una colección de bytes que se corresponden con el objeto X.509 del certificado. En caso de que el tipo de almacén de claves a instanciar no esté reconocido, el método lanzará una excepción *KeyStoreException*. En caso de que el algoritmo usado para comprobar la integración del almacén de claves no esté reconocido, el método lanzará una excepción *NoSuchAlgorithmException*. En caso de que alguno de los certificados del almacén de claves no pueda ser cargado, el método lanzará una excepción *CertificateException*. En caso de que se produzca algún error de lectura del almacén de claves, el método lanzará una excepción *IOException*. Como parámetros recibe:

- **keystore:** Parámetro que representa la colección de bytes que se corresponden con el almacén de claves.

- **keystoreDecodedPass:** Parámetro que representa la contraseña para acceder al almacén de claves.

- **alias:** Parámetro que representa el alias del certificado a obtener.

- **keystoreType:** Parámetro que representa el tipo del almacén de claves.

```
public static PrivateKey getPrivateKeyEntry(byte[ ] keystore, String
keystoreDecodedPass, String alias, String keystoreType, String
privateKeyDecodedPass) throws KeyStoreException, NoSuchAlgorithmException,
CertificateException, IOException, UnrecoverableKeyException
```

Método que obtiene una clave privada almacenada en un almacén de claves. Devuelve un objeto java que representa la clave privada. En caso de que el tipo de almacén de claves a instanciar no esté reconocido, el método lanzará una excepción *KeyStoreException*. En caso de que el algoritmo usado para comprobar la integración del almacén de claves no esté reconocido, el método lanzará una excepción *NoSuchAlgorithmException*. En caso de que alguno de los certificados del almacén de claves no pueda ser cargado, el método lanzará una excepción *CertificateException*. En caso de que se produzca algún error de lectura del almacén de claves, el método lanzará una excepción *IOException*. En caso de que la clave privada no pueda ser recuperada, el método lanzará una excepción *UnrecoverableKeyException*. Como parámetros recibe:

- **keystore:** Parámetro que representa la colección de bytes que se corresponden con el almacén de claves.

- **keystoreDecodedPass:** Parámetro que representa la contraseña para acceder al almacén de claves.

- **alias:** Parámetro que representa el alias de la clave privada a obtener.

- **keystoreType:** Parámetro que representa el tipo del almacén de claves.

- **privateKeyDecodedPass:** Parámetro que representa la contraseña para acceder a la clave privada.

```
public static List<X509Certificate> getListCertificates(byte[ ] keystore, String
keystoreDecodedPass, String keystoreType) throws KeyStoreException,
NoSuchAlgorithmException, CertificateException, IOException
```

Método que obtiene una lista con todos los certificados almacenados en un almacén de claves. En caso de que el tipo de almacén de claves a instanciar no esté reconocido, el método lanzará una excepción *KeyStoreException*. En caso de que el algoritmo usado para comprobar la integración del almacén de claves no esté reconocido, el método lanzará una excepción *NoSuchAlgorithmException*. En caso de que alguno de los certificados del almacén de claves no pueda ser cargado, el método lanzará una excepción *CertificateException*. En caso de que se produzca algún error de lectura del almacén de claves, el método lanzará una excepción *IOException*. Como parámetros recibe:

- **keystore:** Parámetro que representa la colección de bytes que se corresponden con el almacén de claves.

- **keystoreDecodedPass:** Parámetro que representa la contraseña para acceder al almacén de claves.

- **keystoreType:** Parámetro que representa el tipo del almacén de claves.

Clase UtilsResources

Esta clase proporciona funcionalidades para controlar el cierre de recursos. Está compuesta por los métodos:

```
public static void safeCloseInputStream(InputStream is)
```

Método que gestiona el cierre de un flujo de entrada. Como parámetros recibe:

- **is:** Parámetro que representa el flujo de entrada.

```
public static void safeCloseOutputStream(OutputStream os)
```

Método que gestiona el cierre de un flujo de salida. Como parámetros recibe:

- **os:** Parámetro que representa el flujo de salida.

```
public static void safeCloseSocket(Socket socket)
```

Método que gestiona el cierre de un *socket*. Como parámetros recibe:

- **socket:** Parámetro que representa el *socket*.

Clase UtilsSignature

Esta clase proporciona funcionalidades criptográficas relacionadas con firmas electrónicas. Está compuesta por los métodos:

```
public static void validateCertificate(X509Certificate certificate, Date
validationDate, boolean isUpgradeOperation) throws SigningException
```

Método que valida el periodo de validez y el estado de revocación de un certificado. Si el certificado no es válido o se ha producido algún error durante la validación, se lanzará una excepción *SigningException.* Como parámetros recibe:

- **certificate:** Parámetro que representa el certificado a validar.

- **validationDate:** Paramétro que indica la fecha de validación.

- **isUpgradeOperation:** Bandera que indica si la operación original es de actualización o no.

```
public static PDFSignatureDictionary obtainLatestSignatureFromPDF(PdfReader
reader)
```

Método que recupera el diccionario de firma con el número de revisión mayor en una firma de tipo PAdES. Los parámetros recibidos son:

- **reader:** Parámetro que representa el objeto Reader del documento PDF.

```
public static boolean isNotPAdESEnhancedPDF(PdfDictionary pdfDic)
```

Método que indica si un diccionario de firma hace referencia a una firma PDF/PAdES-Basic o a una firma PAdES-BES/PAdES-EPES. Los parámetros recibidos son:

- **pdfDic:** Parámetro que representa el diccionario de firma.

```
public static CMSSignedData getCMSSignature(PDFSignatureDictionary
signatureDictionary) throws SigningException
```

Método que recupera el elemento signedData contenido en un diccionario de firma de un documento PDF. Si se produce algún error se lanzará una excepción *SigningException.* Los parámetros recibidos son:

- **signatureDictionary:** Parámetro que representa el diccionario de firma.

```
public static boolean isImplicit(CMSSignedData cmsSignedData)
```

Método que comprueba si la firma incluye el documento original o no.

- **cmsSignedData:** Parámetro que representa el mensaje de tipo pkcs7-signature.

```
public static boolean equalsHash(PdfArray pdfArrayByteRange, MessageDigest
messageDigestSignature, byte[ ] pdfDocument, byte[ ] hashSignature)
```

Método que compara dos bytes de arrays y comprueba si el hash de cada uno de ellos son iguales o no. Como parámetros recibe:

- **pdfArrayByteRange:** Array de bytes que representa los datos originales sobre los que se calculará el hash.

- **messageDigestSignature:** Algoritmo de resumen a utilizar.

- **pdfDocument:** Conjunto de bytes que representa el primer hash a comparar.
- **hashSignature:** Conjunto de bytes que representa el segundo hash a comparar.

```
public static void checkSubFilterConditionsISO320001(PDFSignatureDictionary
dictionarySignature, CMSSignedData signedData)
```

Método que comprueba que se cumpla la condición especificada en la sección 12.8.3.3.1 de la ISO 32000-1:

adbe.pkcs7.detached: The original signed message digest over the document's byte range shall be incorporated as the normal PKCS#7 SignedData field. No data shall be encapsulated in the PKCS#7 SignedData field.

adbe.pkcs7.sha1: The SHA1 digest of the document's byte range shall be encapsulated in the PKCS#7 SignedData field with ContentInfo of type Data. The digest of that SignedData shall be incorporated as the normal PKCS#7 digest.

Como parámetros recibe:

- **dictionarySignature:** Parámetro que representa el diccionario de firma.
- **signedData:** Parámetro que representa los datos firmados.

```
public static void validatePAdESEnhancedMandatoryAttributes(PDFSignatureDictionary
signatureDictionary, CMSSignedData signedData) throws SigningException
```

Método que valida los atributos obligatorios de una firma *PAdES enhanced*. Si se produce algún error durante el proceso el método lanzará una excepción *SigningException*. Como parámetros recibe:

- **signatureDictionary:** Parámetro que representa el diccionario de firma.
- **signedData:** Parámetro que representa los datos firmados.

```
public static void validatePAdESOptionalAttributes(PDFSignatureDictionary
signatureDictionary, CMSSignedData signedData, boolean isEPES, boolean isBasic)
throws SigningException
```

Método que valida los atributos opcionales de una firma PAdES. Si se produce algún error durante el proceso el método lanzará una excepción *SigningException*. Como parámetros recibe:

- **signatureDictionary:** Parámetro que representa el diccionario de firma.
- **signedData:** Parámetro que representa los datos firmados.
- **isEPES:** Bandera que indica si la firma es de tipo PAdES-EPES o PAdES-BES.

- **isBasic:** Bandera que indica si la firma es de tipo PAdES-Basic o PAdES enhanced.

```
public static X509CertificateHolder getX509CertificateHolderBySignerId(Store
certificatesStore, SignerId signerId)
```

Método que obtiene la estructura de un certificado almacenado. Como parámetros recibe:

- **certificatesStore:** Parámetro que representa el certificado almacenado.
- **signerId:** Parámetro que representa el identificador del firmante usado para buscar el certificado.

```
public static void validatePDFSigner(CMSSignedData signedData, SignerInformation
signerInformation, PdfDictionary pdfSignatureDictionary, Date validationDate)
throws SigningException
```

Método que valida la firma contenida en un documento PDF. Si se produce algún error durante el proceso el método lanzará una excepción *SigningException*. Como parámetros recibe:

- **signedData:** Parámetro que representa los datos firmados.
- **signerInformation:** Parámetro que representa la información del firmante.
- **pdfSignatureDictionary:** Parámetro que representa el diccionario de firma PDF.
- **validationDate:** Parámetro que representa la fecha de validación.

```
public static List<XAdESSignerInfo> getXAdESListSigners(Document doc)
```

Método que obtiene una lista con la principal información relativa a los firmantes de una firma XAdES. Como parámetros recibe:

- **doc:** Parámetro que representa el documento XML.

```
public static List<CAdESSignerInfo> getCAdESListSigners(CMSSignedData signedData)
```

Método que obtiene una lista con la principal información relativa a los firmantes de una firma CAdES. Como parámetros recibe:

- **signedData:** Parámetro que representa los datos firmados.

```
public static void
validateXAdESSigner(org.apache.xml.security.signature.XMLSignature xmlSignature,
```

```
X509Certificate signingCertificate, TimeStampToken tst, Element xmlTst, String
signingMode, byte[ ] signedFile, String signedFileName) throws SigningException
```

Método que valida el firmante de una firma XAdES. Si se produce algún error durante el proceso el método lanzará una excepción *SigningException*. Como parámetros recibe:

- **xmlSignature:** Parámetro que representa la firma XML.
- **signingCertificate**: Parámetro que representa el certificado firmante.
- **tst**: Parámetro que representa el sello de tiempo RFC3161 asociado al firmante.
- **xmlTst**: Parámetro que representa el sello de tiempo XML asociado al firmante.
- **signingMode**: Parámetro que representa el modo en el que se ha realizado la firma XAdES (detached, enveloped o enveloping).
- **signedFile**: Parámetro que representa el documento firmado cuando éstos no están incluidos en el XML.
- **signedFileName**: Parámetro que representa el nombre del documento firmado cuando éstos no están incluidos en el XML.

```
public static X509Certificate getSigningCertificate(CMSSignedData signedData,
SignerInformation signerInformation) throws SigningException
```

Método que obtiene el certificado firmante de un firmante incluido dentro de una forma. Si se produce algún error durante el proceso el método lanzará una excepción *SigningException*. Como parámetros recibe:

- **signedData:** Parámetro que representa los datos firmados.
- **signerInformation**: Parámetro que representa la información relativa al firmante sobre el que se quiere obtener el certificado.

```
public static Document getDocumentFromXML(byte[ ] xmlDocument) throws
SigningException
```

Método que obtiene un objeto java como representación de un documento XML. Si se produce algún error durante el proceso el método lanzará una excepción *SigningException*. Como parámetros recibe:

- **xmlDocument:** Parámetro que representa el documento XML.

```
public static PdfReader obtainLatestRevision(PdfReader reader) throws
SigningException
```

Método que obtiene el diccionario de firma con el número de revisión más reciente. Si se produce algún error durante el proceso el método lanzará una excepción SigningException. Como parámetros recibe:

- **reader:** Parámetro que representa el *reader* del documento PDF.

```
public static void checkPDFCertificationLevel(Map<Integer, InputStream>
mapRevisions) throws SigningException
```

Método que comprueba si se ha añadido alguna firma al documento PDF tras haber sido certificado. Si se produce algún error durante el proceso el método lanzará una excepción SigningException. Como parámetros recibe:

- **mapRevisions:** Parámetro que representa el conjunto de revisiones del documento PDF. Cada revisión representa un diccionario de firma.

Clase UtilsTimestamp

Esta clase proporciona funcionalidades criptográficas relacionadas con sellos de tiempo. Está compuesta por los métodos:

```
public static Object getTimestampFromDssService(byte[ ] dataToStamp, String
applicationID, String signatureType) throws SigningException
```

Método que obtiene un sello de tiempo de TS@. El sello de tiempo puede ser de tipo RFC 3161 (objeto java org.bouncycastle.tsp.TimeStampToken) o bien de tipo XML (objeto java org.w3c.dom.Element). Para poder obtener el sello de tiempo de TS@ será necesario que los ficheros **tsaXXXXX.properties e integra properties** esté configurado correctamente respecto a las propiedades asociadas a la comunicación con TS@. En caso de que se produzca un error durante el proceso y no sea posible obtener el sello de tiempo el método lanzará una excepción *SigningException*. Como parámetros recibe:

- **dataToStamp:** Parámetro que representa los datos a sellar.
- **applicationID:** Parámetro que representa el identificador de aplicación cliente para la comunicación con TS@.
- **signatureType:** Parámetro que representa la URI del tipo de sello de tiempo que generar. Los valores permitidos son:
  - **urn:ietf:rfc:3161** → Sello de tiempo RFC 3161.
  - **urn:oasis:names:tc:dss:1.0:core:schema:XMLTimeStampToken** → Sello de tiempo XML.

```
public static void validateXMLTimestamp(Element tst) throws SigningException
```

Método que valida estructuralmente un sello de tiempo XML (no valida el certificado firmante). En caso de que se produzca un error durante el proceso o si el sello de tiempo no es válido, el método lanzará una excepción *SigningException*. Como parámetros recibe:

- **tst:** Parámetro que representa el sello de tiempo XML que validar.

```
public static Date getGenTimeXMLTimestamp(Element xmlTimestamp) throws
SigningException
```

Método que obtiene la fecha de generación de un sello de tiempo XML. En caso de que la fecha de generación del sello de tiempo no tenga un formato UTC el método lanzará una excepción *SigningException*. Como parámetros recibe:

- **xmlTimestamp:** Parámetro que representa el sello de tiempo XML.

```
public static void checkInputDocumentXMLTimeStamp(Element inputDocuments, Element
signature) throws TSAServiceInvokerException
```

Método que comprueba la integridad de un sello de tiempo XML, es decir, comprueba si el documento de entrada asociado al sello de tiempo es correcto. En caso de que se produzca algún error durante el proceso, o bien el documento de entrada no sea el correcto para el sello de tiempo, el método lanzará una excepción *TSAServiceInvokerException*. Como parámetros recibe:

- **inputDocuments:** Parámetro que representa el elemento *dss:InputDocuments* asociado al sello de tiempo XML que comprobar.

- **signature:** Parámetro que representa el elemento *ds:Signature* del sello de tiempo XML.

```
public static void checkInputDocumentRFC3161TimeStamp(Element inputDocuments,
TimeStampToken tst) throws TSAServiceInvokerException
```

Método que comprueba la integridad de un sello de tiempo RFC 3161, es decir, comprueba si el documento de entrada asociado al sello de tiempo es correcto. En caso de que se produzca algún error durante el proceso, o bien el documento de entrada no sea el correcto para el sello de tiempo, el método lanzará una excepción *TSAServiceInvokerException*. Como parámetros recibe:

- **inputDocuments:** Parámetro que representa el elemento *dss:InputDocuments* asociado al sello de tiempo RFC 3161 que comprobar.

- **tst:** Parámetro que representa el sello de tiempo RFC 3161 como un objeto ASN.1.

Clase UtilsXML

Esta clase proporciona funcionalidades relacionadas con el manejo de elementos XML. Está compuesta por los métodos:

```
public static Document parseDocument(Reader input) throws TransformersException
```

Método que obtiene un documento XML a partir de un flujo de entrada. En caso de que se produzca algún error durante el proceso el método lanzará una excepción *TransformersException*. Como parámetros recibe:

- **input:** Parámetro que representa los datos a sellar.

```
public static String getElementValue(Element e)
```

Método que obtiene el valor de un elemento XML. Como parámetros recibe:

- **e:** Parámetro que representa el elemento XML.

```
public static String getElementValue(Element e, String elementName)
```

Método que obtiene el valor de un elemento hijo de otro elemento XML. El elemento hijo debe estar localizado en el primer nivel de la estructura jerárquica del elemento XML padre. Como parámetros recibe:

- **e:** Parámetro que representa el elemento XML padre.
- **elementName:** Parámetro que representa el nombre del elemento hijo que obtener.

```
public static Element getElement(Element element, String elementName)
```

Método que obtiene un elemento hijo de otro elemento XML. El elemento hijo debe estar localizado en el primer nivel de la estructura jerárquica del elemento XML padre. Como parámetros recibe:

- **e:** Parámetro que representa el elemento XML padre.
- **elementName:** Parámetro que representa el nombre del elemento hijo que obtener.

```
public static Element searchChild(Element e, String childName)
```

Método que obtiene un elemento hijo de otro elemento XML padre. Como parámetros recibe:

- **e:** Parámetro que representa el elemento XML padre.

- **childName:** Parámetro que representa el nombre del elemento hijo que obtener.

```
public static List<Object> searchListChilds(Element e, String childElementName)
```

Método que obtiene una lista con los nodos hijos localizados en el primer nivel de la estructura jerárquica de hijos para un elemento XML padre. Como parámetros recibe:

- **e:** Parámetro que representa el elemento XML padre.
- **childElementName:** Parámetro que representa el nombre de los elementos hijo que obtener.

```
public static List<Element> searchChildElements(Element e)
```

Método que obtiene una lista con los elementos hijos de un elemento XML padre. Como parámetros recibe:

- **e:** Parámetro que representa el elemento XML padre.

```
public static Element createChild(Element e, String childName)
```

Método que crea un elemento hijo para un elemento XML padre. Como parámetros recibe:

- **e:** Parámetro que representa el elemento XML padre.
- **childName:** Parámetro que representa el nombre del elemento a crear.

```
public static boolean existsElement(Element e, String elementName)
```

Método que comprueba si existe un elemento XML como hijo de otro elemento XML. Devuelve un valor lógico que indica si el elemento XML padre contiene el elemento indicado (verdadero) o no (falso). Como parámetros recibe:

- **e:** Parámetro que representa el elemento XML padre.
- **elementName:** Parámetro que representa el nombre del elemento hijo a comprobar.

```
public static Element removeElement(Element e, String elementName)
```

Método que elimina un elemento hijo de otro elemento XML. Devuelve el elemento eliminado. Como parámetros recibe:

- **e:** Parámetro que representa el elemento XML padre.

- **elementName:** Parámetro que representa el nombre del elemento hijo a eliminar.

```
public static Element replaceElementValue(Element e, String elementName, String
elementValue)
```

Método que sustituye un elemento con un valor nuevo. Si el elemento no existe previamente, lo crea. Devuelve el elemento XML padre con el nuevo elemento hijo. Como parámetros recibe:

- **e:** Parámetro que representa el elemento XML padre.
- **elementName:** Parámetro que representa el nombre del elemento hijo a sustituir.
- **elementValue:** Parámetro que representa el nuevo valor del elemento hijo a sustituir.

```
public static Element createElementValue(Element e, String elementName, String
elementValue)
```

Método que crea un nuevo elemento hijo respecto a otro elemento XML padre. Devuelve el elemento padre actualizado con el nuevo elemento hijo. Como parámetros recibe:

- **e:** Parámetro que representa el elemento XML padre.
- **elementName:** Parámetro que representa el nombre del elemento hijo a crear.
- **elementValue:** Parámetro que representa el nuevo valor del elemento hijo a crear.

```
public static String toXMLString(Object xmlElement, String rootName, boolean
asAttributes) throws TransformersException
```

Método que obtiene una cadena de texto con los valores de un elemento XML. En caso de que se produzca un error durante el proceso el método lanzará una excepción *TransformersException*. Como parámetros recibe:

- **xmlElement:** Parámetro que representa el elemento XML.
- **rootName:** Parámetro que representa el nombre del elemento raíz.
- **asAttributes:** Parámetro que indica si los valores del elemento XML deben ser definidos como atributos (verdadero) o como elementos (falso).

```
public static String transformDOMtoString(Element xmlElement, boolean
omitXmlDeclaration) throws TransformersException
```

Método que genera una cadena de texto con formato XML a partir de una estructura DOM arbórea. En caso de que se produzca un error durante el proceso el método lanzará una excepción *TransformersException*. Como parámetros recibe:

- **xmlElement:** Parámetro que representa el elemento XML que procesar.
- **omitXmlDeclaration:** Parámetro que especifica si el procesador XSLT omitir la declaración XML (verdadero) o no (falso).

```
public static String transformDOMtoString(Document doc) throws
TransformersException
```

Método que genera una cadena de texto con formato XML a partir de un documento XML. En caso de que se produzca un error durante el proceso el método lanzará una excepción *TransformersException*. Como parámetros recibe:

- **doc:** Parámetro que representa el documento XML que procesar.

```
public static void deleteNodesNotUsed(Element xmlNode, String[ ]
optionalNodeTypes)
```

Método que elimina todos los nodos y etiquetas que no se usan para un elemento XML. El método busca todos los nodos de tipo *afirmaNodeType* y los elimina si el tipo es igual a alguno de los indicados como parámetro. Como parámetros recibe:

- **xmlNode:** Parámetro que representa el elemento XML que procesar.
- **optionalNodeTypes:** Parámetro que representa la colección con los tipos de nodos que eliminar.

```
public static void removeAfirmaAttribute(Element element)
```

Método que elimina todos los nodos de tipo *afirmaNodeType* contenidos dentro de un elemento XML. Como parámetros recibe:

- **element:** Parámetro que representa el elemento XML que procesar.

```
public static Element insertAttributeValue(Element xmlNode, String attributePath,
String value)
```

Método que añade un atributo a un nodo XML. Devuelve el nodo XML actualizado. Como parámetros recibe:

- **xmlNode:** Parámetro que representa el elemento XML que procesar.

- **attributePath:** Parámetro que representa la ruta que compone el nombre del atributo.
- **value:** Parámetro que representa el valor del atributo.

```
public static Element insertValueElement(Element element, String elementName,
String value)
```

Método que añade un nuevo elemento como hijo de otro elemento XML. Devuelve el elemento XML actualizado con el nuevo hijo. Como parámetros recibe:

- **element:** Parámetro que representa el elemento XML padre que procesar.
- **elementName:** Parámetro que representa el nombre del elemento que añadir.
- **value:** Parámetro que representa el valor del elemento que añadir.

```
public static String getAttributeValue(Element element, String elementPath, String
attributeName)
```

Método que obtiene el valor de un atributo perteneciente a un elemento XML. Como parámetros recibe:

- **element:** Parámetro que representa el elemento XML padre que procesar.
- **elementPath:** Parámetro que representa la ruta XPath del elemento XML que encontrar.
- **attributeName:** Parámetro que representa el nombre del atributo que encontrar.

```
public static String getAttributeValue(Element element, String attributeName)
```

Método que obtiene el valor de un atributo perteneciente a un elemento XML. Como parámetros recibe:

- **element:** Parámetro que representa el elemento XML que procesar.
- **attributeName:** Parámetro que representa el nombre del atributo que obtener.

```
public static String getRelativeXPath(Node nodeChild, Node parent)
```

Método que obtiene la ruta relativa a un nodo hijo a partir de su nodo padre. Como parámetros recibe:

- **nodeChild:** Parámetro que representa el nodo XML hijo.
- **parent:** Parámetro que representa el nodo XML padre.

```
public static String getNodeXPath(Node node)
```

Método que obtiene la ruta absoluta de un nodo. Como parámetros recibe:

- **node:** Parámetro que representa el nodo.

```
public static Element getFirstElementNode(Element element)
```

Método que obtiene el primer elemento XML hijo de otro elemento XML. Como parámetros recibe:

- **element:** Parámetro que representa el elemento XML que procesar.

```
public static Document getDocument(InputStream is) throws TransformerException
```

Método que obtiene un objeto que representa un documento XML a partir de un flujo de entrada. En caso de que se produzca algún error durante el proceso el método lanzará una excepción *TransformerException*. Como parámetros recibe:

- **is:** Parámetro que representa el flujo de entrada que procesar.

```
public static Document newDocument() throws ParserConfigurationException
```

Método que obtiene un nuevo documento XML. En caso de que se produzca algún error durante el proceso el método lanzará una excepción *TransformerException*.

```
public static Document getDocumentWithXsdValidation(File xsdSchema, InputStream
xml) throws TransformersException
```

Método que obtiene un documento XML a partir de un flujo de entrada y lo valida contra un esquema XSD. En caso de que se produzca un error durante el proceso o bien si el documento XML no es válido, el método lanzará una excepción *TransformersException*. Como parámetros recibe:

- **xsdSchema:** Parámetro que representa la definición del esquema XSD.
- **xml:** Parámetro que representa el flujo de entrada.

```
public static Document parseXMLDocument(String xml)
```

Método que obtiene un documento XML a partir de una cadena de texto. Como parámetros recibe:

- **xml:** Parámetro que representa el documento XML como cadena de texto.

9.2.7 Paquete es.gob.afirma.hsm

Este paquete contiene clases que gestionan el manejo de dispositivos HSM. Para poder hacer uso de los métodos definidos en este paquete debe estar configurado correctamente el archivo **hsm.properties**. De este paquete se describirán aquellas clases más destacadas.

Clase HSMKeystore

Esta clase maneja todas las operaciones relacionadas con almacenes de claves HSM. Está compuesta por los métodos:

```
public static PrivateKey getPrivateKey(String alias) throws HSMException
```

Método que obtiene una clave privada almacenada en un HSM a partir de su alias. En caso de que se produzca algún error durante el proceso, el método lanzará una excepción *HSMException*. Como parámetros recibe:

- **alias:** Parámetro que representa el alias de la clave privada a obtener.

```
public static X509Certificate getCertificate(String alias) throws HSMException
```

Método que obtiene un certificado almacenado en un HSM a partir de su alias. En caso de que se produzca algún error durante el proceso, el método lanzará una excepción *HSMException*. Como parámetros recibe:

- **alias:** Parámetro que representa el alias del certificado a obtener.

```
public static PrivateKeyEntry getPrivateKeyEntry(String alias) throws HSMException
```

Método que obtiene una entrada asociada a una clave privada almacenado en un HSM a partir de su alias. En caso de que se produzca algún error durante el proceso, el método lanzará una excepción *HSMException*. Como parámetros recibe:

- **alias:** Parámetro que representa el alias de la entrada a obtener.

9.3 Tipo de integración 3. Acceso a servicios WS, DSS, OCSP y RFC 3161

Para este tipo de integración se dispone del conjunto de paquetes disponibles para los dos tipos de integración anteriores.

9.4 Tipo de integración 4. Procesado de firmas CAdES (Baseline o no) y PAdES (Baseline o no)

Para este tipo de integración se dispone del conjunto de paquetes disponibles para el tipo de integración 1 "Acceso a servicios OCSP y RFC 3161" mas los expuestos a continuación.

9.4.1 Paquete es.gob.afirma.signature

Este paquete contiene todas las clases asociadas a funcionalidades de generación, validación y actualización de firmas. En función del paquete específico contenido dentro de este paquete se encontrarán los métodos para:

- Firmas CAdES (Baseline o no)
- Firmas XAdES (Baseline o no. Sólo disponible para el tipo de integración 5 "Procesado de firmas XAdES (Baseline o no) y ASiC-S Baseline, además de CAdES (Baseline o no) y PAdES (Baseline o no)")
- Firmas PAdES (Baseline o no)
- Firmas ASiC-S Baseline (Sólo disponible para el tipo de integración 5 "Procesado de firmas XAdES (Baseline o no) y ASiC-S Baseline, además de CAdES (Baseline o no) y PAdES (Baseline o no)")
- Políticas de Firma

Interfaz Signer

Esta interfaz define las funcionalidades comunes para la firma, co-firma, contra-firma, actualización para las firmas CAdES (Baseline o no), XAdES (Baseline o no), PAdES (Baseline o no), y ASiC-S Baseline y obtención de los datos originalmente firmados de firmas CAdES (Baseline o no), PAdES (Baseline o no), y ASiC-S Baseline que se pueden realizar desde Integr@. Está compuesta por los métodos:

```
public byte[ ] sign(byte[ ] data, String algorithm, String signatureFormat,
PrivateKeyEntry privateKey, Properties extraParams, boolean includeTimestamp,
String signatureForm, String signaturePolicyID) throws SigningException
```

Método que permite la generación de firmas. En caso de que se produzca algún error durante el proceso y no sea posible generar la firma el método lanzará una excepción *SigningException*. Será necesario tener configurado el fichero **integra.properties** para determinar el nivel de validación del certificado firmante. Como parámetros recibe:

- **data:** Parámetro que representa los datos a firmar.

- **algorithm:** Parámetro que representa el algoritmo de firma. En caso de pasar el fichero a firmar como parámetro, los valores permitidos son:

  - **"SHA1withRSA"**
  - **"SHA256withRSA"**
  - **"SHA384withRSA"**
  - **"SHA512withRSA"**

Sin embargo, si lo que se pasa como parámetro es el hash del fichero a firmar, los valores permitidos son:

  - **"SHA-1"**
  - **"SHA-256"**
  - **"SHA-384"**
  - **"SHA-512"**

- **signatureFormat:** Parámetro que representa el modo de firma. Los valores permitidos son:

  - **"explicit mode"** → La firma a generar será explícita (no incluirá los datos firmados). Este valor es sólo aplicable para la generación de firmas CAdES (Baseline o no).
  - **"explicit hash mode"** → La firma a generar será explícita (no incluirá los datos firmados) y el dato de entrada será el resumen hash del fichero, y no el propio fichero.
  - **"implicit mode"** → La firma a generar será implícita (incluirá los datos firmados). Este valor es sólo aplicable para la generación de firmas CAdES (Baseline o no) y PAdES (Baseline o no).
  - **"XAdES Detached"** → La firma a generar incluirá una referencia a los datos firmados. Este valor es sólo aplicable a la generación de firmas XAdES (Baseline o no).
  - **"XAdES Enveloped"** → La firma a generar incluirá los datos firmados. Este valor es sólo aplicable a la generación de firmas XAdES (Baseline o no).
  - **"XAdES Enveloping"** → La firma a generar posee una estructura XML, y ésta contiene internamente el contenido firmado (en un nodo propio). Este valor es sólo aplicable a la generación de firmas XAdES (Baseline o no).

- **"XAdES Externally Detached"** → La firma a generar incluirá referencias externas. Este valor es sólo aplicable a la generación de firmas XAdES (Baseline o no).

- **privateKey:** Parámetro que representa la clave privada con la que realizar la firma.

- **extraParams:** Conjunto de parámetros adicionales y opcionales. Está definido como un mapa donde los valores permitidos como claves para elementos que incluir son:

  - **cades.policyQualifier** → Clave para asociar el valor del atributo *SigPolicyQualifier* cuando la firma a generar es CAdES (Baseline o no), o PAdES (Baseline o no), con política de firma.

  - **xades.claimedRole** → Clave para asociar el valor del elemento *ClaimedRole* cuando la firma a generar es XAdES (Baseline o no).

  - **xades.policyQualifier** → Clave para asociar el valor del elemento *SigPolicyQualifier* cuando la firma a generar es XAdES-EPES o XAdES B-Level con política de firma.

  - **xades.dataFormatObjectDescription** → Clave para asociar el valor con la descripción del documento original cuando la firma a generar es XAdES-EPES o XAdES B-Level con política de firma.

  - **xades.dataFormatObjectMime** → Clave para asociar el valor con el tipo de datos del documento original cuando la firma a generar es XAdES-EPES o XAdES B-Level con política de firma.

  - **xades.dataFormatObjectEncoding** → Clave para asociar el valor con la codificación del documento original cuando la firma a generar es XAdES-EPES o XAdES B-Level con política de firma.

  - **xades.canonicalizationMethod** → Clave para asociar la URI del algoritmo de canonicalización que aplicar para todas las referencias contenidas dentro del elemento *ds:SignedInfo* de la firma XML cuando la firma a generar es XAdES (Baseline o no), o bien ASiC-S conteniendo una firma XAdES Baseline. La misma URI se usará como transformación de la referencia a los datos cuando estos sean XML. Los valores admitidos serán:

    - "**http://www.w3.org/2006/12/xml-c14n11**"
    - "**http://www.w3.org/2001/10/xml-exc-c14n#"**
    - "**http://www.w3.org/TR/2001/REC-xml-c14n-20010315**"
    - "**http://www.w3.org/2006/12/xmlc14n11#WithComments**" (valor sólo permitido para firmas XAdES no Baseline)

- "**http://www.w3.org/2001/10/xml-exc-c14n#WithComments**" (valor sólo permitido para firmas XAdES no Baseline)

- "**http://www.w3.org/TR/2001/REC-xml-c14n-20010315#WithComments**" (valor sólo permitido para firmas XAdES no Baseline)

En caso de que dicho valor no se indique en el mapa de parámetros adicionales se utilizará como valor por defecto aquél compatible con la especificación XML de Documento Electrónico según ENI, esto es, "**http://www.w3.org/2001/10/xml-exc-c14n#**".

- **Pades.signReason** → Clave para asociar la razón de la firma cuando la firma a generar es PAdES (Baseline o no).

- **Pades.signContact** → Clave para asociar los datos de contacto del firmante cuando la firma a generar es PAdES (Baseline o no).

- **Pades.signLocation** → Clave para asociar la localización donde se produce la firma cuando la firma a generar es PAdES (Baseline o no).

- **Pades.certificationLevel** → Clave para asociar el nivel de certificación a la firma cuando la firma a generar es PAdES (Baseline o no). En el caso de que se quiera indicar que la firma es *Certified* (el documento PDF no admitirá firmas posteriores) el valor será **CERTIFIED_NO_CHANGES_ALLOWED**. En caso de que se quiera indicar que la firma es *Approval* (el documento PDF admitirá firmas posteriores) el valor será **NOT_CERTIFIED**. Si no se incluye esta clave, por defecto la firma a generar será Approval.

- **Pades.image** → Clave para asociar una imagen que se quiere insertar en el documento PDF como rúbrica. Los formatos permitidos son JPEG, PNG, GIF y BMP, codificado en Base64.

- **Pades.imagePage** → Clave para indicar el número de la página en la que se quiere insertar la rúbrica. La numeración de las páginas comienza en uno. Si se indica -1 como número de página, la imagen se insertará en la útlima página del documento, sino, se insertará en la página indicada.

- **Pades.imagePositionOnPageLowerLeftX** → Clave para indicar la coordenada horizontal inferior izquierda de la posición de la imagen dentro de la página.

- **Pades.imagePositionOnPageLowerLeftY** → Clave para indicar la coordenada vertical inferior izquierda de la posición de la imagen dentro de la página.

- **Pades.imagePositionOnPageUpperRightX** → Clave para indicar la coordenada horizontal superior derecha de la posición de la imagen dentro de la página.

- **Pades.imagePositionOnPageUpperRightY** → Clave para indicar la coordenada vertical superior derecha de la posición de la imagen dentro de la página.

- **includeTimestamp:** Parámetro que indica si la firma a generar debe tener sello de tiempo (verdadero) o no (falso). En caso de que se indique que la firma debe tener sello de tiempo será necesario tener configurados correctamente los ficheros **integra.properties** y **tsaXXXXX.properties** para permitir la comunicación con TS@ y así obtener el sello de tiempo, y además, se validará el núcleo de dicho sello de tiempo obtenido de TS@.

- **signatureForm:** Parámetro que representa el formato de la firma a generar. Los valores permitidos son:

  - **CAdES-BES** → Formato definido para CAdES-BES.

  - **CAdES-EPES** → Formato definido para CAdES-EPES. En este caso es necesario tener correctamente configurado el fichero **integra.properties** con todas las propiedades asociadas a las políticas de firma admitidas.

  - **XAdES-BES** → Formato definido para XAdES-BES.

  - **XAdES-EPES** → Formato definido para XAdES-EPES. En este caso es necesario tener correctamente configurado el fichero **integra.properties** con todas las propiedades asociadas a las políticas de firma admitidas.

  - **PAdES-Basic** → Formato definido para PAdES-Basic.

  - **PAdES-BES** → Formato definido para PAdES-BES.

  - **PAdES-EPES** → Formato definido para PAdES-EPES. En este caso es necesario tener correctamente configurado el fichero **integra.properties** con todas las propiedades asociadas a las políticas de firma admitidas.

  - **CAdES B-Level** → Formato definido para CAdES B-Level. Si se indica este formato desde la implementación de la interfaz Signer asociada a firmas ASiC-S Baseline se generará una firma ASiC-S Baseline que contendrá una firma CAdES B-Level.

  - **XAdES B-Level** → Formato definido para XAdES B-Level. Si se indica este formato desde la implementación de la interfaz Signer asociada a firmas ASiC-S Baseline se generará una firma ASiC-S Baseline que contendrá una firma XAdES B-Level.

  - **PAdES B-Level** → Formato definido para PAdES B-Level.

- **signaturePolicyID:** Parámetro que representa el identificador de política de firma a usar en la generación de la firma. Será obligatorio si se ha indicado como formato de firma a generar **CAdES-EPES**, **XAdES-EPES** o **PAdES-EPES**. Este identificador debe coincidir con alguno

de los definidos en el fichero **integra.properties**, por lo que este fichero deberá estar correctamente configurado.

```
public byte[ ] coSign(byte[ ] signature, byte[ ] document, String algorithm,
PrivateKeyEntry privateKey, Properties extraParams, boolean includeTimestamp,
String signatureForm, String signaturePolicyID) throws SigningException
```

Método que permite la generación de co-firmas. Este método es sólo aplicable a firmas CAdES (Baseline o no), XAdES (Baseline o no) y PAdES (Baseline o no). En caso de que se produzca algún error durante el proceso y no sea posible generar la co-firma el método lanzará una excepción *SigningException*. Será necesario tener configurado el fichero **integra.properties** para determinar el nivel de validación del certificado firmante. Como parámetros recibe:

- **signature:** Parámetro que representa la firma a co-firmar.
- **document:** Parámetro que representa los datos usados para generar la firma a co-firmar.
- **algorithm:** Parámetro que representa el algoritmo de firma. Los valores permitidos son:
  - **SHA1withRSA**
  - **SHA256withRSA**
  - **SHA384withRSA**
  - **SHA512withRSA**
- **privateKey:** Parámetro que representa la clave privada con la que realizar la co-firma.
- **extraParams:** Conjunto de parámetros adicionales y opcionales. Está definido como un mapa donde los valores permitidos como claves para elementos que incluir son:
  - **cades.policyQualifier** → Clave para asociar el valor del atributo SigPolicyQualifier cuando la co-firma a generar es CAdES (Baseline o no), con política de firma.
  - **xades.claimedRole** → Clave para asociar el valor del elemento *ClaimedRole* cuando la co-firma a generar es XAdES (Baseline o no), con política de firma.
  - **xades.policyQualifier** → Clave para asociar el valor del elemento *SigPolicyQualifier* cuando la co-firma a generar es XAdES (Baseline o no), con política de firma.
  - **xades.dataFormatObjectDescription** → Clave para asociar el valor con la descripción del documento original cuando la co-firma a generar es XAdES (Baseline o no), con política de firma.

- **xades.dataFormatObjectMime** → Clave para asociar el valor con el tipo de datos del documento original cuando la co-firma a generar es XAdES (Baseline o no), con política de firma.

- **xades.dataFormatObjectEncoding** → Clave para asociar el valor con la codificación del documento original cuando la co-firma a generar es XAdES (Baseline o no), con política de firma.

- **xades.canonicalizationMethod** → Clave para asociar la URI del algoritmo de canonicalización que aplicar para todas las referencias contenidas dentro del elemento *ds:SignedInfo* de la firma XML cuando la co-firma a generar es XAdES (Baseline o no). La misma URI se usará como transformación de la referencia a los datos cuando estos sean XML. Los valores admitidos serán:

  - "**http://www.w3.org/2006/12/xml-c14n11**"
  - "**http://www.w3.org/2001/10/xml-exc-c14n#"**
  - "**http://www.w3.org/TR/2001/REC-xml-c14n-20010315**"
  - "**http://www.w3.org/2006/12/xmlc14n11#WithComments**" (valor sólo permitido para firmas XAdES no Baseline)
  - "**http://www.w3.org/2001/10/xml-exc-c14n#WithComments**" (valor sólo permitido para firmas XAdES no Baseline)
  - "**http://www.w3.org/TR/2001/REC-xml-c14n-20010315#WithComments**" (valor sólo permitido para firmas XAdES no Baseline)

  En caso de que dicho valor no se indique en el mapa de parámetros adicionales se utilizará como valor por defecto aquél compatible con la especificación XML de Documento Electrónico según ENI, esto es, "**http://www.w3.org/2001/10/xml-exc-c14n#**".

- **includeTimestamp:** Parámetro que indica si la co-firma a generar debe tener sello de tiempo (verdadero) o no (falso). En caso de que se indique que la co-firma debe tener sello de tiempo será necesario tener configurados correctamente los ficheros **integra.properties** y **tsaXXXXX.properties** para permitir la comunicación con TS@ y así obtener el sello de tiempo, y además, se validará el núcleo de dicho sello de tiempo obtenido de TS@.

- **signatureForm:** Parámetro que representa el formato de la co-firma a generar. Los valores permitidos son:

  - **CAdES-BES** → Formato definido para CAdES-BES.

- **CAdES-EPES** → Formato definido para CAdES-EPES. En este caso es necesario tener correctamente configurado el fichero **integra.properties** con todas las propiedades asociadas a las políticas de firma admitidas.
- **PAdES-BES** → Formato definido para PAdES-BES.
- **PAdES-EPES** → Formato definido para PAdES-EPES. En este caso es necesario tener correctamente configurado el fichero **integra.properties** con todas las propiedades asociadas a las políticas de firma admitidas.
- **XAdES-BES** → Formato definido para XAdES-BES.
- **XAdES-EPES** → Formato definido para XAdES-EPES. En este caso es necesario tener correctamente configurado el fichero **integra.properties** con todas las propiedades asociadas a las políticas de firma admitidas.
- **CAdES B-Level** → Formato definido para CAdES B-Level.
- **XAdES B-Level** → Formato definido para XAdES B-Level.
- **PAdES B-Level** → Formato definido para PAdES B-Level.

- **signaturePolicyID:** Parámetro que representa el identificador de política de firma a usar en la generación de la co-firma. Será obligatorio si se ha indicado como formato de firma a generar **CAdES-EPES**, **XAdES-EPES** o **PAdES-EPES**. Este identificador debe coincidir con alguno de los definidos en el fichero **integra.properties**, por lo que este fichero deberá estar correctamente configurado.

```
[1]public byte[ ] counterSign(byte[ ] signature, String algorithm, PrivateKeyEntry
privateKey, Properties extraParams, boolean includeTimestamp, String
signatureForm, String signaturePolicyID) throws SigningException
```

Método que permite la generación de contra-firmas. Este método es sólo aplicable a firmas CAdES (Baseline o no), XAdES (Baseline o no) y PAdES (Baseline o no). En caso de que se produzca algún error durante el proceso y no sea posible generar la contra-firma el método lanzará una excepción *SigningException*. Será necesario tener configurado el fichero **integra.properties** para determinar el nivel de validación del certificado firmante. Como parámetros recibe:

- **signature:** Parámetro que representa la firma a contra-firmar.
- **algorithm:** Parámetro que representa el algoritmo de firma. Los valores permitidos son:

[1] En el caso de XAdES (Baseline o no), según su estándar, no se permite la generación de contra-firmas que no sean Detached. Es por ello que, en el caso de que se pretenda contra-firmar una firma no Detached (Enveloped o Enveloping), dicha firma será reconfigurada previamente a tipo Detached para, a continuación, llevar a cabo la contra-firma.

- **SHA1withRSA**
- **SHA256withRSA**
- **SHA384withRSA**
- **SHA512withRSA**

- **privateKey:** Parámetro que representa la clave privada con la que realizar la contra-firma.
- **extraParams:** Conjunto de parámetros adicionales y opcionales. Está definido como un mapa donde los valores permitidos como claves para elementos que incluir son:
  - **cades.policyQualifier** → Clave para asociar el valor del atributo SigPolicyQualifier cuando la contra-firma a generar es CAdES (Baseline o no), con política de firma.
  - **xades.claimedRole** → Clave para asociar el valor del elemento *ClaimedRole* cuando la contra-firma a generar es XAdES (Baseline o no), con política de firma.
  - **xades.policyQualifier** → Clave para asociar el valor del elemento *SigPolicyQualifier* cuando la contra-firma a generar es XAdES (Baseline o no), con política de firma.
  - **xades.dataFormatObjectDescription** → Clave para asociar el valor con la descripción del documento original cuando la contra-firma a generar es XAdES (Baseline o no), con política de firma.
  - **xades.dataFormatObjectMime** → Clave para asociar el valor con el tipo de datos del documento original cuando la contra-firma a generar es XAdES (Baseline o no), con política de firma.
  - **xades.dataFormatObjectEncoding** → Clave para asociar el valor con la codificación del documento original cuando la contra-firma a generar es XAdES (Baseline o no), con política de firma.
  - **xades.canonicalizationMethod** → Clave para asociar la URI del algoritmo de canonicalización que aplicar para todas las referencias contenidas dentro del elemento *ds:SignedInfo* de la firma XML cuando la contra-firma a generar es XAdES (Baseline o no), o bien ASiC-S conteniendo una firma XAdES Baseline. La misma URI se usará como transformación de la referencia a los datos cuando estos sean XML. Los valores admitidos serán:
    - "**http://www.w3.org/2006/12/xml-c14n11**"
    - "**http://www.w3.org/2001/10/xml-exc-c14n#**"

- "**http://www.w3.org/TR/2001/REC-xml-c14n-20010315**"
- "**http://www.w3.org/2006/12/xmlc14n11#WithComments**" (valor sólo permitido para firmas XAdES no Baseline)
- "**http://www.w3.org/2001/10/xml-exc-c14n#WithComments**" (valor sólo permitido para firmas XAdES no Baseline)
- "**http://www.w3.org/TR/2001/REC-xml-c14n-20010315#WithComments**" (valor sólo permitido para firmas XAdES no Baseline)

En caso de que dicho valor no se indique en el mapa de parámetros adicionales se utilizará como valor por defecto aquél compatible con la especificación XML de Documento Electrónico según ENI, esto es, "**http://www.w3.org/2001/10/xml-exc-c14n#**".

- **includeTimestamp:** Parámetro que indica si la contra-firma a generar debe tener sello de tiempo (verdadero) o no (falso). En caso de que se indique que la contra-firma debe tener sello de tiempo será necesario tener configurados correctamente los ficheros **integra.properties** y **tsaXXXXX.properties** para permitir la comunicación con TS@ y así obtener el sello de tiempo, y además, se validará el núcleo de dicho sello de tiempo obtenido de TS@.

- **signatureForm:** Parámetro que representa el formato de la contra-firma a generar. Los valores permitidos son:

  - **CAdES-BES** → Formato definido para CAdES-BES.
  - **CAdES-EPES** → Formato definido para CAdES-EPES. En este caso es necesario tener correctamente configurado el fichero **integra.properties** con todas las propiedades asociadas a las políticas de firma admitidas.
  - **XAdES-BES** → Formato definido para XAdES-BES.
  - **XAdES-EPES** → Formato definido para XAdES-EPES. En este caso es necesario tener correctamente configurado el fichero **integra.properties** con todas las propiedades asociadas a las políticas de firma admitidas.
  - **PAdES-BES** → Formato definido para PAdES-BES.
  - **PAdES-EPES** → Formato definido para PAdES-EPES. En este caso es necesario tener correctamente configurado el fichero **integra.properties** con todas las propiedades asociadas a las políticas de firma admitidas.
  - **CAdES B-Level** → Formato definido para CAdES B-Level.
  - **XAdES B-Level** → Formato definido para XAdES B-Level.

- **PAdES B-Level** → Formato definido para PAdES B-Level.

- **signaturePolicyID:** Parámetro que representa el identificador de política de firma a usar en la generación de la contra-firma. Será obligatorio si se ha indicado como formato de firma a generar **CAdES-EPES**, **XAdES-EPES** o **PAdES-EPES**. Este identificador debe coincidir con alguno de los definidos en el fichero **integra.properties**, por lo que este fichero deberá estar correctamente configurado.

```
public byte[ ] upgrade(byte[ ] signature, List<X509Certificate> listCertificates)
throws SigningException
```

Método que, en el caso de firmas ASN.1 y XML, actualiza la firma añadiendo un sello de tiempo a todos los firmantes indicados que no posean un sello de tiempo previo. En el caso de que la firma a actualizar sea de tipo PDF el proceso de actualización consistirá en añadir un diccionario de sello de tiempo, independientemente de que ya tuviera uno previamente. Si la firma a actualizar es ASiC-S, el método actualizará su firma ASN.1 o XML contenida añadiendo un sello de tiempo a todos los firmantes indicados que no posean un sello de tiempo previo. Será necesario tener configurados correctamente los ficheros integra.properties, para determinar el nivel de validación de los certificados firmantes, y tsaXXXXX.properties para permitir la comunicación con TS@ y así obtener el sello de tiempo, así como indicar el nivel de validación para los certificados firmantes. El fichero integra.properties deberá estar configurado con los datos de acceso ocsp en el caso de que se haya indicado que se debe validar el estar de revocación del certificado firmante. En caso de que se produzca algún error durante el proceso y no sea posible llevar a cabo la actualización el método lanzará una excepción SigningException.

Para las firmas ASN.1, y XML, de aquellos firmantes que se hayan indicado actualizar y que no tengan un sello de tiempo previo, se validará el certificado firmante respecto a la fecha actual en base al valor indicado en la propiedad **CERTIFICATE_VALIDATION_LEVEL** del archivo **integra.properties**, salvo si se ha indicado que no se haga validación, en cuyo caso se llevará a cabo la validación simple. Además, se validará el núcleo del sello de tiempo que añadir a cada firmante. Para firmas PDF, si no contiene ningún diccionario de sello de tiempo previo, se validarán todos los certificados firmantes que contenga respecto a la fecha actual (si no posee sello de tiempo asociado), o respecto a la fecha del sello de tiempo asociado (en caso de tenerlo), en base al valor indicado en la propiedad **CERTIFICATE_VALIDATION_LEVEL** del archivo **integra.properties**, salvo si se ha indicado que no se haga validación, en cuyo caso se llevará a cabo la validación simple, y se validará el núcleo del sello de tiempo que se añadirá en el diccionario de sello de tiempo. Si la firma PDF contiene algún diccionario de sello de tiempo previo, se validarán todos los certificados firmantes que contenga respecto a la fecha del sello de tiempo más reciente contenido en los diccionarios de sello de tiempo, en base al valor indicado en la propiedad **CERTIFICATE_VALIDATION_LEVEL** del archivo **integra.properties**, salvo si se ha indicado que no se haga validación, en cuyo caso se llevará a cabo la validación simple, y se validará el núcleo del sello de tiempo que se añadirá en el diccionario de sello de tiempo.

Como parámetros recibe:

- **signature:** Parámetro que representa la firma a actualizar.

- **listCertificates:** Parámetro que representa la lista de firmantes a los que actualizar añadiendo un sello de tiempo. Si la lista de firmantes a actualizar es nula o vacía se añadirá el sello de tiempo a todos los firmantes de la firma. Este parámetro es sólo aplicable a firmas ASN.1 o XML.

```
public OriginalSignedData getSignedData(byte[] signature) throws SigningException
```

Método que permite la extracción de los datos originalmente firmados. En caso de que se produzca algún error durante el proceso y no sea posible obtener los datos, se lanzará una excepción SigningException. Como parámetro recibe:

- **signature:** Parámetro que representa la firma de la que se quiere extraer la información.

El método devuelve un objeto OriginalSignedData, que contendrá información acerca de los datos originalmente firmados, diferenciando dicha información en función del tipo de la firma procesada. Si la firma es de tipo PAdES (Baseline o no), el método devolverá la revisión que se corresponde con el documento PDF sin firmar. Si la firma es de tipo CAdES (Baseline o no), si la firma es implícita, devolverá los datos firmados y si es explícita, devolverá el resumen de los datos firmados y el algoritmo de resumen usado para calcularlo. Si la firma es ASiC-S, devolverá el fichero que se encuentra dentro del fichero ZIP y que se corresponde con los datos firmados. Si la firma es XAdES (Baseline o no) el método lanza directamente la excepción SigningException ya que no está soportado para las firmas XAdES.

Clase SignatureFormatDetector

Esta clase define métodos que permiten el reconocimiento de formatos de firma. Está compuesta por los métodos:

```
public static String getSignatureFormat(byte[ ] signature)
```

Método que obtiene el formato de una firma. Devuelve una cadena de texto que identifica el formato de la firma. Los valores que puede devolver son:

- **UNRECOGNIZED** → El formato no está reconocido.
- **CAdES-A** → El formato de la firma es CAdES-A.
- **CAdES-XL1** → El formato de la firma es CAdES-XL1.
- **CAdES-XL2** → El formato de la firma es CAdES-XL2.
- **CAdES-X1** → El formato de la firma es CAdES-X1.
- **CAdES-X2** → El formato de la firma es CAdES-X2.

- **CAdES-C** → El formato de la firma es CAdES-C.
- **CAdES-T** → El formato de la firma es CAdES-T.
- **CAdES-EPES** → El formato de la firma es CAdES-EPES.
- **CAdES-BES** → El formato de la firma es CAdES-BES.
- **CAdES LTA-Level** → El formato de la firma es CAdES LTA-Level.
- **CAdES LT-Level** → El formato de la firma es CAdES LT-Level.
- **CAdES T-Level** → El formato de la firma es CAdES T-Level.
- **CAdES B-Level** → El formato de la firma es CAdES B-Level.
- **XAdES-A** → El formato de la firma es XAdES-A.
- **XAdES-XL1** → El formato de la firma es XAdES-XL1.
- **XAdES-XL2** → El formato de la firma es XAdES-XL2.
- **XAdES-X1** → El formato de la firma es XAdES-X1.
- **XAdES-X2** → El formato de la firma es XAdES-X2.
- **XAdES-C** → El formato de la firma es XAdES-C.
- **XAdES-T** → El formato de la firma es XAdES-T.
- **XAdES-EPES** → El formato de la firma es XAdES-EPES.
- **XAdES-BES** → El formato de la firma es XAdES-BES.
- **XAdES LTA-Level** → El formato de la firma es XAdES LTA-Level.
- **XAdES LT-Level** → El formato de la firma es XAdES LT-Level.
- **XAdES T-Level** → El formato de la firma es XAdES T-Level.
- **XAdES B-Level** → El formato de la firma es XAdES B-Level.
- **PDF** → El formato de la firma es PDF.
- **PAdES-Basic** → El formato de la firma es PAdES-Basic.
- **PAdES-BES** → El formato de la firma es PAdES-BES.
- **PAdES-EPES** → El formato de la firma es PAdES-EPES.

- **PAdES-LTV** → El formato de la firma es PAdES-LTV.
- **PAdES LTA-Level** → El formato de la firma es PAdES LTA-Level.
- **PAdES LT-Level** → El formato de la firma es PAdES LT-Level.
- **PAdES T-Level** → El formato de la firma es PAdES T-Level.
- **PAdES B-Level** → El formato de la firma es PAdES B-Level.
- **ASiC-S LTA-Level** → El formato de la firma es ASiC-S conteniendo una firma CAdES LTa-Level o un documento XML firmado cuyo formato de firma más avanzado es XAdES LTA-Level.
- **ASiC-S LT-Level** → El formato de la firma es ASiC-S conteniendo una firma CAdES LTa-Level o un documento XML firmado cuyo formato de firma más avanzado es XAdES LTA-Level.
- **ASiC-S T-Level** → El formato de la firma es ASiC-S conteniendo una firma CAdES LTa-Level o un documento XML firmado cuyo formato de firma más avanzado es XAdES LTA-Level.
- **ASiC-S B-Level** → El formato de la firma es ASiC-S conteniendo una firma CAdES LTa-Level o un documento XML firmado cuyo formato de firma más avanzado es XAdES LTA-Level.

Como parámetros recibe:

- **signature:** Parámetro que representa la firma a la que determinar el formato.

```
public static boolean isASN1Format(byte[ ] signature)
```

Método que indica si una firma es ASN.1. Devuelve un valor lógico que indica si la firma es ASN.1 (verdadero) o no (falso). Como parámetros recibe:

- **signature:** Parámetro que representa la firma que comprobar.

```
public static boolean isXMLFormat(byte[ ] signature)
```

Método que indica si una firma es XML. Devuelve un valor lógico que indica si la firma es XML (verdadero) o no (falso). Como parámetros recibe:

- **signature:** Parámetro que representa la firma que comprobar.

```
public static boolean isPDFFormat(byte[ ] signature)
```

Método que indica si una firma es de tipo PDF. Devuelve un valor lógico que indica si la firma es de tipo PDF (verdadero) o no (falso). Como parámetros recibe:

- **signature:** Parámetro que representa la firma que comprobar.

```
public static boolean isASiCFormat(byte[ ] signature)
```

Método que indica si una firma es de tipo ASiC-S. Devuelve un valor lógico que indica si la firma es de tipo ASiC-S (verdadero) o no (falso). Como parámetros recibe:

- **signature:** Parámetro que representa la firma que comprobar.

```
public static boolean isPAdESEPES(PDFSignatureDictionary signatureDictionary)
```

Método que indica si la firma contenida en un diccionario de firma de un documento PDF es PAdES-EPES. Devuelve un valor lógico que indica si la firma contenida es PAdES-EPES (verdadero) o no (falso). Como parámetros recibe:

- **signatureDictionary:** Parámetro que representa el diccionario de firma.

```
public static boolean isPAdESBES(PDFSignatureDictionary signatureDictionary)
```

Método que indica si la firma contenida en un diccionario de firma de un documento PDF es PAdES-BES. Devuelve un valor lógico que indica si la firma contenida es PAdES-BES (verdadero) o no (falso). Como parámetros recibe:

- **signatureDictionary:** Parámetro que representa el diccionario de firma.

```
public static boolean isPAdESBasic(PDFSignatureDictionary signatureDictionary)
```

Método que indica si la firma contenida en un diccionario de firma de un documento PDF es PAdES-Basic. Devuelve un valor lógico que indica si la firma contenida es PAdES-Basic (verdadero) o no (falso). Como parámetros recibe:

- **signatureDictionary:** Parámetro que representa el diccionario de firma.

```
public static boolean isPDF(PDFSignatureDictionary signatureDictionary)
```

Método que indica si la firma contenida en un diccionario de firma de un documento PDF es PDF. Devuelve un valor lógico que indica si la firma contenida es PDF (verdadero) o no (falso). Como parámetros recibe:

- **signatureDictionary:** Parámetro que representa el diccionario de firma.

```
public static boolean hasSignaturePolicyIdentifier(SignerInformation
signerInformation)
```

Método que indica si una firma ASN.1 contiene el atributo SignaturePolicyIdentifier. Devuelve un valor lógico que indica si en la información del firmante se encuentra el atributo SignaturePolicyIdentifier (verdadero) o no (falso). Como parámetros recibe:

- **signerInformation:** Parámetro que representa la información asociada al firmante de la firma ASN.1.

```
public static boolean hasSignaturePolicyIdentifier(Element dsSignature)
```

Método que indica si una firma XML contiene el elemento <xades:SignaturePolicyIdentifier>. Devuelve un valor lógico que indica si dentro de los elementos no firmados de la firma se encuentra el elemento <xades:SignaturePolicyIdentifier> (verdadero) o no (falso). Como parámetros recibe:

- **dsSignature:** Parámetro que representa el elemento <ds:Signature> de la firma XML.

Clase OriginalSignedData

Clase que contiene la información acerca de los datos originalmente firmados.

```
public final byte[] getSignedData()
```

Método que devuelve los datos firmados.

```
public final byte[] getHashSignedData()
```

Método que devuelve el resumen de los datos firmados.

```
public final byte[] getHashAlgorithm()
```

Método que devuelve el algoritmo de resumen utilizado para calcular el resumen de los datos firmados.

```
public final byte[] getMimetype()
```

Método que devuelve el valor del mimetype del documento firmado.

9.4.2 Paquete es.gob.afirma.signature.validation

Este paquete contiene todas las clases relacionadas con la información acerca de procesos de validación sobre firmantes y contra-firmantes.

Interfaz ISignatureValidationTaskID

Interfaz que define todos los identificadores de tareas de validación soportadas para ejecutar sobre un firmante o contra-firmante de una firma. Está compuesto por los atributos constantes:

```
Long ID_SIGNATURE_CORE_VALIDATION
```

Constante que identifica la tarea de validación del núcleo de firma para un firmante o contra-firmante.

```
Long ID_PUBLIC_KEY_INFO_VALIDATION
```

Constante que identifica la tarea de validación de la información de clave pública para un firmante o contra-firmante.

```
Long ID_SIGNING_TIME_VALIDATION
```

Constante que identifica la tarea de validación del instante de firma para un firmante o contra-firmante.

```
Long ID_SIGNATURE_POLICY_VALIDATION
```

Constante que identifica la tarea de validación de la política de firma para un firmante o contra-firmante.

```
Long ID_SIGNING_CERTIFICATE_VALIDATION
```

Constante que identifica la tarea de validación del certificado firmante para un firmante o contra-firmante.

```
Long ID_SIGNATURE_TIME_STAMP_ATTRIBUTES_VALIDATION
```

Constante que identifica la tarea de validación de los atributos signature-time-stamp asociados a un firmante o contra-firmante.

```
Long ID_SIGNATURE_TIME_STAMP_ELEMENTS_VALIDATION
```

Constante que identifica la tarea de validación de los elementos SignatureTimeStamp asociados a un firmante o contra-firmante.

```
Long ID_PDF_STRUCTURALLY_VALIDATION
```

Constante que identifica la tarea de validación estructural de un diccionario de firma contenido en un documento PDF firmado.

Interfaz ITimestampValidationTaskID

Interfaz que define todos los identificadores de tareas de validación soportadas para ejecutar sobre un sello de tiempo. Está compuesto por los atributos constantes:

```
Long ID_TIMESTAMP_SIGNATURE_VALIDATION
```

Constante que identifica la tarea de validación de la firma del sello de tiempo.

```
Long ID_STAMPED_DATA_VALIDATION
```

Constante que identifica la tarea de validación de los datos sellados por el sello de tiempo.

```
Long ID_SIGNING_CERTIFICATE_VALIDATION
```

Constante que identifica la tarea de validación del certificado firmante del sello de tiempo.

```
Long ID_REFERENCES_VALIDATION
```

Constante que identifica la tarea de validación de las referencias contenidas en un sello de tiempo XML.

```
Long ID_PDF_STRUCTURALLY_VALIDATION
```

Constante que identifica la tarea de validación estructural de un diccionario de sello de tiempo contenido en un documento PDF firmado.

```
Long ID_SIGNING_TIME_VALIDATION
```

Constante que identifica la tarea de validación de la fecha de generación del sello de tiempo que compone un diccionario de sello de tiempo contenido en un documento PDF firmado.

Clase ValidationResult

Clase que contiene toda la información relativa al resultado de un proceso de validación de firma. Esta clase es sólo aplicable a procesos de validación de firmas que no sean PDF. Está compuesta por los métodos:

```
public final boolean isIntegrallyCorrect()
```

Método que indica si una firma es íntegramente correcta. Devuelve un valor lógico que indica si la integridad de la firma es válida (verdadero) o no (falso).

```
public final void setIntegrallyCorrect(boolean isIntegrallyCorrectParam)
```

Método que establece si una firma es íntegramente correcta. Como parámetros recibe:

- **isIntegrallyCorrectParam:** Parámetro que indica si la integridad de la firma es válida (verdadero) o no (falso).

```
public final String getErrorMsg()
```

Método que devuelve la descripción del error producido en el proceso de validación de la firma, en caso de que ésta no sea válida.

```
public final void setErrorMsg(String errorMsgParam)
```

Método que establece la descripción del error producido en el proceso de validación de la firma. Como parámetros recibe:

- **errorMsgParam:** Parámetro que representa la descripción del error producido en el proceso de validación de la firma.

```
public final boolean isCorrect()
```

Método que indica si el resultado del proceso de validación de la firma ha sido correcto. Devuelve un valor lógico que indica si la firma es válida (verdadero) o no (falso).

```
public final void setIsCorrect(boolean statusParam)
```

Método que establece el resultado del proceso de validación de la firma. Como parámetros recibe:

- **statusParam:** Parámetro que indica si la firma es válida (verdadero) o no (falso).

```
public final List<SignerValidationResult> getListSignersValidationResults()
```

Método que devuelve una lista con información de validación para cada firmante contenido en la firma.

```
public final void setListSignersValidationResults(List<SignerValidationResult>
listSignersValidationResultsParam)
```

Método que establece la lista con los elementos que representan el resultado de validación de cada firmante contenido en la firma. Como parámetros recibe:

- **listSignersValidationResultsParam:** Parámetro que representa la lista con la información de validación para cada firmante contenido en la firma.

Clase SignerValidationResult

Clase que contiene toda la información relacionada con el resultado del proceso de validación de un firmante o contra-firmante contenido en una firma que no es PDF. Está compuesta por los métodos:

```
public final String getFormat()
```

Método que devuelve el formato de firma asociado al firmante o contra-firmante.

```
public final void setFormat(String formatParam)
```

Método que establece el formato de firma asociado al firmante o contra-firmante. Como parámetros recibe:

- **formatParam:** Parámetro que representa el formato de firma asociado al firmante o contra-firmante.

```
public final boolean isCorrect()
```

Método que indica si el proceso de validación del firmante o contra-firmante ha sido correcto. Devuelve un valor lógico que indica si el firmante o contra-firmante es válido (verdadero) o no (falso).

```
public final void setCorrect(boolean isCorrectParam)
```

Método que establece si el proceso de validación del firmante o contra-firmante ha sido correcto. Como parámetros recibe:

- **isCorrectParam:** Parámetro que indica si el firmante o contra-firmante es válido (verdadero) o no (falso).

```
public final X509Certificate getSigningCertificate()
```

Método que devuelve un objeto que representa el certificado del firmante o contra-firmante.

```
public final void setSigningCertificate(X509Certificate signingCertificateParam)
```

Método que establece el certificado del firmante o contra-firmante. Como parámetros recibe:

- **signingCertificateParam:** Parámetro que representa el certificado del firmante o contra-firmante.

```
public final List<ValidationInfo> getListValidations()
```

Método que devuelve una lista con información relacionada de cada validación aplicada sobre el firmante o contra-firmante.

```
public final void setListValidations(List<ValidationInfo> listValidationsParam)
```

Método que establece una lista con las validaciones aplicadas sobre el firmante o contra-firmante. Como parámetros recibe:

- **listValidationsParam:** Parámetro que representa una lista con información relacionada de cada validación aplicada sobre el firmante o contra-firmante.

```
public final List<TimestampValidationResult> getListTimestampsValidations()
```

Método que devuelve una lista con información relacionada de cada validación aplicada sobre los sellos de tiempo contenidos en atributos signature-time-stamp (si el firmante o contra-firmante es ASN.1) o en elementos SignatureTimeStamp (si el firmante o contra-firmante es XML).

```
public final void setListTimestampsValidations(List<TimestampValidationResult>
listTimestampsValidationsParam)
```

Método que establece una lista con las validaciones aplicadas sobre los sellos de tiempo contenidos en atributos signature-time-stamp (si el firmante o contra-firmante es ASN.1) o en elementos SignatureTimeStamp (si el firmante o contra-firmante es XML). Como parámetros recibe:

- **listTimestampsValidationsParam:** Parámetro que representa una lista con información relacionada de cada validación aplicada sobre los sellos de tiempo contenidos en atributos signature-time-stamp (si el firmante o contra-firmante es ASN.1) o en elementos SignatureTimeStamp (si el firmante o contra-firmante es XML).

```
public final List<SignerValidationResult>
getListCounterSignersValidationsResults()
```

Método que devuelve una lista con información relacionada de cada validación aplicada sobre los contra-firmantes que posee este firmante o contra-firmante.

```
public final void
setListCounterSignersValidationsResults(List<SignerValidationResult>
listCounterSignersValidationsResultsParam)
```

Método que establece una lista con las validaciones aplicadas sobre los contra-firmantes que posee el firmante o contra-firmante. Como parámetros recibe:

- **listCounterSignersValidationsResultsParam:** Parámetro que representa una lista con información relacionada de cada validación aplicada sobre los contra-firmantes que posee este firmante o contra-firmante.

```
public final String getErrorMsg()
```

Método que devuelve la descripción del error producido en el proceso de validación del firmante o contra-firmante, en caso de que éste no sea válido.

```
public final void setErrorMsg(String errorMsgParam)
```

Método que establece la descripción del error producido en el proceso de validación del firmante o contra-firmante. Como parámetros recibe:

- **errorMsgParam:** Parámetro que representa la descripción del error producido en el proceso de validación del firmante o contra-firmante.

Clase ValidationInfo

Clase que representa la información relacionada con una tarea de validación de firma aplicada sobre un firmante o contra-firmante. Está compuesta por los métodos:

```
public final Long getIdValidationTask()
```

Método que devuelve el identificador asociado a la tarea de validación. Los identificadores están registrados en la Interfaz ISignatureValidationTaskID.

```
public final void setIdValidationTask(Long idValidationTaskParam)
```

Método que establece el identificador asociado a la tarea de validación. Como parámetros recibe:

- **idValidationTaskParam:** Parámetro que representa el identificador asociado a la tarea de validación. Los identificadores admitidos están registrados en la Interfaz ISignatureValidationTaskID.

```
public final boolean isSucess()
```

Método que indica si la validación ha finalizado correctamente. Devuelve un valor lógico que indica si la tarea de validación ha finalizado sin errores (verdadero) o no (falso).

```
public final void setSucess(boolean isSucessParam)
```

Método que establece si la validación ha finalizado correctamente. Como parámetros recibe:

- **isSucessParam:** Parámetro que indica si la tarea de validación ha finalizado sin errores (verdadero) o no (falso).

```
public final String getErrorMsg()
```

Método que devuelve la descripción del error producido por el cúal la validación no se considera correcta.

```
public final void setErrorMsg(String errorMsgParam)
```

Método que establece la descripción del error producido por el cúal la validación no se considera correcta. Como parámetros recibe:

- **errorMsgParam:** Parámetro que representa la descripción del error producido por el cúal la validación no se considera correcta.

Clase TimestampValidationResult

Clase que contiene toda la información relacionada con el resultado del proceso de validación de un sello de tiempo contenido en un atributo signature-time-stamp (si la firma es ASN.1) o en un elemento SignatureTimeStamp (si la firma es XML) asociado a un firmante o contra-firmante. Está compuesta por los métodos:

```
public final boolean isXML()
```

Método que indica si el sello de tiempo es de tipo XML. Devuelve un valor lógico que indica si el sello de tiempo es XML (verdadero) o ASN.1 (falso).

```
public final void setXML(boolean isXMLParam)
```

Método que establece si el sello de tiempo es de tipo XML. Como parámetros recibe:

- **isXMLParam:** Parámetro que indica si el sello de tiempo es XML (verdadero) o ASN.1 (falso).

```
public final boolean isCorrect()
```

Método que indica si el proceso de validación del sello de tiempo ha sido correcto. Devuelve un valor lógico que indica si el sello de tiempo es válido (verdadero) o no (falso).

```
public final void setCorrect(boolean isCorrectParam)
```

Método que establece si el proceso de validación del sello de tiempo ha sido correcto. Como parámetros recibe:

- **isCorrectParam:** Parámetro que indica si el sello de tiempo es válido (verdadero) o no (falso).

```
public final X509Certificate getSigningCertificate()
```

Método que devuelve un objeto que representa el certificado firmante del sello de tiempo.

```
public final void setSigningCertificate(X509Certificate signingCertificateParam)
```

Método que establece el certificado firmante del sello de tiempo. Como parámetros recibe:

- **signingCertificateParam:** Parámetro que representa el certificado firmante del sello de tiempo.

```
public final List<TimeStampValidationInfo> getListValidations()
```

Método que devuelve una lista con información relacionada de cada validación aplicada sobre el sello de tiempo.

```
public final void setListValidations(List<TimeStampValidationInfo>
listValidationsParam)
```

Método que establece una lista con información relacionada de cada validación aplicada sobre el sello de tiempo. Como parámetros recibe:

- **listValidationsParam:** Parámetro que representa una lista con información relacionada de cada validación aplicada sobre el sello de tiempo.

Clase TimeStampValidationInfo

Clase que representa la información relacionada con una tarea de validación de firma aplicada sobre un sello de tiempo contenido en un atributo signature-time-stamp o un elemento SignatureTimeStamp, asociado a un firmante o contra-firmante. Está compuesta por los métodos:

```
public final Long getIdValidationTask()
```

Método que devuelve el identificador asociado a la tarea de validación. Los identificadores están registrados en la Interfaz ITimestampValidationTaskID.

```
public final void setIdValidationTask(Long idValidationTaskParam)
```

Método que establece el identificador asociado a la tarea de validación. Como parámetros recibe:

- **idValidationTaskParam:** Parámetro que representa el identificador asociado a la tarea de validación. Los identificadores están registrados en la Interfaz ITimestampValidationTaskID.

```
public final boolean isSucess()
```

Método que indica si la validación ha finalizado correctamente. Devuelve un valor lógico que indica si la tarea de validación ha finalizado sin errores (verdadero) o no (falso).

```
public final void setSucess(boolean isSucessParam)
```

Método que establece si la validación ha finalizado correctamente. Como parámetros recibe:

- **isSucessParam:** Parámetro que indica si la tarea de validación ha finalizado sin errores (verdadero) o no (falso).

```
public final String getErrorMsg()
```

Método que devuelve la descripción del error producido por el cúal la validación no se considera correcta.

```
public final void setErrorMsg(String errorMsgParam)
```

Método que establece la descripción del error producido por el cúal la validación no se considera correcta. Como parámetros recibe:

- **errorMsgParam:** Parámetro que representa la descripción del error producido por el cúal la validación no se considera correcta.

Clase PDFValidationResult

Clase que contiene toda la información relativa al resultado de un proceso de validación de firma. Esta clase es sólo aplicable a procesos de validación de documentos PDF firmados. Está compuesta por los métodos:

```
public final boolean isIntegrallyCorrect()
```

Método que indica si el documento PDF es íntegramente correcto. Devuelve un valor lógico que indica si la integridad del documento PDF es válida (verdadero) o no (falso).

```
public final void setIntegrallyCorrect(boolean isIntegrallyCorrectParam)
```

Método que establece si el documento PDF es íntegramente correcto. Como parámetros recibe:

- **isIntegrallyCorrectParam:** Parámetro que indica si la integridad del documento PDF es válida (verdadero) o no (falso).

```
public final String getErrorMsg()
```

Método que devuelve la descripción del error producido en el proceso de validación del documento PDF firmado, en caso de que éste no sea válido.

```
public final void setErrorMsg(String errorMsgParam)
```

Método que establece la descripción del error producido en el proceso de validación del documento PDF firmado. Como parámetros recibe:

- **errorMsgParam:** Parámetro que representa la descripción del error producido en el proceso de validación del documento PDF firmado.

```
public final boolean isCorrect()
```

Método que indica si el resultado del proceso de validación del documento PDF firmado ha sido correcto, o lo que es lo mismo, si todos sus diccionarios de firma y de sello de tiempo contenidos son válidos. Devuelve un valor lógico que indica si el documento PDF firmado es válido (verdadero) o no (falso).

```
public final void setCorrect(boolean isCorrectParam)
```

Método que establece si el resultado del proceso de validación del documento PDF firmado ha sido correcto, o lo que es lo mismo, si todos sus diccionarios de firma y de sello de tiempo contenidos son válidos. Como parámetros recibe:

- **isCorrectParam:** Parámetro que indica si el documento PDF firmado es válido (verdadero) o no (falso).

```
public final List<PDFSignatureDictionaryValidationResult>
getListPDFSignatureDictionariesValidationResults()
```

Método que devuelve una lista con información relacionada con las validaciones aplicadas sobre los diccionarios de firma contenidos en el documento PDF.

```
public final void
setListPDFSignatureDictionariesValidationResults(List<PDFSignatureDictionaryValida
tionResult> listPDFSignatureDictionariesValidationResultsParam)
```

Método que establece una lista con información relacionada con las validaciones aplicadas sobre los diccionarios de firma contenidos en el documento PDF. Como parámetros recibe:

- **listPDFSignatureDictionariesValidationResultsParam:** Parámetro que representa una lista con información relacionada con las validaciones aplicadas sobre los diccionarios de firma contenidos en el documento PDF.

```
public final List<PDFDocumentTimeStampDictionaryValidationResult>
getListPDFDocumentTimeStampDictionariesValidationResults()
```

Método que devuelve una lista con información relacionada con las validaciones aplicadas sobre los diccionarios de sello de tiempo contenidos en el documento PDF.

```
public final void
setListPDFDocumentTimeStampDictionariesValidationResults(List<PDFDocumentTimeStamp
DictionaryValidationResult>
listPDFDocumentTimeStampDictionariesValidationResultsParam)
```

Método que establece una lista con información relacionada con las validaciones aplicadas sobre los diccionarios de sello de tiempo contenidos en el documento PDF. Como parámetros recibe:

- **listPDFDocumentTimeStampDictionariesValidationResultsParam:** Parámetro que representa una lista con información relacionada con las validaciones aplicadas sobre los diccionarios de sello de tiempo contenidos en el documento PDF.

```
public final String getSignatureFormat()
```

Método que devuelve el formato de firma asociado al documento PDF firmado.

```
public final void setSignatureFormat(String signatureFormatParam)
```

Método que establece el formato de firma asociado al documento PDF firmado. Como parámetros recibe:

- **signatureFormatParam:** Parámetro que representa el formato de firma asociado al documento PDF firmado.

Clase PDFSignatureDictionaryValidationResult

Clase que contiene toda la información relacionada con el resultado de un proceso de validación aplicado sobre un diccionario de firma contenido en un documento PDF. Está compuesta por los métodos:

```
public final String getErrorMsg()
```

Método que devuelve la descripción del error producido en el proceso de validación del diccionario de firma, en caso de que éste no sea válido.

```
public final void setErrorMsg(String errorMsgParam)
```

Método que establece la descripción del error producido en el proceso de validación del diccionario de firma. Como parámetros recibe:

- **errorMsgParam:** Parámetro que representa la descripción del error producido en el proceso de validación del diccionario de firma.

```
public final boolean isCorrect()
```

Método que indica si el resultado del proceso de validación del diccionario de firma ha sido correcto. Devuelve un valor lógico que indica si el diccionario de firma es válido (verdadero) o no (falso).

```
public final void setCorrect(boolean isCorrectParam)
```

Método que establece si el resultado del proceso de validación del diccionario de firma ha sido correcto. Como parámetros recibe:

- **isCorrectParam:** Parámetro que indica si el diccionario de firma es válido (verdadero) o no (falso).

```
public final String getDictionaryName()
```

Método que devuelve el nombre del diccionario de firma.

```
public final void setDictionaryName(String dictionaryNameParam)
```

Método que establece el nombre del diccionario de firma. Como parámetros recibe:

- **dictionaryNameParam:** Parámetro que representa el nombre del diccionario de firma.

```
public final X509Certificate getSigningCertificate()
```

Método que devuelve un objeto que representa el certificado del único firmante contenido en el diccionario de firma.

```
public final void setSigningCertificate(X509Certificate signingCertificateParam)
```

Método que establece el certificado del único firmante contenido en el diccionario de firma. Como parámetros recibe:

- **signingCertificateParam:** Parámetro que representa el certificado del único firmante contenido en el diccionario de firma.

```
public final List<ValidationInfo> getListValidations()
```

Método que devuelve una lista con información acerca de las validaciones aplicadas sobre el diccionario de firma.

```
public final void setListValidations(List<ValidationInfo> listValidationsParam)
```

Método que establece una lista con información acerca de las validaciones aplicadas sobre el diccionario de firma. Como parámetros recibe:

- **listValidationsParam:** Parámetro que representa una lista con información acerca de las validaciones aplicadas sobre el diccionario de firma.

```
public final List<TimestampValidationResult> getListTimestampsValidations()
```

Método que devuelve una lista con información relacionada de cada validación aplicada sobre los sellos de tiempo contenidos en atributos signature-time-stamp del firmante que constituye el diccionario de firma.

```
public final void setListTimestampsValidations(List<TimestampValidationResult>
listTimestampsValidationsParam)
```

Método que establece una lista con información relacionada de cada validación aplicada sobre los sellos de tiempo contenidos en atributos signature-time-stamp del firmante que constituye el diccionario de firma. Como parámetros recibe:

- **listTimestampsValidationsParam:** Parámetro que representa una lista con información relacionada de cada validación aplicada sobre los sellos de tiempo contenidos en atributos signature-time-stamp del firmante que constituye el diccionario de firma.

Clase PDFDocumentTimeStampDictionaryValidationResult

Clase que contiene toda la información relacionada con el resultado de un proceso de validación aplicado sobre un diccionario de sello de tiempo contenido en un documento PDF. Está compuesta por los métodos:

```
public final String getErrorMsg()
```

Método que devuelve la descripción del error producido en el proceso de validación del diccionario de sello de tiempo, en caso de que éste no sea válido.

```
public final void setErrorMsg(String errorMsgParam)
```

Método que establece la descripción del error producido en el proceso de validación del diccionario de sello de tiempo. Como parámetros recibe:

- **errorMsgParam:** Parámetro que representa la descripción del error producido en el proceso de validación del diccionario de sello de tiempo.

```
public final boolean isCorrect()
```

Método que indica si el resultado del proceso de validación del diccionario de sello de tiempo ha sido correcto. Devuelve un valor lógico que indica si el diccionario de sello de tiempo es válido (verdadero) o no (falso).

```
public final void setCorrect(boolean isCorrectParam)
```

Método que establece si el resultado del proceso de validación del diccionario de sello de tiempo ha sido correcto. Como parámetros recibe:

- **isCorrectParam:** Parámetro que indica si el diccionario de sello de tiempo es válido (verdadero) o no (falso).

```
public final X509Certificate getSigningCertificate()
```

Método que devuelve un objeto que representa el certificado firmante del sello de tiempo contenido en el diccionario de sello de tiempo.

```
public final void setSigningCertificate(X509Certificate signingCertificateParam)
```

Método que establece el certificado firmante del sello de tiempo contenido en el diccionario de sello de tiempo. Como parámetros recibe:

- **signingCertificateParam:** Parámetro que representa el certificado firmante del sello de tiempo contenido en el diccionario de sello de tiempo.

```
public final List<TimeStampValidationInfo> getListValidations()
```

Método que devuelve una lista con información acerca de las validaciones aplicadas sobre el diccionario de sello de tiempo.

```
public final void setListValidations(List<TimeStampValidationInfo>
listValidationsParam)
```

Método que establece una lista con información acerca de las validaciones aplicadas sobre el diccionario de sello de tiempo. Como parámetros recibe:

- **listValidationsParam:** Parámetro que representa una lista con información acerca de las validaciones aplicadas sobre el diccionario de sello de tiempo.

```
public final String getDictionaryName()
```

Método que devuelve el nombre del diccionario de sello de tiempo.

```
public final void setDictionaryName(String dictionaryNameParam)
```

Método que establece el nombre del diccionario de sello de tiempo. Como parámetros recibe:

- **dictionaryNameParam:** Parámetro que representa el nombre del diccionario de sello de tiempo.

9.4.3 Paquete es.gob.afirma.signature.cades

Este paquete contiene todas las clases asociadas a funcionalidades de generación, validación y actualización de firmas CAdES (Baseline o no). De este paquete se describirán aquellas clases más destacadas.

Clase CadesSigner

Esta clase gestiona todas las operaciones asociadas a firmas CAdES no Baseline. Implementa la interfaz *Signer* descrita en el punto 0 Está compuesta por los métodos:

```
public byte[ ] sign(byte[ ] data, String algorithm, String signatureFormat,
PrivateKeyEntry privateKey, Properties extraParams, boolean includeTimestamp,
String signatureForm, String signaturePolicyID) throws SigningException
```

Método definido en la Interfaz Signer.

```
public byte[ ] coSign(byte[ ] signature, byte[ ] document, String algorithm,
PrivateKeyEntry privateKey, Properties extraParams, boolean includeTimestamp,
String signatureForm, String signaturePolicyID) throws SigningException
```

Método definido en la Interfaz Signer.

```
public byte[ ] counterSign(byte[ ] signature, String algorithm, PrivateKeyEntry
privateKey, Properties extraParams, boolean includeTimestamp, String
signatureForm, String signaturePolicyID) throws SigningException
```

Método definido en la Interfaz Signer.

```
public ValidationResult verifySignature(byte[ ] eSignature, byte[ ] document)
```

Método que valida el conjunto de firmantes de una firma CAdES no Baseline. Devuelve un objeto que contiene toda la información asociada al proceso de validación. En el caso de que la firma sea CAdES - EPES será necesario haber configurado previamente el fichero **integra.properties** para poder hacer una validación de la política de firma en base a alguna de las políticas de firma definidas en dicho fichero. Por otro lado, si en el fichero **integra.properties** el nivel de validación para los certificados está definido como *Completo*, se llevará a cabo la validación del estado de revocación del certificado firmante vía OCSP en base a los parámetros configurados en el mismo fichero.

Se validarán todos los certificados firmantes respecto a la fecha actual (si carecen de sello de tiempo), o respecto a la fecha del sello de tiempo asociado (si poseen sello de tiempo), en base al valor indicado en la propiedad **CERTIFICATE_VALIDATION_LEVEL** del archivo **integra.properties**, y el núcleo del sello de tiempo de aquellos firmantes que lo incluyan.

Como parámetros recibe:

- **eSignature:** Parámetro que representa la firma CAdES a validar.
- **document:** Parámetro que representa el documento original firmado por la firma a validar.

```
public byte[ ] upgrade(byte[ ] signature, List<X509Certificate> listCertificates)
throws SigningException
```

Método definido en la Interfaz Signer.

```
public OriginalSignedData getSignedData(byte[] signature) throws SigningException
```

Método definido en la Interfaz Signer.

Clase CAdESBaselineSigner

Esta clase gestiona todas las operaciones asociadas a firmas CAdES Baseline. Implementa la interfaz *Signer* descrita en el punto 0 Está compuesta por los métodos:

```
public byte[ ] sign(byte[ ] data, String algorithm, String signatureFormat,
PrivateKeyEntry privateKey, Properties extraParams, boolean includeTimestamp,
String signatureForm, String signaturePolicyID) throws SigningException
```

Método definido en la Interfaz Signer.

```
public byte[ ] coSign(byte[ ] signature, byte[ ] document, String algorithm,
PrivateKeyEntry privateKey, Properties extraParams, boolean includeTimestamp,
String signatureForm, String signaturePolicyID) throws SigningException
```

Método definido en la Interfaz Signer.

```
public byte[ ] counterSign(byte[ ] signature, String algorithm, PrivateKeyEntry
privateKey, Properties extraParams, boolean includeTimestamp, String
signatureForm, String signaturePolicyID) throws SigningException
```

Método definido en la Interfaz Signer.

```
public ValidationResult verifySignature(byte[ ] eSignature, byte[ ] document)
```

Método que valida el conjunto de firmantes de una firma CAdES Baseline. Devuelve un objeto que contiene toda la información asociada al proceso de validación. En el caso de que la firma contenga política de firma será necesario haber configurado previamente el fichero **integra.properties** para poder hacer una validación de la política de firma en base a alguna de las políticas de firma definidas en dicho fichero. Por otro lado, si en el fichero **integra.properties** el nivel de validación para los certificados está definido como *Completo*, se llevará a cabo la validación del estado de revocación del certificado firmante vía OCSP en base a los parámetros configurados.

Se validarán todos los certificados firmantes respecto a la fecha actual (si carecen de sello de tiempo), o respecto a la fecha del sello de tiempo asociado (si poseen sello de tiempo), en base al valor indicado en la propiedad **CERTIFICATE_VALIDATION_LEVEL** del archivo **integra.properties**, y el núcleo del sello de tiempo de aquellos firmantes que lo incluyan.

Como parámetros recibe:

- **eSignature:** Parámetro que representa la firma CAdES a validar.
- **document:** Parámetro que representa el documento original firmado por la firma a validar.

```
public byte[ ] upgrade(byte[ ] signature, List<X509Certificate> listCertificates)
throws SigningException
```

Método definido en la Interfaz Signer.

```
public OriginalSignedData getSignedData(byte[] signature) throws SigningException
```

Método definido en la Interfaz Signer.

9.4.4 es.gob.afirma.signature.pades

Este paquete contiene todas las clases asociadas a funcionalidades de generación, validación y actualización de firmas PAdES. De este paquete se describirán aquellas clases más destacadas.

Clase PadesSigner

Esta clase gestiona todas las operaciones asociadas a firmas PAdES no Baseline. Implementa la interfaz *Signer* descrita en el punto 0. Está compuesta por los métodos:

```
public byte[ ] sign(byte[ ] data, String algorithm, String signatureFormat,
PrivateKeyEntry privateKey, Properties extraParams, boolean includeTimestamp,
String signatureForm, String signaturePolicyID) throws SigningException
```

Método definido en la Interfaz Signer.

```
public byte[ ] coSign(byte[ ] signature, byte[ ] document, String algorithm,
PrivateKeyEntry privateKey, Properties extraParams, boolean includeTimestamp,
String signatureForm, String signaturePolicyID) throws SigningException
```

Método definido en la Interfaz Signer. El método lanza directamente la excepción *SigningException* ya que no está soportado para firmas PAdES no Baseline.

```
public byte[ ] counterSign(byte[ ] signature, String algorithm, PrivateKeyEntry
privateKey, Properties extraParams, boolean includeTimestamp, String
signatureForm, String signaturePolicyID) throws SigningException
```

Método definido en la Interfaz Signer. El método lanza directamente la excepción *SigningException* ya que no está soportado para firmas PAdES no Baseline.

```
public PDFValidationResult verifySignature(byte[ ] pdfDocument)
```

Método que valida un documento PDF firmado, incluyendo sus diccionarios de firma y sus diccionarios de sello de tiempo. Devuelve un objeto que contiene toda la información asociada al proceso de validación. En el caso de que alguna de las firmas contenidas contenga un firmante que incluya política de firma entre sus atributos firmados será necesario haber configurado previamente el fichero **integra.properties** para poder hacer una validación de la política de firma en base a alguna de las políticas de firma definidas en dicho fichero. Por otro lado, si en el fichero **integra.properties** el

nivel de validación para los certificados está definido como *Completo*, se llevará a cabo la validación del estado de revocación del certificado firmante vía OCSP en base a los parámetros configurados.

Si el documento PDF incluye diccionarios de sello de tiempo, se validarán todos los certificados firmantes respecto a la fecha del diccionario de sello de tiempo más reciente en base al valor indicado en la propiedad **CERTIFICATE_VALIDATION_LEVEL** del archivo **integra.properties**, el núcleo de los sellos de tiempo de aquellos firmantes que posean sello de tiempo, y el núcleo del sello de tiempo de los sellos de tiempo incluidos en los diccionarios de sello de tiempo. Si el documento PDF no incluye diccionarios de sello de tiempo, se validarán todos los certificados firmantes respecto a la fecha actual (si carecen de sello de tiempo), o respecto a la fecha del sello de tiempo asociado (si poseen sello de tiempo), en base al valor indicado en la propiedad **CERTIFICATE_VALIDATION_LEVEL** del archivo **integra.properties**, y el núcleo del sello de tiempo de aquellos firmantes que lo incluyan.

Como parámetros recibe:

- **pdfDocument:** Parámetro que representa el documento PDF que contiene las firmas a validar.

```
public byte[ ] upgrade(byte[ ] signature, List<X509Certificate> listCertificates)
throws SigningException
```

Método definido en la Interfaz Signer.

```
public void setCalledByFacade(boolean isCalledByFacadeParam)
```

Método que establece el valor de la propiedad de clase que indica si las operaciones definidas en la clase han sido invocadas desde la fachada de invocación o no.

Como parámetros recibe:

- **isCalledByFacadeParam:** Parámetro que indica si las operaciones definidas en la clase han sido invocadas desde la fachada de invocación (verdadero) o no (falso).

```
public OriginalSignedData getSignedData(byte[] signature) throws SigningException
```

Método definido en la Interfaz Signer.

Clase PAdESBaselineSigner

Esta clase gestiona todas las operaciones asociadas a firmas PAdES Baseline. Implementa la interfaz *Signer* descrita en el punto 0. Está compuesta por los métodos:

```
public byte[ ] sign(byte[ ] data, String algorithm, String signatureFormat,
PrivateKeyEntry privateKey, Properties extraParams, boolean includeTimestamp,
String signatureForm, String signaturePolicyID) throws SigningException
```

Método definido en la Interfaz Signer.

```
public byte[ ] coSign(byte[ ] signature, byte[ ] document, String algorithm,
PrivateKeyEntry privateKey, Properties extraParams, boolean includeTimestamp,
String signatureForm, String signaturePolicyID) throws SigningException
```

Método definido en la Interfaz Signer. El método lanza directamente la excepción *SigningException* ya que no está soportado para firmas PAdES Baseline.

```
public byte[ ] counterSign(byte[ ] signature, String algorithm, PrivateKeyEntry
privateKey, Properties extraParams, boolean includeTimestamp, String
signatureForm, String signaturePolicyID) throws SigningException
```

Método definido en la Interfaz Signer. El método lanza directamente la excepción *SigningException* ya que no está soportado para firmas PAdES Baseline.

```
public PDFValidationResult verifySignature(byte[ ] pdfDocument)
```

Método que valida un documento PDF firmado, incluyendo sus diccionarios de firma y sus diccionarios de sello de tiempo. Devuelve un objeto que contiene toda la información asociada al proceso de validación. En el caso de que alguna de las firmas contenidas incluya política de firma en su núcleo de firma CAdES será necesario haber configurado previamente el fichero **integra.properties** para poder hacer una validación de la política de firma en base a alguna de las políticas de firma definidas en dicho fichero. Por otro lado, si en el fichero **integra.properties** el nivel de validación para los certificados está definido como *Completo*, se llevará a cabo la validación del estado de revocación del certificado firmante vía OCSP en base a los parámetros configurados.

Si el documento PDF incluye diccionarios de sello de tiempo, se validarán todos los certificados firmantes respecto a la fecha del diccionario de sello de tiempo más reciente en base al valor indicado en la propiedad **CERTIFICATE_VALIDATION_LEVEL** del archivo **integra.properties**, el núcleo de los sellos de tiempo de aquellos firmantes que posean sello de tiempo, y el núcleo del sello de tiempo de los sellos de tiempo incluidos en los diccionarios de sello de tiempo. Si el documento PDF no incluye diccionarios de sello de tiempo, se validarán todos los certificados firmantes respecto a la fecha actual (si carecen de sello de tiempo), o respecto a la fecha del sello de tiempo asociado (si poseen sello de tiempo), en base al valor indicado en la propiedad **CERTIFICATE_VALIDATION_LEVEL** del archivo **integra.properties**, y el núcleo del sello de tiempo de aquellos firmantes que lo incluyan.

Como parámetros recibe:

- **pdfDocument:** Parámetro que representa el documento PDF que contiene las firmas a validar.

```
public byte[ ] upgrade(byte[ ] signature, List<X509Certificate> listCertificates)
throws SigningException
```

Método definido en la Interfaz Signer.

```
public void setCalledByFacade(boolean isCalledByFacadeParam)
```

Método que establece el valor de la propiedad de clase que indica si las operaciones definidas en la clase han sido invocadas desde la fachada de invocación o no.

Como parámetros recibe:

- **isCalledByFacadeParam:** Parámetro que indica si las operaciones definidas en la clase han sido invocadas desde la fachada de invocación (verdadero) o no (falso).

```
public OriginalSignedData getSignedData(byte[] signature) throws SigningException
```

Método definido en la Interfaz Signer.

9.4.5 Paquete es.gob.afirma.signature.policy

Este paquete contiene todas las clases asociadas a la gestión de políticas de firma. De este paquete se describirán aquellas clases más destacadas.

Clase SignaturePolicyManager

Clase que permite añadir y validar elementos asociados a políticas de firma. Está compuesta por los métodos:

```
public static void addASN1SignPolicy(ASN1EncodableVector contexExpecific, String
qualifier, String policyID, Properties properties, boolean isPAdES) throws
SignaturePolicyException
```

Método que añade el atributo SignaturePolicyIdentifier a una firma CAdES (Baseline o no) o PAdES (Baseline o no). Para poder llevar a cabo esta operación, el archivo **integra.properties** deberá estar configurado correctamente en lo que se refiere a políticas de firma ASN.1 o PDF. En caso de que se produzca un error durante el proceso y no sea posible añadirlo, el método lanzará una excepción *SignaturePolicyException*. Como parámetros recibe:

- **contexExpecific:** Parámetro que representa el conjunto de elementos asociados a un firmante de una firma ASN.1 o PDF.

- **qualifier:** Parámetro que representa el valor del atributo SigPolicyQualifier.

- **policyID:** Parámetro que representa el identificador de la política de firma que añadir. Debe coincidir con alguno de los identificadores de política de firma para firmas ASN.1 o PDF definidos en el archivo **integra.properties**.

- **properties:** Parámetro que representa el conjunto de propiedades definidas en el archivo **integra.properties**.

- **isPAdES:** Parámetro que indica si la firma es PAdES (verdadero) o CAdES (falso).

```
public static void addXMLSignPolicy(XAdES_EPES xades, String qualifier, String
policyID, Properties properties) throws SignaturePolicyException
```

Método que añade el elemento <xades:SignaturePolicyIdentifier> a una firma XAdES (Baseline o no). Para poder llevar a cabo esta operación, el archivo **integra.properties** deberá estar configurado correctamente en lo que se refiere a políticas de firma XML. En caso de que se produzca un error durante el proceso y no sea posible añadirlo, el método lanzará una excepción *SignaturePolicyException*. Como parámetros recibe:

- **xades:** Parámetro que representa la firma XAdES.

- **qualifier:** Parámetro que representa el valor del elemento <xades:SignaturePolicyIdentifier>.

- **policyID:** Parámetro que representa el identificador de la política de firma que añadir. Debe coincidir con alguno de los identificadores de política de firma para firmas XML definidos en el archivo **policy.properties**.

- **properties:** Parámetro que representa el conjunto de propiedades definidas en el archivo **policy.properties**.

```
public static void validateGeneratedPAdESEPESSignature(PdfDictionary
pdfSignatureDictionary, String policyID, Properties properties) throws
SignaturePolicyException
```

Método que valida una firma PAdES (Baseline o no), que contiene un núcleo de firma CAdES con política de firma, generada desde Integr@. Para poder llevar a cabo esta operación, el archivo **integra.properties** deberá estar configurado correctamente en lo que se refiere a políticas de firma PDF. En caso de que se produzca un error durante el proceso o la firma no sea válida según la política de firma asociada, el método lanzará una excepción *SignaturePolicyException*. Como parámetros recibe:

- **pdfSignatureDictionary:** Parámetro que representa el diccionario de firma que contiene la firma PAdES que validar.

- **policyID:** Parámetro que representa el identificador de la política de firma a usar para validar la firma. Debe coincidir con alguno de los identificadores de política de firma para firmas PDF definidos en el archivo **policy.properties**.

- **properties:** Parámetro que representa el conjunto de propiedades definidas en el archivo **policy.properties**.

```
public static void validateGeneratedCAdESEPESSignature(SignerInfo signerInfo,
String policyID, Properties properties, boolean isCounterSigner) throws
SignaturePolicyException
```

Método que valida una firma CAdES (Baseline o no) con política de firma generada desde Integr@. Para poder llevar a cabo esta operación, el archivo **integra.properties** deberá estar configurado correctamente en lo que se refiere a políticas de firma ASN.1. En caso de que se produzca un error durante el proceso o la firma no sea válida según la política de firma asociada, el método lanzará una excepción *SignaturePolicyException*. Como parámetros recibe:

- **signerInfo:** Parámetro que representa los datos del firmante de la firma CAdES que validar.

- **policyID:** Parámetro que representa el identificador de la política de firma a usar para validar la firma. Debe coincidir con alguno de los identificadores de política de firma para firmas ASN.1 definidos en el archivo **integra.properties**.

- **properties:** Parámetro que representa el conjunto de propiedades definidas en el archivo **integra.properties**.

- **isCounterSigner:** Parámetro que indica si el firmante es un contra-firmante (verdadero) o no (falso).

```
public static void validateGeneratedXAdESEPESSignature(Element dsSignature, String
policyID, Properties properties) throws SignaturePolicyException
```

Método que valida una firma XAdES (Baseline o no) con política de firma generada desde Integr@. Para poder llevar a cabo esta operación, el archivo **integra.properties** deberá estar configurado correctamente en lo que se refiere a políticas de firma XML. En caso de que se produzca un error durante el proceso o la firma no sea válida según la política de firma asociada, el método lanzará una excepción *SignaturePolicyException*. Como parámetros recibe:

- **dsSignature:** Parámetro que representa el elemento <ds:Signature> de la firma XAdES que validar.

- **policyID:** Parámetro que representa el identificador de la política de firma a usar para validar la firma. Debe coincidir con alguno de los identificadores de política de firma para firmas XML definidos en el archivo **integra.properties**.

- **properties:** Parámetro que representa el conjunto de propiedades definidas en el archivo **integra.properties**.

```
public static void validatePAdESEPESSignature(SignerInformation signerInformation,
PdfDictionary pdfSignatureDictionary, Properties properties) throws
SignaturePolicyException
```

Método que valida una firma PAdES (Baseline o no) que incluye un núcleo de firma CAdES con política de firma. Para poder llevar a cabo esta operación, el archivo **integra.properties** deberá estar configurado correctamente en lo que se refiere a políticas de firma PDF. En caso de que se produzca un error durante el proceso o la firma no sea válida según la política de firma asociada, el método lanzará una excepción *SignaturePolicyException*. Como parámetros recibe:

- **signerInformation:** Parámetro que representa los datos del firmante de la firma PAdES que validar.
- **pdfSignatureDictionary:** Parámetro que representa el diccionario de firma que contiene la firma PAdES que validar.
- **properties:** Parámetro que representa el conjunto de propiedades definidas en el archivo **integra.properties**.

```
public static void validateCAdESEPESSignature(SignerInformation signerInformation,
Properties properties, boolean isCounterSigner , boolean includeContent) throws
SignaturePolicyException
```

Método que valida una firma CAdES (Baseline o no) que contiene política de firma. Para poder llevar a cabo esta operación, el archivo **integra.properties** deberá estar configurado correctamente en lo que se refiere a políticas de firma ASN.1. En caso de que se produzca un error durante el proceso o la firma no sea válida según la política de firma asociada, el método lanzará una excepción *SignaturePolicyException*. Como parámetros recibe:

- **signerInformation:** Parámetro que representa los datos del firmante de la firma CAdES que validar.
- **properties:** Parámetro que representa el conjunto de propiedades definidas en el archivo **integra.properties**.
- **isCounterSigner:** Parámetro que indica si el firmante es un contra-firmante (verdadero) o no (falso).
- **includeContent:** Parámetro que indica si la firma incluye los datos firmados (verdadero) o no (falso).

```
public static void validateXAdESEPESSignature(Element dsSignature, Properties
properties, String signingMode) throws SignaturePolicyException
```

Método que valida una firma XAdES (Baseline o no) que contiene política de firma. Para poder llevar a cabo esta operación, el archivo **integra.properties** deberá estar configurado correctamente en lo que se refiere a políticas de firma XML. En caso de que se produzca un error durante el proceso o la firma no sea válida según la política de firma asociada, el método lanzará una excepción *SignaturePolicyException*. Como parámetros recibe:

- **dsSignature:** Parámetro que representa el elemento <ds:Signature> de la firma XAdES que validar.
- **properties:** Parámetro que representa el conjunto de propiedades definidas en el archivo **integra.properties**.
- **signingMode:** Parámetro que representa el modo de firma de la firma XAdES que validar. Los valores posibles son:
  - **Detached:** Indica que la firma incluye una referencia a los datos firmados.
  - **Enveloped:** Indica que la firma incluye los datos firmados.
  - **Enveloping:** Indica que la frma posee una estructura XML que contiene los datos firmados en un nodo propio.

```
public static boolean isValidXMLHashAlgorithmByPolicy(String uriAlgorithm, String
policyID, Properties properties)
```

Método que comprueba si la URI de un algoritmo de resumen es válida según una política de firma concreta. Para poder llevar a cabo esta operación, el archivo **integra.properties** deberá estar configurado correctamente en lo que se refiere a políticas de firma XML. El método devuelve un valor lógico que indica si la URI es válida (verdadero) o no (falso). Como parámetros recibe:

- **uriAlgorithm:** Parámetro que representa la URI del algoritmo de hash.
- **policyID:** Parámetro que representa el identificador de la política de firma a usar para validar la URI. Debe coincidir con alguno de los identificadores de política de firma para firmas XML definidos en el archivo **integra.properties**.
- **properties:** Parámetro que representa el conjunto de propiedades definidas en el archivo **integra.properties**.

```
public static boolean isValidASN1HashAlgorithmByPolicy(AlgorithmIdentifier
digestAlgorithmId, String policyID, Properties properties)
```

Método que comprueba si el OID de un algoritmo de resumen es válido según una política de firma concreta. Para poder llevar a cabo esta operación, el archivo **integra.properties** deberá estar

configurado correctamente en lo que se refiere a políticas de firma ASN.1 y PDF. El método devuelve un valor lógico que indica si el OID es válido (verdadero) o no (falso). Como parámetros recibe:

- **digestAlgorithmId:** Parámetro que representa el OID del algoritmo de hash.
- **policyID:** Parámetro que representa el identificador de la política de firma a usar para validar el OID. Debe coincidir con alguno de los identificadores de política de firma para firmas ASN.1 o PDF definidos en el archivo **integra.properties**.
- **properties:** Parámetro que representa el conjunto de propiedades definidas en el archivo **integra.properties**.

```
public static boolean isValidASN1SignAlgorithmByPolicy(AlgorithmIdentifier
signAlgorithmId, String policyID, Properties properties)
```

Método que comprueba si el OID de un algoritmo de firma es válido según una política de firma concreta. Para poder llevar a cabo esta operación, el archivo **integra.properties** deberá estar configurado correctamente en lo que se refiere a políticas de firma ASN.1 y PDF. El método devuelve un valor lógico que indica si el OID es válido (verdadero) o no (falso). Como parámetros recibe:

- **signAlgorithmId:** Parámetro que representa el OID del algoritmo de firma.
- **policyID:** Parámetro que representa el identificador de la política de firma a usar para validar el OID. Debe coincidir con alguno de los identificadores de política de firma para firmas ASN.1 o PDF definidos en el archivo **integra.properties**.
- **properties:** Parámetro que representa el conjunto de propiedades definidas en el archivo **integra.properties**.

```
public static boolean isValidXMLSignAlgorithmByPolicy(String uriAlgorithm, String
policyID, Properties properties)
```

Método que comprueba si la URI de un algoritmo de firma es válida según una política de firma concreta. Para poder llevar a cabo esta operación, el archivo **integra.properties** deberá estar configurado correctamente en lo que se refiere a políticas de firma XML. El método devuelve un valor lógico que indica si la URI es válida (verdadero) o no (falso). Como parámetros recibe:

- **uriAlgorithm:** Parámetro que representa la URI del algoritmo de firma.
- **policyID:** Parámetro que representa el identificador de la política de firma a usar para validar la URI. Debe coincidir con alguno de los identificadores de política de firma para firmas XML definidos en el archivo **integra.properties**.
- **properties:** Parámetro que representa el conjunto de propiedades definidas en el archivo **integra.properties**.

```
public static boolean isValidASN1SigningModeByPolicy(boolean includeContent,
String policyID, Properties properties)
```

Método que comprueba si el modo de firma para una firma CAdES (Baseline o no) está permitido según una política de firma concreta. Para poder llevar a cabo esta operación, el archivo **integra.properties** deberá estar configurado correctamente en lo que se refiere a políticas de firma ASN.1. El método devuelve un valor lógico que indica si el modo de firma es válido (verdadero) o no (falso). Como parámetros recibe:

- **includeContent:** Parámetro que indica si la firma contiene los datos firmados (verdadero) o no (falso).
- **policyID:** Parámetro que representa el identificador de la política de firma a usar para validar el modo de firma. Debe coincidir con alguno de los identificadores de política de firma para firmas ASN.1 definidos en el archivo **integra.properties**.
- **properties:** Parámetro que representa el conjunto de propiedades definidas en el archivo **integra.properties**.

```
public static boolean isValidXMLSigningModeByPolicy(String signingMode, String
policyID, Properties properties)
```

Método que comprueba si el modo de firma para una firma XAdES (Baseline o no) está permitido según una política de firma concreta. Para poder llevar a cabo esta operación, el archivo policy.properties deberá estar configurado correctamente en lo que se refiere a políticas de firma XML. El método devuelve un valor lógico que indica si el modo de firma es válido (verdadero) o no (falso). Como parámetros recibe:

- **signingMode:** Parámetro que representa el modo de firma de la firma XAdES que validar. Los valores posibles son:
  - **Detached:** Indica que la firma incluye una referencia a los datos firmados.
  - **Enveloped:** Indica que la firma incluye los datos firmados.
  - **Enveloping:** Indica que la frma posee una estructura XML que contiene los datos firmados en un nodo propio.
- **policyID:** Parámetro que representa el identificador de la política de firma a usar para validar el modo de firma. Debe coincidir con alguno de los identificadores de política de firma para firmas ASN.1 definidos en el archivo **integra.properties**.
- **properties:** Parámetro que representa el conjunto de propiedades definidas en el archivo **integra.properties**.

9.4.6 Paquete es.gob.afirma.integraFacade

Paquete que contiene todas las clases asociadas a la fachada de invocación. Esta fachada ofrece funcionalidades de generación y validación de firma electrónica en los formatos CAdES (Baseline o no), XAdES (Baseline o no), PAdES (Baseline o no), y ASiC-S Baseline, si bien las firmas XAdES (Baseline o no) y ASiC-S Baseline solo están disponibles en el tipo de integración 5 "Procesado de firmas XAdES (Baseline o no) y ASiC-S Baseline, además de CAdES (Baseline o no) y PAdES (Baseline o no) o". De este paquete se describirán aquellas clases más destacadas.

Clase IntegraFacade

Esta clase representa la fachada de generación, actualización y validación de firmas en los formatos CAdES (Baseline o no), XAdES (Baseline o no), PAdES (Baseline o no), y ASiC-S Baseline que permite Integr@, pero agrupando y facilitando el uso de todos los métodos asociados, definidos en las clases que forman parte del Paquete es.gob.afirma.signature. Está compuesta por los métodos:

```
public static byte[ ] generateSignature(byte[ ] dataToSign, PrivateKeyEntry
privateKey, boolean includeSignaturePolicy, boolean includeTimestamp) throws
SigningException
```

Método que permite generar una firma CAdES-BES, CAdES-EPES, CAdES-T, CAdES B-Level, CAdES T-Level, XAdES-BES, XAdES-EPES, XAdES-T, XAdES B-Level, XAdES T-Level, PAdES-BES, PAdES-EPES, PAdES B-Level, PAdES T-Level, ASiC- S conteniendo una firma CAdES Baseline (B-Level o T-Level), o ASiC-S conteniendo una firma XAdES Baseline (B-Level o T-Level). En el caso de CAdES y PAdES la firma generada será implícita, en el caso de XAdES, la firma generada será 'detached', en el caso de ASiC-S conteniendo una firma CAdES Baseline, ésta será explícita, y en el caso de ASiC-S conteniendo una firma XAdES Baseline, ésta será ''detached". Será necesario tener configurado el fichero **integra.properties** en lo que respecta a las propiedades para la validación de los firmantes, a las propiedades para la comunicación con TS@ (si la firma a generar debe tener sello de tiempo), y a las propiedades para indicar el algoritmo de firma a utilizar y el tipo de firma a generar. Si se ha indicado en el fichero **integra.properties** que la validación de los firmantes incluirá el estado de revocación será necesario tener configurado el fichero **integra.properties** con las propiedades asociadas a la validación de cerficados por OCSP. Si la firma a generar debe tener política de firma, será necesario haberlo configurado también para poder hacer una validación de la política de firma en base a alguna de las políticas de firma definidas en dicho fichero. Si la firma a generar debe tener sello de tiempo, será necesario tener configurado correctamente el fichero **tsaXXXXX.properties** para permitir la comunicación con TS@ y así obtener el sello de tiempo, y además, se validará el núcleo de dicho sello de tiempo obtenido de TS@. En caso de que se produzca algún error durante la generación de la firma, el método lanzará una excepción *SigningException*. Como parámetros recibe:

- **dataToSign:** Parámetro que representa los datos a firmar.
- **privateKey:** Parámetro que representa la clave privada a utilizar para generar la firma.
- **includeSignaturePolicy:** Parámetro que indica si la firma a generar debe incluir política de firma (verdadero) o no (falso).

- **includeTimestamp:** Parámetro que indica si la firma a generar debe incluir sello de tiempo (verdadero) o no (falso). Si la firma a generar es PAdES (Baseline o no) y se indica que debe incluir sello de tiempo, dicho sello de tiempo se incluirá en el núcleo de firma CAdES que contendrá la firma PAdES.

```
public static byte[ ] generateCoSignature(byte[ ] signature, byte[ ] signedData,
PrivateKeyEntry privateKey, boolean includeSignaturePolicy, boolean
includeTimestamp) throws SigningException
```

Método que genera una co-firma CAdES-BES, CAdES-EPES, CAdES-T, CAdES B-Level, CAdES T-Level, XAdES-BES, XAdES-EPES, XAdES-T, XAdES B-Level o XAdES T-Level. Será necesario tener configurado el fichero **integraFacade.properties** en lo que respecta a las propiedades para la validación de los firmantes, a las propiedades para la comunicación con TS@ (si la co-firma a generar debe tener sello de tiempo), y a las propiedades para indicar el algoritmo de firma a utilizar y el tipo de co-firma a generar. Si se ha indicado en el fichero **integraFacade.properties** que la validación de los firmantes incluirá el estado de revocación será necesario tener configurado el fichero **ocsp.properties** con las propiedades asociadas a la validación de cerficados por OCSP. Si la co-firma a generar debe tener política de firma, será necesario haber configurado previamente el fichero **policy.properties** para poder hacer una validación de la política de firma en base a alguna de las políticas de firma definidas en dicho fichero. Si la co-firma a generar debe tener sello de tiempo, será necesario tener configurado correctamente el fichero **tsaServiceInvoker.properties** para permitir la comunicación con TS@ y así obtener el sello de tiempo, y además, se validará el núcleo de dicho sello de tiempo obtenido de TS@. En caso de que se produzca algún error durante la generación de la co-firma, el método lanzará una excepción *SigningException*. Como parámetros recibe:

- **signature:** Parámetro que representa la firma original que co-firmar.
- **signedData:** Parámetro que representa los datos originales que fueron firmados por la firma que co-firmar.
- **privateKey:** Parámetro que representa la clave privada a utilizar para generar la co-firma.
- **includeSignaturePolicy:** Parámetro que indica si la firma a generar debe incluir política de firma (verdadero) o no (falso).
- **includeTimestamp:** Parámetro que indica si la firma a generar debe incluir sello de tiempo (verdadero) o no (falso).

```
[2]public static byte[ ] generateCounterSignature(byte[ ] signature, PrivateKeyEntry
privateKey, boolean includeSignaturePolicy, boolean includeTimestamp) throws
SigningException
```

Método que genera una contra-firma CAdES-BES, CAdES-EPES, CAdES-T, CAdES B-Level, CAdES T-Level, XAdES-BES, XAdES-EPES, XAdES-T, XAdES B-Level o XAdES T-Level. Será necesario tener configurado el fichero **integra.properties** en lo que respecta a las propiedades para la validación de los firmantes, a las propiedades para la comunicación con TS@ (si la contra-firma a generar debe tener sello de tiempo), y a las propiedades para indicar el algoritmo de firma a utilizar y el tipo de contra-firma a generar. Si se ha indicado en el fichero **integra.properties** que la validación de los firmantes incluirá el estado de revocación será necesario tener configurado el fichero con las propiedades asociadas a la validación de cerficados por OCSP. Si la contra-firma a generar debe tener política de firma, será necesario haberlo configurado para poder hacer una validación de la política de firma en base a alguna de las políticas de firma definidas en dicho fichero. Si la contra-firma a generar debe tener sello de tiempo, será necesario tener configurado correctamente el fichero **tsaXXXXX.properties** para permitir la comunicación con TS@ y así obtener el sello de tiempo, y además, se validará el núcleo de dicho sello de tiempo obtenido de TS@. En caso de que se produzca algún error durante la generación de la contra-firma, el método lanzará una excepción *SigningException*. Como parámetros recibe:

- **signature:** Parámetro que representa la firma original que contra-firmar.
- **privateKey:** Parámetro que representa la clave privada a utilizar para generar la contra-firma.
- **includeSignaturePolicy:** Parámetro que indica si la firma a generar debe incluir política de firma (verdadero) o no (falso).
- **includeTimestamp:** Parámetro que indica si la firma a generar debe incluir sello de tiempo (verdadero) o no (falso).

```
public static byte[ ] upgradeSignature(byte[ ] signature, List<X509Certificate>
listSigners) throws SigningException
```

Método que actualiza una firma añadiendo un sello de tiempo a sus firmantes. El sello de tiempo será añadido sólo a aquellos firmantes que no posean ya un sello de tiempo. En el caso de que la firma a actualizar sea de tipo PDF el proceso de actualización consistirá en añadir un diccionario de sello de tiempo, independientemente de que ya tuviera uno previo. En el caso de que la firma sea ASiC-S Baseline conteniendo una firma CAdES Baseline o un documento XML firmado por firmas XAdES Baseline la actualización consistirá en añadir un sello de tiempo a dichos firmantes que poseieran uno previamente. Será necesario tener configurado el fichero **integra.properties** en lo que respecta a las

[2] En el caso de XAdES (Baseline o no), según su estándar, no se permite la generación de contra-firmas que no sean Detached. Es por ello que, en el caso de que se pretenda contra-firmar una firma no Detached (Enveloped o Enveloping), dicha firma será reconfigurada previamente a tipo Detached para, a continuación, llevar a cabo la contra-firma.

propiedades para la validación de los firmantes, y a las propiedades para la comunicación con TS@. Si se ha indicado en el fichero **integra.properties** que la validación de los firmantes incluirá el estado de revocación será necesario configurar las propiedades asociadas a la validación de cerficados por OCSP. Será necesario tener configurado correctamente el fichero **tsaXXXXX.properties** para permitir la comunicación con TS@ y así obtener el sello de tiempo. En caso de que se produzca algún error durante el proceso de actualización, el método lanzará una excepción *SigningException*.

Para firmas ASN.1, y XML, de aquellos firmantes que se hayan indicado actualizar y que no tengan un sello de tiempo previo, se validará el certificado firmante respecto a la fecha actual en base al valor indicado en la propiedad **CERTIFICATE_VALIDATION_LEVEL** del archivo **integra.properties**, salvo si se ha indicado que no se haga validación, en cuyo caso se llevará a cabo la validación simple. Además, se validará el núcleo del sello de tiempo que añadir a cada firmante. Para firmas PDF, si no contiene ningún diccionario de sello de tiempo previo, se validarán todos los certificados firmantes que contenga respecto a la fecha actual (si no posee sello de tiempo asociado), o respecto a la fecha del sello de tiempo asociado (en caso de tenerlo), en base al valor indicado en la propiedad **CERTIFICATE_VALIDATION_LEVEL** del archivo **integra.properties**, salvo si se ha indicado que no se haga validación, en cuyo caso se llevará a cabo la validación simple, y se validará el núcleo del sello de tiempo que se añadirá en el diccionario de sello de tiempo. Si la firma PDF contiene algún diccionario de sello de tiempo previo, se validarán todos los certificados firmantes que contenga respecto a la fecha del sello de tiempo más reciente contenido en los diccionarios de sello de tiempo, en base al valor indicado en la propiedad **CERTIFICATE_VALIDATION_LEVEL** del archivo **integra.properties**, salvo si se ha indicado que no se haga validación, en cuyo caso se llevará a cabo la validación simple, y se validará el núcleo del sello de tiempo que se añadirá en el diccionario de sello de tiempo.

Como parámetros recibe:

- **signature:** Parámetro que representa la firma que actualizar.
- **listSigners:** Parámetro que representa la lista de firmantes que actualizar. Si la lista es nula o vacía se actualizarán todos los firmantes. Este parámetro es sólo aplicable a firmas ASN.1, XML o ASiC-S Baseline.

```
public static Object verifySignature(byte[ ] signature, byte[ ] signedData)
```

Método que valida una firma. Devuelve un objeto que contiene toda la información asociada al proceso de validación. Si la firma a validar es de tipo PDF devolverá un objeto de tipo Clase PDFValidationResult, mientras que en otro caso el objeto a devolver será de tipo Clase ValidationResult. En el caso de que algún firmante tenga asociado política de firma será necesario haber configurado previamente el fichero **integra.properties** para poder hacer una validación de la política de firma en base a alguna de las políticas de firma definidas en dicho fichero. Por otro lado, si en el fichero **integra.properties** el nivel de validación para los certificados está definido como Completo, se llevará a cabo la validación del estado de revocación del certificado firmante vía OCSP en base a los parámetros configurados.

En el caso de firmas ASN.1 y XML, se validarán todos los certificados firmantes respecto a la fecha actual (si carecen de sello de tiempo), o respecto a la fecha del sello de tiempo asociado (si poseen

sello de tiempo), en base al valor indicado en la propiedad **CERTIFICATE_VALIDATION_LEVEL** del archivo **integra.properties**, y el núcleo del sello de tiempo de aquellos firmantes que lo incluyan.

En el caso de firmas PDF, si el documento PDF incluye diccionarios de sello de tiempo, se validarán todos los certificados firmantes respecto a la fecha del diccionario de sello de tiempo más reciente en base al valor indicado en la propiedad **CERTIFICATE_VALIDATION_LEVEL** del archivo **integra.properties**, el núcleo de los sellos de tiempo de aquellos firmantes que posean sello de tiempo, y el núcleo del sello de tiempo de los sellos de tiempo incluidos en los diccionarios de sello de tiempo. Si el documento PDF no incluye diccionarios de sello de tiempo, se validarán todos los certificados firmantes respecto a la fecha actual (si carecen de sello de tiempo), o respecto a la fecha del sello de tiempo asociado (si poseen sello de tiempo), en base al valor indicado en la propiedad **CERTIFICATE_VALIDATION_LEVEL** del archivo **integra.properties**, y el núcleo del sello de tiempo de aquellos firmantes que lo incluyan.

Como parámetros recibe:

- **signature:** Parámetro que representa la firma a validar.
- **signedData:** Parámetro que representa los datos originales que han sido firmados por la firma a validar. Sólo será requerido en el caso de que la firma a validar sea ASN.1 y explícita.

```
public static byte[ ] generateSignaturePAdESRubric(byte[ ] dataToSign,
PrivateKeyEntry privateKey, boolean includeSignaturePolicy, boolean
includeTimestamp, byte[ ] image, String imagePage, int lowerLeftX, int lowerLeftY,
int upperRightX, int upperRightY) throws SigningException
```

Método que permite generar una firma PAdES (Baseline o no) con rúbrica. Será necesario tener configurado el fichero **integra.properties** en lo que respecta a las propiedades para la validación de los firmantes, a las propiedades para la comunicación con TS@ (si la firma a generar debe tener sello de tiempo), y a las propiedades para indicar el algoritmo de firma a utilizar y el tipo de firma a generar. Si se ha indicado en el fichero **integra.properties** que el certificado firmante debe ser validado mediante un servicio OCSP será necesario configurar el fichero **integra.properties**, para permitir la comunicación con el servidor OCSP. En caso de que se produzca algún error durante la generación de la firma, el método lanzará una excepción *SigningException*. Como parámetros recibe:

- **dataToSign:** Parámetro que representa los datos a firmar.
- **privateKey:** Parámetro que representa la clave privada a utilizar para generar la firma CAdES que se incluirá en el diccionario de firma.
- **includeSignaturePolicy:** Parámetro que indica si la firma CAdES que se incluirá en el diccionario de firma contendrá política de firma o no.
- **includeTimestamp:** Parámetro que indica si la firma CAdES que se incluirá en el diccionario de firma contendrá sello de tiempo o no.

- **image:** Parámetro que representa la imagen a insertar en el documento PDF como rúbrica.

- **imagePage:** Parámetro que representa el número de página en el que se quiere insertar la rúbrica. Si es "-1", la rúbrica se insertará en la última página del documento.

- **lowerLeftX:** Parámetro que representa la coordenada horizontal inferior izquierda de la posición donde será insertada la imagen como rúbrica en la página correspondiente.

- **lowerLeftY:** Parámetro que representa la coordenada vertical inferior izquierda de la posición donde será insertada la imagen como rúbrica en la página correspondiente.

- **upperRightX:** Parámetro que representa la coordenada horizontal superior derecha de la posición donde será insertada la imagen como rúbrica en la página correspondiente.

- **upperRightY:** Parámetro que representa la coordenada vertical superior derecha de la posición donde será insertada la imagen como rúbrica en la página correspondiente.

```
public static byte[ ] generateSignaturePAdESRubric(byte[ ] dataToSign,
PrivateKeyEntry privateKey, boolean includeSignaturePolicy, boolean
includeTimestamp, byte[ ] image, String imagePage, int lowerLeftX, int lowerLeftY,
int upperRightX, int upperRightY) throws SigningException
```

Método que permite generar multi-firmas PAdES (Baseline o no) con rúbrica. Será necesario tener configurado el fichero **integra.properties** en lo que respecta a las propiedades para la validación de los firmantes, a las propiedades para la comunicación con TS@ (si la firma a generar debe tener sello de tiempo), y a las propiedades para indicar el algoritmo de firma a utilizar y el tipo de firma a generar. Si se ha indicado en el fichero **integra.properties** que el certificado firmante debe ser validado mediante un servicio OCSP será necesario configurar el fichero **integra.properties**, para permitir la comunicación con el servidor OCSP. En caso de que se produzca algún error durante la generación de la firma, el método lanzará una excepción *SigningException*. Como parámetros recibe:

- **dataToSign:** Parámetro que representa los datos a firmar.

- **privateKey:** Parámetro que representa la clave privada a utilizar para generar la firma CAdES que se incluirá en el diccionario de firma.

- **includeSignaturePolicy:** Parámetro que indica si la firma CAdES que se incluirá en el diccionario de firma contendrá política de firma o no.

- **includeTimestamp:** Parámetro que indica si la firma CAdES que se incluirá en el diccionario de firma contendrá sello de tiempo o no.

- **image:** Parámetro que representa la imagen a insertar en el documento PDF como rúbrica.

- **imagePage:** Parámetro que representa el número de página en el que se quiere insertar la rúbrica. Si es "-1", la rúbrica se insertará en la última página del documento.

- **lowerLeftX:** Parámetro que representa la coordenada horizontal inferior izquierda de la posición donde será insertada la imagen como rúbrica en la página correspondiente.

- **lowerLeftY:** Parámetro que representa la coordenada vertical inferior izquierda de la posición donde será insertada la imagen como rúbrica en la página correspondiente.

- **upperRightX:** Parámetro que representa la coordenada horizontal superior derecha de la posición donde será insertada la imagen como rúbrica en la página correspondiente.

- **upperRightY:** Parámetro que representa la coordenada vertical superior derecha de la posición donde será insertada la imagen como rúbrica en la página correspondiente.

9.4.7 Paquete es.gob.afirma.utils

Este paquete contiene clases que representan utilidades, así como interfaces con constantes. De este paquete se describirán aquellas clases más destacadas. Teniendo en cuenta que para este tipo de integración no funcionarán todas las clases y métodos del paquete es.gob.afirma.utils al no necesitarse inluir ciertas librerías de terceros innecesarias para este tipo de integración.

Clase Base64Coder

Esta clase define métodos que permiten la codificación y decodificación de datos en Base 64. Está compuesta por los métodos:

```
public static byte[ ] encodeBase64(byte[ ] data) throws TransformersException
```

Método que permite codificar en Base 64 una colección de bytes. Devuelve una colección de bytes codificados en Base 64. En caso de que se produzca algún error durante el proceso, el método lanzará una excepción *TransformersException*. Como parámetros recibe:

- **data:** Parámetro que representa la colección de bytes a codificar en Base 64.

```
public static byte[ ] encodeBase64(byte[ ] data, int offset, int len) throws
TransformersException
```

Método que permite codificar en Base 64 una colección de bytes, indicando la posición inicial y final de los bytes a codificar. Devuelve una colección de bytes codificados en Base 64. En caso de que se produzca algún error durante el proceso, el método lanzará una excepción *TransformersException*. Como parámetros recibe:

- **data:** Parámetro que representa la colección de bytes a codificar en Base 64.

- **offset:** Parámetro que representa la posición dentro de la colección de bytes para iniciar la codificación en Base 64.

- **len:** Parámetro que representa la posición dentro de la colección de bytes para finalizar la codificación en Base 64.

```
public static byte[ ] decodeBase64(byte[ ] data) throws TransformersException
```

Método que permite decodificar una colección de bytes codificados en Base 64. Devuelve una colección de bytes decodificados en Base 64. En caso de que se produzca algún error durante el proceso, el método lanzará una excepción *TransformersException*. Como parámetros recibe:

- **data:** Parámetro que representa la colección de bytes codificados en Base 64 a decodificar.

```
public static byte[ ] decodeBase64(byte[ ] data, int offset, int len) throws
TransformersException
```

Método que permite decodificar una colección de bytes codificados en Base 64, indicando la posición inicial y final de los bytes a decodificar. Devuelve una colección de bytes decodificada en Base 64. En caso de que se produzca algún error durante el proceso, el método lanzará una excepción *TransformersException*. Como parámetros recibe:

- **data:** Parámetro que representa la colección de bytes codificados en Base 64.

- **offset:** Parámetro que representa la posición dentro de la colección de bytes para iniciar la decodificación en Base 64.

- **len:** Parámetro que representa la posición dentro de la colección de bytes para finalizar la decodificación en Base 64.

```
public static boolean isBase64Encoded(byte[ ] data) throws TransformersException
```

Método que indica si una colección de bytes se encuentran codificados en Base 64. Devuelve un valor lógico que indica si la colección de bytes está codificada en Base 64 (verdadero) o no (falso). En caso de que se produzca algún error durante el proceso, el método lanzará una excepción *TransformersException*. Como parámetros recibe:

- **data:** Parámetro que representa la colección de bytes a comprobar.

```
public static String encodeBase64(String data) throws TransformersException
```

Método que permite codificar en Base 64 una cadena de texto. Devuelve una cadena de texto codificada en Base 64. En caso de que se produzca algún error durante el proceso, el método lanzará una excepción *TransformersException.* Como parámetros recibe:

- **data:** Parámetro que representa la cadena de texto a codificar en Base 64.

```
public static String decodeBase64(String data) throws TransformersException
```

Método que permite decodificar una cadena de texto codificada en Base 64. Devuelve una cadena de texto decodificada en Base 64. En caso de que se produzca algún error durante el proceso, el método lanzará una excepción *TransformersException.* Como parámetros recibe:

- **data:** Parámetro que representa la cadena de texto codificada en Base 64.

Clase CryptoUtil

Esta clase define métodos asociados a operaciones criptográficas de cálculo de resumen de datos o con funciones de hash. Está compuesta por los métodos:

```
public static byte[ ] digest(String algorithm, byte[ ] data) throws
SigningException
```

Método que permite calcular el resumen de un conjunto de datos a partir de un determinado algoritmo de hash. Devuelve una colección de bytes que se corresponde con el resumen calculado. En caso de que no sea posible llevar a cabo la operación se devolverá un valor nulo. Si la implementación asociada al algoritmo de resumen indicado como parámetro no existe para el proveedor criptográfico usado, el método lanzará una excepción *SigningException*. Como parámetros recibe:

- **algorithm:** Parámetro que representa el nombre del algoritmo de hash.
- **data:** Parámetro que representa la colección de datos que procesar.

```
public static AlgorithmIdentifier getAlgorithmIdentifierByName(String
hashAlgorithm)
```

Método que obtiene el OID de un algoritmo de hash a partir de su nombre. En caso de que no sea posible obtener el OID se devolverá un valor nulo. Como parámetros recibe:

- **hashAlgorithm:** Parámetro que representa el nombre del algoritmo de hash.

```
public static String getDigestAlgorithmName(final String pseudoName)
```

Método que obtiene el nombre de un algoritmo de hash a partir de su alias. Por ejemplo, si el alias del algoritmo de hash fuera “sHa1” el método devolvería como nombre del algoritmo “SHA-1”. En

caso de que no sea posible obtener el nombre del algoritmo se devolverá un valor nulo. Como parámetros recibe:

- **pseudoName:** Parámetro que representa el alias del algoritmo de hash.

Clase GenericUtils

Esta clase contiene utilidades de uso genérico. Está compuesta por los métodos:

```
public static boolean assertStringValue(String value)
```

Método que comprueba si una cadena de texto no es vacía ni nula. Devuelve un valor lógico que indica si la cadena de texto no es vacía ni nula (verdadero) o por el contrario es vacía o nula (falso). Como parámetros recibe:

- **value:** Parámetro que representa la cadena de texto a comprobar.

```
public static boolean assertArrayValid(byte[ ] data)
```

Método que comprueba si una colección de bytes no es vacía ni nula. Devuelve un valor lógico que indica si la colección de bytes no es vacía ni nula (verdadero) o por el contrario es vacía o nula (falso). Como parámetros recibe:

- **data:** Parámetro que representa la colección de bytes a comprobar.

```
public static String getValueFromMapsTree(String path, Map<String, Object>
treeValues)
```

Método que obtiene un valor concreto de un árbol de mapas a partir de una ruta indicada. Devuelve una cadena de texto que se corresponde con el valor buscado. Como parámetros recibe:

- **path:** Parámetro que representa la ruta en el árbol de mapas, utilizando como separador el carácter '/'.

- **treeValues:** Parámetro que representa el árbol de mapas que procesar.

```
public static byte[ ] getDataFromInputStream(final InputStream input) throws
IOException
```

Método que obtiene una colección de bytes a partir de una secuencia de datos. Devuelve la colección de bytes leídos de la secuencia de datos de entrada. En caso de que se produjese algún error durante el proceso, el método lanzará una excepción *IOException*. Como parámetros recibe:

- **input:** Parámetro que representa la secuencia de datos de entrada que procesar.

```
public static boolean checkNullValues(Object... values)
```

Método que comprueba si ninguno de los valores de un conjunto es nulo. Devuelve un valor lógico que indica si ningún valor es nulo (verdadero) o bien alguno es nulo (falso). Como parámetros recibe:

- **values:** Parámetro que representa el conjunto de valores a procesar.

```
public static void printResult(byte[ ] result, Logger logger)
```

Método que codifica en Base 64 y escribe en el log un conjunto de datos. Como parámetros recibe:

- **result:** Parámetro que representa el conjunto de datos que codificar en Base 64 y escribir en el log.
- **logger:** Parámetro que representa el elemento que gestiona el log.

Clase UtilsCertificate

Esta clase contiene define utilidades asociadas a la gestión de certificados. Está compuesta por los métodos:

```
public static String canonicalizeX500Principal(String x500PrincipalName)
```

Método que canonicaliza un elemento X.500 Principal de un certificado. Devuelve una cadena de texto que se corresponde con el elemento canonicalizado. Como parámetros recibe:

- **x500PrincipalName:** Parámetro que representa un elemento X.500 Principal de un certificado.

```
public static X509Certificate generateCertificate(byte[ ] certificateBytes) throws
CertificateException
```

Método que genera un certificado a partir de una colección de bytes. Devuelve un objeto X.509 que representa el certificado. En caso de que no sea posible obtener el certificado a partir de la colección de bytes, el método lanzará una excepción *CertificateException*. Como parámetros recibe:

- **certificateBytes:** Parámetro que representa el certificado.

```
public static boolean equals(X509Certificate cert1, X509Certificate cert2)
```

Método que compara la clave pública, el emisor y el número de serie de dos certificados. Devuelve un valor lógico que indica si ambos certificados son iguales (verdadero) o no (falso). Como parámetros recibe:

- **cert1:** Parámetro que representa el primer certificado a comparar.
- **cert2:** Parámetro que representa el segundo certificado a comparar.

Clase UtilsFileSystem

Esta clase contiene utilidades de uso genérico. Está compuesta por los métodos:

```
public static synchronized String readFileBase64Encoded(String path, boolean
isRelativePath)
```

Método que lee un archivo y lo codifica en Base 64. Devuelve una cadena de texto codificada en Base 64 que representa el contenido del archivo. Como parámetros recibe:

- **path:** Parámetro que representa la ruta al fichero. Puede ser una ruta completa o relativa.
- **isRelativePath:** Parámetro que indica si la ruta al fichero es relativa (verdadero) o completa (falso).

```
public static synchronized byte[ ] getArrayByteFileBase64Encoded(String path,
boolean isRelativePath)
```

Método que lee un archivo y lo codifica en Base 64. Devuelve una colección de bytes codificada en Base 64 que representa el contenido del archivo. Como parámetros recibe:

- **path:** Parámetro que representa la ruta al fichero. Puede ser una ruta completa o relativa.
- **isRelativePath:** Parámetro que indica si la ruta al fichero es relativa (verdadero) o completa (falso).

```
public static synchronized byte[ ] readFile(String path, boolean isRelativePath)
```

Método que lee un archivo. Devuelve una colección de bytes que representa el contenido del archivo. Como parámetros recibe:

- **path:** Parámetro que representa la ruta al fichero. Puede ser una ruta completa o relativa.

- **isRelativePath:** Parámetro que indica si la ruta al fichero es relativa (verdadero) o completa (falso).

```
public static void writeFile(byte[ ] data, String filename) throws IOException
```

Método que genera un archivo a partir de un conjunto de datos. En caso de producirse algún error durante el proceso, el método lanzará una excepción *IOException*. Como parámetros recibe:

- **data:** Parámetro que representa el conjunto de datos que se corresponden con el archivo.
- **filename:** Parámetro que representa la ruta completa al archivo a generar.

Clase UtilsKeystore

Esta clase contiene utilidades que permiten el manejo de almacenes de claves. Está compuesta por los métodos:

```
public static KeyStore loadKeystore(String path, String password, String type)
throws KeyStoreException, NoSuchAlgorithmException, CertificateException,
IOException
```

Método que lee un almacén de claves. En caso de que el tipo de almacén de claves a instanciar no esté reconocido, el método lanzará una excepción *KeyStoreException*. En caso de que el algoritmo usado para comprobar la integración del almacén de claves no esté reconocido, el método lanzará una excepción *NoSuchAlgorithmException*. En caso de que alguno de los certificados del almacén de claves no pueda ser cargado, el método lanzará una excepción *CertificateException*. En caso de que se produzca algún error de lectura del almacén de claves, el método lanzará una excepción *IOException*. Devuelve un objeto java que representa el almacén de claves. Como parámetros recibe:

- **path:** Parámetro que representa la ruta completa al almacén de claves.
- **password:** Parámetro que representa la contraseña para acceder al almacén de claves.
- **type:** Parámetro que representa el tipo del almacén de claves.

```
public static byte[ ] getCertificateEntry(byte[ ] keystore, String
keystoreDecodedPass, String alias, String keystoreType) throws KeyStoreException,
NoSuchAlgorithmException, CertificateException, IOException
```

Método que obtiene un certificado almacenado en un almacén de claves. Devuelve una colección de bytes que se corresponden con el objeto X.509 del certificado. En caso de que el tipo de almacén de claves a instanciar no esté reconocido, el método lanzará una excepción *KeyStoreException*. En caso de que el algoritmo usado para comprobar la integración del almacén de claves no esté reconocido, el método lanzará una excepción *NoSuchAlgorithmException*. En caso de que alguno de los

certificados del almacén de claves no pueda ser cargado, el método lanzará una excepción *CertificateException*. En caso de que se produzca algún error de lectura del almacén de claves, el método lanzará una excepción *IOException.* Como parámetros recibe:

- **keystore:** Parámetro que representa la colección de bytes que se corresponden con el almacén de claves.
- **keystoreDecodedPass:** Parámetro que representa la contraseña para acceder al almacén de claves.
- **alias:** Parámetro que representa el alias del certificado a obtener.
- **keystoreType:** Parámetro que representa el tipo del almacén de claves.

```
public static PrivateKey getPrivateKeyEntry(byte[ ] keystore, String
keystoreDecodedPass, String alias, String keystoreType, String
privateKeyDecodedPass) throws KeyStoreException, NoSuchAlgorithmException,
CertificateException, IOException, UnrecoverableKeyException
```

Método que obtiene una clave privada almacenada en un almacén de claves. Devuelve un objeto java que representa la clave privada. En caso de que el tipo de almacén de claves a instanciar no esté reconocido, el método lanzará una excepción *KeyStoreException*. En caso de que el algoritmo usado para comprobar la integración del almacén de claves no esté reconocido, el método lanzará una excepción *NoSuchAlgorithmException*. En caso de que alguno de los certificados del almacén de claves no pueda ser cargado, el método lanzará una excepción *CertificateException*. En caso de que se produzca algún error de lectura del almacén de claves, el método lanzará una excepción *IOException*. En caso de que la clave privada no pueda ser recuperada, el método lanzará una excepción *UnrecoverableKeyException*. Como parámetros recibe:

- **keystore:** Parámetro que representa la colección de bytes que se corresponden con el almacén de claves.
- **keystoreDecodedPass:** Parámetro que representa la contraseña para acceder al almacén de claves.
- **alias:** Parámetro que representa el alias de la clave privada a obtener.
- **keystoreType:** Parámetro que representa el tipo del almacén de claves.
- **privateKeyDecodedPass:** Parámetro que representa la contraseña para acceder a la clave privada.

```
public static List<X509Certificate> getListCertificates(byte[ ] keystore, String
keystoreDecodedPass, String keystoreType) throws KeyStoreException,
NoSuchAlgorithmException, CertificateException, IOException
```

Método que obtiene una lista con todos los certificados almacenados en un almacén de claves. En caso de que el tipo de almacén de claves a instanciar no esté reconocido, el método lanzará una excepción *KeyStoreException*. En caso de que el algoritmo usado para comprobar la integración del almacén de claves no esté reconocido, el método lanzará una excepción *NoSuchAlgorithmException*. En caso de que alguno de los certificados del almacén de claves no pueda ser cargado, el método lanzará una excepción *CertificateException*. En caso de que se produzca algún error de lectura del almacén de claves, el método lanzará una excepción *IOException*. Como parámetros recibe:

- **keystore:** Parámetro que representa la colección de bytes que se corresponden con el almacén de claves.
- **keystoreDecodedPass:** Parámetro que representa la contraseña para acceder al almacén de claves.
- **keystoreType:** Parámetro que representa el tipo del almacén de claves.

Clase UtilsResources

Esta clase proporciona funcionalidades para controlar el cierre de recursos. Está compuesta por los métodos:

```
public static void safeCloseInputStream(InputStream is)
```

Método que gestiona el cierre de un flujo de entrada. Como parámetros recibe:

- **is:** Parámetro que representa el flujo de entrada.

```
public static void safeCloseOutputStream(OutputStream os)
```

Método que gestiona el cierre de un flujo de salida. Como parámetros recibe:

- **os:** Parámetro que representa el flujo de salida.

```
public static void safeCloseSocket(Socket socket)
```

Método que gestiona el cierre de un *socket*. Como parámetros recibe:

- **socket:** Parámetro que representa el *socket*.

```
public static void safeClosePDFStamper(PdfStamper stamper)
```

Método que gestiona el cierre de un objeto que permite modificar un documento PDF. Como parámetros recibe:

- **stamper:** Parámetro que permite modificar un documento PDF.

Clase UtilsSignature

Esta clase proporciona funcionalidades criptográficas relacionadas con firmas electrónicas. Está compuesta por los métodos:

```
public static void validateCertificate(X509Certificate certificate, Date
validationDate, boolean isUpgradeOperation) throws SigningException
```

Método que valida el periodo de validez y el estado de revocación de un certificado. Si el certificado no es válido o se ha producido algún error durante la validación, se lanzará una excepción *SigningException.* Como parámetros recibe:

- **certificate:** Parámetro que representa el certificado a validar.
- **validationDate:** Paramétro que indica la fecha de validación.
- **isUpgradeOperation:** Bandera que indica si la operación original es de actualización o no.

```
public static PDFSignatureDictionary obtainLatestSignatureFromPDF(PdfReader
reader)
```

Método que recupera el diccionario de firma con el número de revisión mayor en una firma de tipo PAdES. Los parámetros recibidos son:

- **reader:** Parámetro que representa el objeto Reader del documento PDF.

```
public static boolean isNotPAdESEnhancedPDF(PdfDictionary pdfDic)
```

Método que indica si un diccionario de firma hace referencia a una firma PDF/PAdES-Basic o a una firma PAdES-BES/PAdES-EPES. Los parámetros recibidos son:

- **pdfDic:** Parámetro que representa el diccionario de firma.

```
public static CMSSignedData getCMSSignature(PDFSignatureDictionary
signatureDictionary) throws SigningException
```

Método que recupera el elemento signedData contenido en un diccionario de firma de un documento PDF. Si se produce algún error se lanzará una excepción *SigningException.* Los parámetros recibidos son:

- **signatureDictionary:** Parámetro que representa el diccionario de firma.

```
public static boolean isImplicit(CMSSignedData cmsSignedData)
```

Método que comprueba si la firma incluye el documento original o no.

- **cmsSignedData:** Parámetro que representa el mensaje de tipo pkcs7-signature.

```
public static boolean equalsHash(PdfArray pdfArrayByteRange, MessageDigest
messageDigestSignature, byte[ ] pdfDocument, byte[ ] hashSignature)
```

Método que compara dos bytes de arrays y comprueba si el hash de cada uno de ellos son iguales o no. Como parámetros recibe:

- **pdfArrayByteRange:** Array de bytes que representa los datos originales sobre los que se calculará el hash.
- **messageDigestSignature:** Algoritmo de resumen a utilizar.
- **pdfDocument:** Conjunto de bytes que representa el primer hash a comparar.
- **hashSignature:** Conjunto de bytes que representa el segundo hash a comparar.

```
public static void checkSubFilterConditionsISO320001(PDFSignatureDictionary
dictionarySignature, CMSSignedData signedData)
```

Método que comprueba que se cumpla la condición especificada en la sección 12.8.3.3.1 de la ISO 32000-1:

adbe.pkcs7.detached: The original signed message digest over the document's byte range shall be incorporated as the normal PKCS#7 SignedData field. No data shall be encapsulated in the PKCS#7 SignedData field.

adbe.pkcs7.sha1: The SHA1 digest of the document's byte range shall be encapsulated in the PKCS#7 SignedData field with ContentInfo of type Data. The digest of that SignedData shall be incorporated as the normal PKCS#7 digest.

Como parámetros recibe:

- **dictionarySignature:** Parámetro que representa el diccionario de firma.
- **signedData:** Parámetro que representa los datos firmados.

```
public static void validatePAdESEnhancedMandatoryAttributes(PDFSignatureDictionary
signatureDictionary, CMSSignedData signedData) throws SigningException
```

Método que valida los atributos obligatorios de una firma *PAdES enhanced*. Si se produce algún error durante el proceso el método lanzará una excepción *SigningException*. Como parámetros recibe:

- **signatureDictionary:** Parámetro que representa el diccionario de firma.
- **signedData:** Parámetro que representa los datos firmados.

```
public static void validatePAdESOptionalAttributes(PDFSignatureDictionary
signatureDictionary, CMSSignedData signedData, boolean isEPES, boolean isBasic)
throws SigningException
```

Método que valida los atributos opcionales de una firma PAdES. Si se produce algún error durante el proceso el método lanzará una excepción *SigningException*. Como parámetros recibe:

- **signatureDictionary:** Parámetro que representa el diccionario de firma.
- **signedData:** Parámetro que representa los datos firmados.
- **isEPES:** Bandera que indica si la firma es de tipo PAdES-EPES o PAdES-BES.
- **isBasic:** Bandera que indica si la firma es de tipo PAdES-Basic o PAdES enhanced.

```
public static X509CertificateHolder getX509CertificateHolderBySignerId(Store
certificatesStore, SignerId signerId)
```

Método que obtiene la estructura de un certificado almacenado. Como parámetros recibe:

- **certificatesStore:** Parámetro que representa el certificado almacenado.
- **signerId:** Parámetro que representa el identificador del firmante usado para buscar el certificado.

```
public static void validatePDFSigner(CMSSignedData signedData, SignerInformation
signerInformation, PdfDictionary pdfSignatureDictionary, Date validationDate)
throws SigningException
```

Método que valida la firma contenida en un documento PDF. Si se produce algún error durante el proceso el método lanzará una excepción *SigningException*. Como parámetros recibe:

- **signedData:** Parámetro que representa los datos firmados.
- **signerInformation:** Parámetro que representa la información del firmante.

- **pdfSignatureDictionary:** Parámetro que representa el diccionario de firma PDF.
- **validationDate:** Parámetro que representa la fecha de validación.

```
public static List<XAdESSignerInfo> getXAdESListSigners(Document doc)
```

Método que obtiene una lista con la principal información relativa a los firmantes de una firma XAdES. Como parámetros recibe:

- **doc:** Parámetro que representa el documento XML.

```
public static List<CAdESSignerInfo> getCAdESListSigners(CMSSignedData signedData)
```

Método que obtiene una lista con la principal información relativa a los firmantes de una firma CAdES. Como parámetros recibe:

- **signedData:** Parámetro que representa los datos firmados.

```
public static void
validateXAdESSigner(org.apache.xml.security.signature.XMLSignature xmlSignature,
X509Certificate signingCertificate, TimeStampToken tst, Element xmlTst, String
signingMode, byte[ ] signedFile, String signedFileName) throws SigningException
```

Método que valida el firmante de una firma XAdES. Si se produce algún error durante el proceso el método lanzará una excepción *SigningException*. Como parámetros recibe:

- **xmlSignature:** Parámetro que representa la firma XML.
- **signingCertificate**: Parámetro que representa el certificado firmante.
- **tst**: Parámetro que representa el sello de tiempo RFC3161 asociado al firmante.
- **xmlTst**: Parámetro que representa el sello de tiempo XML asociado al firmante.
- **signingMode**: Parámetro que representa el modo en el que se ha realizado la firma XAdES (detached, enveloped o enveloping).
- **signedFile**: Parámetro que representa el documento firmado cuando éstos no están incluidos en el XML.
- **signedFileName**: Parámetro que representa el nombre del documento firmado cuando éstos no están incluidos en el XML.

```
public static X509Certificate getSigningCertificate(CMSSignedData signedData,
SignerInformation signerInformation) throws SigningException
```

Método que obtiene el certificado firmante de un firmante incluido dentro de una forma. Si se produce algún error durante el proceso el método lanzará una excepción *SigningException*. Como parámetros recibe:

- **signedData:** Parámetro que representa los datos firmados.
- **signerInformation**: Parámetro que representa la información relativa al firmante sobre el que se quiere obtener el certificado.

```
public static Document getDocumentFromXML(byte[ ] xmlDocument) throws
SigningException
```

Método que obtiene un objeto java como representación de un documento XML. Si se produce algún error durante el proceso el método lanzará una excepción *SigningException*. Como parámetros recibe:

- **xmlDocument:** Parámetro que representa el documento XML.

```
public static PdfReader obtainLatestRevision(PdfReader reader) throws
SigningException
```

Método que obtiene el diccionario de firma con el número de revisión más reciente. Si se produce algún error durante el proceso el método lanzará una excepción SigningException. Como parámetros recibe:

- **reader:** Parámetro que representa el *reader* del documento PDF.

```
public static void checkPDFCertificationLevel(Map<Integer, InputStream>
mapRevisions) throws SigningException
```

Método que comprueba si se ha añadido alguna firma al documento PDF tras haber sido certificado. Si se produce algún error durante el proceso el método lanzará una excepción SigningException. Como parámetros recibe:

- **mapRevisions:** Parámetro que representa el conjunto de revisiones del documento PDF. Cada revisión representa un diccionario de firma.

Clase UtilsTimestamp

Esta clase proporciona funcionalidades criptográficas relacionadas con sellos de tiempo. Está compuesta por los métodos:

```
public static TimeStampToken getTimestampFromRFC3161Service(byte[ ] dataToStamp,
String applicationID, String tsaCommunicationMode) throws SigningException
```

Método que obtiene un sello de tiempo ASN.1 del servicio RFC 3161 de TS@. Para poder obtener el sello de tiempo de TS@ será necesario que el fichero **tsaXXXXX.properties** esté configurado correctamente respecto a las propiedades asociadas a la comunicación con TS@. En caso de que se produzca un error durante el proceso y no sea posible obtener el sello de tiempo el método lanzará una excepción *SigningException*. Como parámetros recibe:

- **dataToStamp:** Parámetro que representa los datos a sellar.
- **applicationID:** Parámetro que representa el identificador de aplicación cliente para la comunicación con TS@.
- **tsaCommunicationMode:** Parámetro que representa el protocolo definido para la comunicación con TS@. Los valores permitidos son:
  - **RFC3161-TCP** → Comunicación TCP.
  - **RFC3161-HTTPS** → Comunicación HTTPS.
  - **RFC3161-SSL** → Comunicación SSL.

```
public static X509Certificate getSigningCertificate(TimeStampToken tst) throws
SigningException
```

Método que obtiene el certificado firmante de un sello de tiempo ASN.1. En caso de que se produzca un error durante el proceso o no sea posible obtener el certificado el método lanzará una excepción *SigningException*. Como parámetros recibe:

- **tst:** Parámetro que representa el sello de tiempo ASN.1.

```
public static void validateASN1Timestamp(TimeStampToken tst) throws
SigningException
```

Método que valida estructuralmente un sello de tiempo ASN.1 (no valida el certificado firmante). En caso de que se produzca un error durante el proceso o si el sello de tiempo no es válido, el método lanzará una excepción *SigningException*. Como parámetros recibe:

- **tst:** Parámetro que representa el sello de tiempo ASN.1 que validar.

```
public static TimeStampToken getTimeStampToken(SignerInformation
signerInformation) throws SigningException
```

Método que obtiene un sello de tiempo ASN.1 de la información asociada al firmante de una firma, si es que ésta contiene un sello de tiempo. Si no contuviese un sello de tiempo se devolvería un valor nulo. En caso de que el sello de tiempo se encuentre mal formado el método lanzará una excepción *SigningException*. Como parámetros recibe:

- **signerInformation:** Parámetro que representa la información asociada al firmante.

```
public static Date getGenTimeXMLTimestamp(Element xmlTimestamp) throws
SigningException
```

Método que obtiene la fecha de generación de un sello de tiempo XML. En caso de que la fecha de generación del sello de tiempo no tenga un formato UTC el método lanzará una excepción *SigningException*. Como parámetros recibe:

- **xmlTimestamp:** Parámetro que representa el sello de tiempo XML.

Clase UtilsSignatureOp

Esta clase contiene un gran conjunto de mètodos utiles para el procesado y generación de firmas. Únicamente citaremos sus métodos más destacados:

```
public static Date getExpirationDate(byte[ ] signature)
```

Método que permite calcular la fecha de expiración de una firma a partir de su array de bytes. La fecha de expiración de una firma viene determinada por la fecha de caducidad más próxima del conjunto de certificados firmantes y certificados firmantes de sellos de tiempo. Como parámetro recibe:

- **signature:** Parámetro que representa el array de bytes de la firma a procesar.

9.4.8 Paquete es.gob.afirma.hsm

Este paquete contiene clases que gestionan el manejo de dispositivos HSM. Para poder hacer uso de los métodos definidos en este paquete debe estar configurado correctamente el archivo **hsm.properties**. De este paquete se describirán aquellas clases más destacadas.

Clase HSMKeystore

Esta clase maneja todas las operaciones relacionadas con almacenes de claves HSM. Está compuesta por los métodos:

```
public static PrivateKey getPrivateKey(String alias) throws HSMException
```

Método que obtiene una clave privada almacenada en un HSM a partir de su alias. En caso de que se produzca algún error durante el proceso, el método lanzará una excepción *HSMException*. Como parámetros recibe:

- **alias:** Parámetro que representa el alias de la clave privada a obtener.

```
public static X509Certificate getCertificate(String alias) throws HSMException
```

Método que obtiene un certificado almacenado en un HSM a partir de su alias. En caso de que se produzca algún error durante el proceso, el método lanzará una excepción *HSMException*. Como parámetros recibe:

- **alias:** Parámetro que representa el alias del certificado a obtener.

```
public static PrivateKeyEntry getPrivateKeyEntry(String alias) throws HSMException
```

Método que obtiene una entrada asociada a una clave privada almacenado en un HSM a partir de su alias. En caso de que se produzca algún error durante el proceso, el método lanzará una excepción *HSMException*. Como parámetros recibe:

- **alias:** Parámetro que representa el alias de la entrada a obtener.

9.5 Tipo de integración 5. Procesado de firmas XAdES (Baseline o no) y ASiC-S Baseline, además de CAdES (Baseline o no) y PAdES (Baseline o no)

Para este tipo de integración se dispone del conjunto de paquetes disponibles para el tipo de integración anterior además de los expuestos a continuación.

9.5.1 Paquete es.gob.afirma.signature.xades

Este paquete contiene todas las clases asociadas a funcionalidades de generación, validación y actualización de firmas XAdES (Baseline o no). De este paquete se describirán aquellas clases más destacadas.

Clase XadesSigner

Esta clase gestiona todas las operaciones asociadas a firmas XAdES no Baseline. Implementa la interfaz *Signer* descrita en el punto 0. Está compuesta por los métodos:

```
public byte[ ] sign(byte[ ] data, String algorithm, String signatureFormat,
PrivateKeyEntry privateKey, Properties extraParams, boolean includeTimestamp,
String signatureForm, String signaturePolicyID) throws SigningException
```

Método definido en la Interfaz Signer.

```
public byte[ ] coSign(byte[ ] signature, byte[ ] document, String algorithm,
PrivateKeyEntry privateKey, Properties extraParams, boolean includeTimestamp,
String signatureForm, String signaturePolicyID) throws SigningException
```

Método definido en la Interfaz Signer.

```
public byte[ ] counterSign(byte[ ] signature, String algorithm, PrivateKeyEntry
privateKey, Properties extraParams, boolean includeTimestamp, String
signatureForm, String signaturePolicyID) throws SigningException
```

Método definido en la Interfaz Signer.

```
public ValidationResult verifySignature(byte[ ] xmlDocument)
```

Método que valida el conjunto de firmantes de las firmas XAdES no Baseline contenidas dentro de un documento XML firmado. Devuelve un objeto que contiene toda la información asociada al proceso de validación. En el caso de que algún firmante incluya política de firma será necesario haber configurado previamente el fichero **integra.properties** para poder hacer una validación de la política de firma en base a alguna de las políticas de firma definidas en dicho fichero. Por otro lado, si en el fichero **integra.properties** el nivel de validación para los certificados está definido como *Completo*, se llevará a cabo la validación del estado de revocación del certificado firmante vía OCSP en base a los parámetros configurados.

Se validarán todos los certificados firmantes respecto a la fecha actual (si carecen de sello de tiempo), o respecto a la fecha del sello de tiempo asociado (si poseen sello de tiempo), en base al valor indicado en la propiedad **CERTIFICATE_VALIDATION_LEVEL** del archivo **integra.properties**, y el núcleo del sello de tiempo de aquellos firmantes que lo incluyan.

Como parámetros recibe:

- **xmlDocument:** Parámetro que representa el documento XML que contiene las firmas XAdES no Baseline a validar.

```
public byte[ ] upgrade(byte[ ] signature, List<X509Certificate> listCertificates)
throws SigningException
```

Método definido en la Interfaz Signer.

```
public OriginalSignedData getSignedData(byte[] signature) throws SigningException
```

Método definido en la Interfaz Signer. Lanza directamente la excepción SigningException, ya que este tipo de firma no es soportado por este método.

Clase XAdESBaselineSigner

Esta clase gestiona todas las operaciones asociadas a firmas XAdES Baseline. Implementa la interfaz *Signer* descrita en el punto 0. Está compuesta por los métodos:

```
public byte[ ] sign(byte[ ] data, String algorithm, String signatureFormat,
PrivateKeyEntry privateKey, Properties extraParams, boolean includeTimestamp,
String signatureForm, String signaturePolicyID) throws SigningException
```

Método definido en la Interfaz Signer.

```
public byte[ ] coSign(byte[ ] signature, byte[ ] document, String algorithm,
PrivateKeyEntry privateKey, Properties extraParams, boolean includeTimestamp,
String signatureForm, String signaturePolicyID) throws SigningException
```

Método definido en la Interfaz Signer.

```
public byte[ ] counterSign(byte[ ] signature, String algorithm, PrivateKeyEntry
privateKey, Properties extraParams, boolean includeTimestamp, String
signatureForm, String signaturePolicyID) throws SigningException
```

Método definido en la Interfaz Signer.

```
public ValidationResult verifySignature(byte[ ] xmlDocument)
```

Método que valida el conjunto de firmantes de las firmas XAdES Baseline contenidas dentro de un documento XML firmado. Devuelve un objeto que contiene toda la información asociada al proceso de validación. En el caso de que algún firmante incluya política de firma será necesario haber configurado previamente el fichero **integra.properties** para poder hacer una validación de la política de firma en base a alguna de las políticas de firma definidas en dicho fichero. Por otro lado, si en el fichero **integra.properties** el nivel de validación para los certificados está definido como *Completo*, se llevará a cabo la validación del estado de revocación del certificado firmante vía OCSP en base a los parámetros configurados.

Se validarán todos los certificados firmantes respecto a la fecha actual (si carecen de sello de tiempo), o respecto a la fecha del sello de tiempo asociado (si poseen sello de tiempo), en base al valor indicado en la propiedad **CERTIFICATE_VALIDATION_LEVEL** del archivo **integra.properties**, y el núcleo del sello de tiempo de aquellos firmantes que lo incluyan.

Como parámetros recibe:

- **xmlDocument:** Parámetro que representa el documento XML que contiene las firmas XAdES Baseline a validar.

```
public byte[ ] upgrade(byte[ ] signature, List<X509Certificate> listCertificates)
throws SigningException
```

Método definido en la Interfaz Signer.

```
public OriginalSignedData getSignedData(byte[] signature) throws SigningException
```

Método definido en la Interfaz Signer. Lanza directamente la excepción SigningException, ya que este tipo de firma no es soportado por este método.

9.5.2 Paquete es.gob.afirma.signature.asic

Este paquete contiene todas las clases asociadas a funcionalidades de generación, validación y actualización de firmas ASiC-S Baseline. De este paquete se describirán aquellas clases más destacadas.

Clase ASiCSBaselineSigner

Esta clase gestiona todas las operaciones asociadas a firmas ASiC-S Baseline. Implementa la interfaz *Signer* descrita en el punto 0. Está compuesta por los métodos:

```
public byte[ ] sign(byte[ ] data, String algorithm, String signatureFormat,
PrivateKeyEntry privateKey, Properties extraParams, boolean includeTimestamp,
String signatureForm, String signaturePolicyID) throws SigningException
```

Método definido en la Interfaz Signer.

```
public byte[ ] coSign(byte[ ] signature, byte[ ] document, String algorithm,
PrivateKeyEntry privateKey, Properties extraParams, boolean includeTimestamp,
String signatureForm, String signaturePolicyID) throws SigningException
```

Método definido en la Interfaz Signer. Este método no está soportado para firmas ASiC-S Baseline.

```
public byte[ ] counterSign(byte[ ] signature, String algorithm, PrivateKeyEntry
privateKey, Properties extraParams, boolean includeTimestamp, String
signatureForm, String signaturePolicyID) throws SigningException
```

Método definido en la Interfaz Signer. Este método no está soportado para firmas ASiC-S Baseline.

```
public ValidationResult verifySignature(byte[ ] asicSSignature)
```

Método que valida la estructura, el contenido, y el conjunto de firmantes de las firmas XAdES Baseline contenidas dentro de un documento XML firmado (si la firma ASiC-S contiene un documento XML firmado), o bien, el conjunto de firmantes de la firma CAdES Baseline (si la firma ASiC-S contiene una firma CAdES Baseline). Devuelve un objeto que contiene toda la información asociada al proceso de validación. En el caso de que algún firmante incluya política de firma será necesario haber configurado previamente el fichero **integra.properties** para poder hacer una validación de la política de firma en base a alguna de las políticas de firma definidas en dicho fichero. Por otro lado, si en el fichero **integra.properties** el nivel de validación para los certificados está definido como *Completo*, se llevará a cabo la validación del estado de revocación del certificado firmante vía OCSP en base a los parámetros configurados.

Se validarán todos los certificados firmantes respecto a la fecha actual (si carecen de sello de tiempo), o respecto a la fecha del sello de tiempo asociado (si poseen sello de tiempo), en base al valor indicado en la propiedad **CERTIFICATE_VALIDATION_LEVEL** del archivo **integra.properties**, y el núcleo del sello de tiempo de aquellos firmantes que lo incluyan.

Como parámetros recibe:

- **asicSSignature:** Parámetro que representa la firma ASiC-S que validar.

```
public byte[ ] upgrade(byte[ ] signature, List<X509Certificate> listCertificates)
throws SigningException
```

Método definido en la Interfaz Signer.

```
public OriginalSignedData getSignedData(byte[] signature) throws SigningException
```

Método definido en la Interfaz Signer.

9.5.3 Paquete es.gob.afirma.utils

Este paquete contiene clases que representan utilidades, así como interfaces con constantes. De este paquete se describirán aquellas clases más destacadas.

Clase Base64Coder

Esta clase define métodos que permiten la codificación y decodificación de datos en Base 64. Está compuesta por los métodos:

```
public static byte[ ] encodeBase64(byte[ ] data) throws TransformersException
```

Método que permite codificar en Base 64 una colección de bytes. Devuelve una colección de bytes codificados en Base 64. En caso de que se produzca algún error durante el proceso, el método lanzará una excepción *TransformersException*. Como parámetros recibe:

- **data:** Parámetro que representa la colección de bytes a codificar en Base 64.

```
public static byte[ ] encodeBase64(byte[ ] data, int offset, int len) throws
TransformersException
```

Método que permite codificar en Base 64 una colección de bytes, indicando la posición inicial y final de los bytes a codificar. Devuelve una colección de bytes codificados en Base 64. En caso de que se produzca algún error durante el proceso, el método lanzará una excepción *TransformersException*. Como parámetros recibe:

- **data:** Parámetro que representa la colección de bytes a codificar en Base 64.
- **offset:** Parámetro que representa la posición dentro de la colección de bytes para iniciar la codificación en Base 64.
- **len:** Parámetro que representa la posición dentro de la colección de bytes para finalizar la codificación en Base 64.

```
public static byte[ ] decodeBase64(byte[ ] data) throws TransformersException
```

Método que permite decodificar una colección de bytes codificados en Base 64. Devuelve una colección de bytes decodificados en Base 64. En caso de que se produzca algún error durante el proceso, el método lanzará una excepción *TransformersException*. Como parámetros recibe:

- **data:** Parámetro que representa la colección de bytes codificados en Base 64 a decodificar.

```
public static byte[ ] decodeBase64(byte[ ] data, int offset, int len) throws
TransformersException
```

Método que permite decodificar una colección de bytes codificados en Base 64, indicando la posición inicial y final de los bytes a decodificar. Devuelve una colección de bytes decodificada en Base 64. En

caso de que se produzca algún error durante el proceso, el método lanzará una excepción *TransformersException*. Como parámetros recibe:

- **data:** Parámetro que representa la colección de bytes codificados en Base 64.
- **offset:** Parámetro que representa la posición dentro de la colección de bytes para iniciar la decodificación en Base 64.
- **len:** Parámetro que representa la posición dentro de la colección de bytes para finalizar la decodificación en Base 64.

```
public static boolean isBase64Encoded(byte[ ] data) throws TransformersException
```

Método que indica si una colección de bytes se encuentran codificados en Base 64. Devuelve un valor lógico que indica si la colección de bytes está codificada en Base 64 (verdadero) o no (falso). En caso de que se produzca algún error durante el proceso, el método lanzará una excepción *TransformersException.* Como parámetros recibe:

- **data:** Parámetro que representa la colección de bytes a comprobar.

```
public static String encodeBase64(String data) throws TransformersException
```

Método que permite codificar en Base 64 una cadena de texto. Devuelve una cadena de texto codificada en Base 64. En caso de que se produzca algún error durante el proceso, el método lanzará una excepción *TransformersException.* Como parámetros recibe:

- **data:** Parámetro que representa la cadena de texto a codificar en Base 64.

```
public static String decodeBase64(String data) throws TransformersException
```

Método que permite decodificar una cadena de texto codificada en Base 64. Devuelve una cadena de texto decodificada en Base 64. En caso de que se produzca algún error durante el proceso, el método lanzará una excepción *TransformersException.* Como parámetros recibe:

- **data:** Parámetro que representa la cadena de texto codificada en Base 64.

Clase CryptoUtil

Esta clase define métodos asociados a operaciones criptográficas de cálculo de resumen de datos o con funciones de hash. Está compuesta por los métodos:

```
public static byte[ ] digest(String algorithm, byte[ ] data) throws
SigningException
```

Método que permite calcular el resumen de un conjunto de datos a partir de un determinado algoritmo de hash. Devuelve una colección de bytes que se corresponde con el resumen calculado. En caso de que no sea posible llevar a cabo la operación se devolverá un valor nulo. Si la implementación asociada al algoritmo de resumen indicado como parámetro no existe para el proveedor criptográfico usado, el método lanzará una excepción *SigningException*. Como parámetros recibe:

- **algorithm:** Parámetro que representa el nombre del algoritmo de hash.
- **data:** Parámetro que representa la colección de datos que procesar.

```
public static String translateAlgorithmIdentifier(AlgorithmIdentifier
algorithmIdentifier)
```

Método que obtiene el nombre de un algoritmo de hash a partir de su OID. Si no es posible obtener el nombre del algoritmo se devolverá un valor nulo. Como parámetros recibe:

- **algorithmIdentifier:** Parámetro que representa el OID del algoritmo de hash.

```
public static AlgorithmIdentifier getAlgorithmIdentifierByName(String
hashAlgorithm)
```

Método que obtiene el OID de un algoritmo de hash a partir de su nombre. En caso de que no sea posible obtener el OID se devolverá un valor nulo. Como parámetros recibe:

- **hashAlgorithm:** Parámetro que representa el nombre del algoritmo de hash.

```
public static String translateXmlDigestAlgorithm(String digestAlg)
```

Método que obtiene el nombre de un algoritmo de hash a partir de su URI. En caso de que no sea posible obtener el nombre del algoritmo se devolverá un valor nulo. Como parámetros recibe:

- **digestAlg:** Parámetro que representa la URI del algoritmo de hash.

```
public static String getDigestAlgorithmName(final String pseudoName)
```

Método que obtiene el nombre de un algoritmo de hash a partir de su alias. Por ejemplo, si el alias del algoritmo de hash fuera “sHa1” el método devolvería como nombre del algoritmo “SHA-1”. En caso de que no sea posible obtener el nombre del algoritmo se devolverá un valor nulo. Como parámetros recibe:

- **pseudoName:** Parámetro que representa el alias del algoritmo de hash.

```
public static MessageImprint generateMessageImprintFromXMLAlgorithm(String
hashAlgXML, byte[ ] data)
```

Método que calcula el "MessageImprint" de un conjunto de datos aplicando un determinado algoritmo de resumen. Como parámetros recibe:

- **hashAlgXML:** Parámetro que representa la URI del algoritmo de resumen.
- **data:** Parámetro que representa el conjunto de datos.

Clase EVisorUtil

Esta clase define utilidades relacionadas con los WS de eVisor. Está compuesta por los métodos:

```
public static Map<String, Object> newBarcodeMap(String message, String type,
Map<String, String> configParams)
```

Método que obtiene un mapa con todos los valores del nodo *< srsm:Barcode>*. Devuelve un mapa con tres elementos, el primero tiene como clave la etiqueta *srsm:Message* y como valor el mensaje del código de barras; el segundo tiene como clave la etiqueta *srsm:Type* y como valor el tipo del código de barras; el tercero tiene como clave la etiqueta con formato XPath *srsm:Configuration/srsm:Parameter* y como valor un mapa con los parámetros de configuración del código de barras. Como parámetros recibe:

- **message:** Parámetro que representa el mensaje del código de barras.
- **type:** Parámetro que representa el tipo del código de barras.
- **configParams:** Parámetros de configuración del código de barras.

```
public static Map<?, ?>[ ] newParameterMap(Map<String, String> configParams)
```

Método que obtiene una colección de mapas a partir de los parámetros de configuración de un código de barras. Devuelve una colección de mapas donde cada mapa se compone de dos registros, uno para la etiqueta *srsm:ParameterId*, y otro para la etiqueta *srsm:ParameterValue*. En caso de que no sea posible procesar los parámetros de entrada se devolverá un valor nulo. Como parámetros recibe:

- **configParams:** Parámetros de configuración del código de barras.

Clase GenericUtils

Esta clase contiene utilidades de uso genérico. Está compuesta por los métodos:

```
public static boolean assertStringValue(String value)
```

Método que comprueba si una cadena de texto no es vacía ni nula. Devuelve un valor lógico que indica si la cadena de texto no es vacía ni nula (verdadero) o por el contrario es vacía o nula (falso). Como parámetros recibe:

- **value:** Parámetro que representa la cadena de texto a comprobar.

```
public static boolean assertArrayValid(byte[ ] data)
```

Método que comprueba si una colección de bytes no es vacía ni nula. Devuelve un valor lógico que indica si la colección de bytes no es vacía ni nula (verdadero) o por el contrario es vacía o nula (falso). Como parámetros recibe:

- **data:** Parámetro que representa la colección de bytes a comprobar.

```
public static String getValueFromMapsTree(String path, Map<String, Object>
treeValues)
```

Método que obtiene un valor concreto de un árbol de mapas a partir de una ruta indicada. Devuelve una cadena de texto que se corresponde con el valor buscado. Como parámetros recibe:

- **path:** Parámetro que representa la ruta en el árbol de mapas, utilizando como separador el carácter '/'.
- **treeValues:** Parámetro que representa el árbol de mapas que procesar.

```
public static byte[ ] getDataFromInputStream(final InputStream input) throws
IOException
```

Método que obtiene una colección de bytes a partir de una secuencia de datos. Devuelve la colección de bytes leídos de la secuencia de datos de entrada. En caso de que se produjese algún error durante el proceso, el método lanzará una excepción *IOException*. Como parámetros recibe:

- **input:** Parámetro que representa la secuencia de datos de entrada que procesar.

```
public static boolean checkNullValues(Object... values)
```

Método que comprueba si ninguno de los valores de un conjunto es nulo. Devuelve un valor lógico que indica si ningún valor es nulo (verdadero) o bien alguno es nulo (falso). Como parámetros recibe:

- **values:** Parámetro que representa el conjunto de valores a procesar.

```
public static void printResult(byte[ ] result, Logger logger)
```

Método que codifica en Base 64 y escribe en el log un conjunto de datos. Como parámetros recibe:

- **result:** Parámetro que representa el conjunto de datos que codificar en Base 64 y escribir en el log.
- **logger:** Parámetro que representa el elemento que gestiona el log.

Clase UtilsCertificate

Esta clase contiene define utilidades asociadas a la gestión de certificados. Está compuesta por los métodos:

```
public static String canonicalizeX500Principal(String x500PrincipalName)
```

Método que canonicaliza un elemento X.500 Principal de un certificado. Devuelve una cadena de texto que se corresponde con el elemento canonicalizado. Como parámetros recibe:

- **x500PrincipalName:** Parámetro que representa un elemento X.500 Principal de un certificado.

```
public static X509Certificate generateCertificate(byte[ ] certificateBytes) throws
CertificateException
```

Método que genera un certificado a partir de una colección de bytes. Devuelve un objeto X.509 que representa el certificado. En caso de que no sea posible obtener el certificado a partir de la colección de bytes, el método lanzará una excepción *CertificateException*. Como parámetros recibe:

- **certificateBytes:** Parámetro que representa el certificado.

```
public static boolean equals(X509Certificate cert1, X509Certificate cert2)
```

Método que compara la clave pública, el emisor y el número de serie de dos certificados. Devuelve un valor lógico que indica si ambos certificados son iguales (verdadero) o no (falso). Como parámetros recibe:

- **cert1:** Parámetro que representa el primer certificado a comparar.
- **cert2:** Parámetro que representa el segundo certificado a comparar.

Clase UtilsFileSystem

Esta clase contiene utilidades de uso genérico. Está compuesta por los métodos:

```
public static synchronized String readFileBase64Encoded(String path, boolean
isRelativePath)
```

Método que lee un archivo y lo codifica en Base 64. Devuelve una cadena de texto codificada en Base 64 que representa el contenido del archivo. Como parámetros recibe:

- **path:** Parámetro que representa la ruta al fichero. Puede ser una ruta completa o relativa.
- **isRelativePath:** Parámetro que indica si la ruta al fichero es relativa (verdadero) o completa (falso).

```
public static synchronized byte[ ] getArrayByteFileBase64Encoded(String path,
boolean isRelativePath)
```

Método que lee un archivo y lo codifica en Base 64. Devuelve una colección de bytes codificada en Base 64 que representa el contenido del archivo. Como parámetros recibe:

- **path:** Parámetro que representa la ruta al fichero. Puede ser una ruta completa o relativa.
- **isRelativePath:** Parámetro que indica si la ruta al fichero es relativa (verdadero) o completa (falso).

```
public static synchronized byte[ ] readFile(String path, boolean isRelativePath)
```

Método que lee un archivo. Devuelve una colección de bytes que representa el contenido del archivo. Como parámetros recibe:

- **path:** Parámetro que representa la ruta al fichero. Puede ser una ruta completa o relativa.
- **isRelativePath:** Parámetro que indica si la ruta al fichero es relativa (verdadero) o completa (falso).

```
public static void writeFile(byte[ ] data, String filename) throws IOException
```

Método que genera un archivo a partir de un conjunto de datos. En caso de producirse algún error durante el proceso, el método lanzará una excepción *IOException*. Como parámetros recibe:

- **data:** Parámetro que representa el conjunto de datos que se corresponden con el archivo.
- **filename:** Parámetro que representa la ruta completa al archivo a generar.

Clase UtilsKeystore

Esta clase contiene utilidades que permiten el manejo de almacenes de claves. Está compuesta por los métodos:

```
public static KeyStore loadKeystore(String path, String password, String type)
throws KeyStoreException, NoSuchAlgorithmException, CertificateException,
IOException
```

Método que lee un almacén de claves. En caso de que el tipo de almacén de claves a instanciar no esté reconocido, el método lanzará una excepción *KeyStoreException*. En caso de que el algoritmo usado para comprobar la integración del almacén de claves no esté reconocido, el método lanzará una excepción *NoSuchAlgorithmException*. En caso de que alguno de los certificados del almacén de claves no pueda ser cargado, el método lanzará una excepción *CertificateException*. En caso de que se produzca algún error de lectura del almacén de claves, el método lanzará una excepción *IOException*. Devuelve un objeto java que representa el almacén de claves. Como parámetros recibe:

- **path:** Parámetro que representa la ruta completa al almacén de claves.
- **password:** Parámetro que representa la contraseña para acceder al almacén de claves.
- **type:** Parámetro que representa el tipo del almacén de claves.

```
public static byte[ ] getCertificateEntry(byte[ ] keystore, String
keystoreDecodedPass, String alias, String keystoreType) throws KeyStoreException,
NoSuchAlgorithmException, CertificateException, IOException
```

Método que obtiene un certificado almacenado en un almacén de claves. Devuelve una colección de bytes que se corresponden con el objeto X.509 del certificado. En caso de que el tipo de almacén de claves a instanciar no esté reconocido, el método lanzará una excepción *KeyStoreException*. En caso de que el algoritmo usado para comprobar la integración del almacén de claves no esté reconocido, el método lanzará una excepción *NoSuchAlgorithmException*. En caso de que alguno de los certificados del almacén de claves no pueda ser cargado, el método lanzará una excepción *CertificateException*. En caso de que se produzca algún error de lectura del almacén de claves, el método lanzará una excepción *IOException.* Como parámetros recibe:

- **keystore:** Parámetro que representa la colección de bytes que se corresponden con el almacén de claves.
- **keystoreDecodedPass:** Parámetro que representa la contraseña para acceder al almacén de claves.
- **alias:** Parámetro que representa el alias del certificado a obtener.

- **keystoreType:** Parámetro que representa el tipo del almacén de claves.

```
public static PrivateKey getPrivateKeyEntry(byte[ ] keystore, String
keystoreDecodedPass, String alias, String keystoreType, String
privateKeyDecodedPass) throws KeyStoreException, NoSuchAlgorithmException,
CertificateException, IOException, UnrecoverableKeyException
```

Método que obtiene una clave privada almacenada en un almacén de claves. Devuelve un objeto java que representa la clave privada. En caso de que el tipo de almacén de claves a instanciar no esté reconocido, el método lanzará una excepción *KeyStoreException*. En caso de que el algoritmo usado para comprobar la integración del almacén de claves no esté reconocido, el método lanzará una excepción *NoSuchAlgorithmException*. En caso de que alguno de los certificados del almacén de claves no pueda ser cargado, el método lanzará una excepción *CertificateException*. En caso de que se produzca algún error de lectura del almacén de claves, el método lanzará una excepción *IOException*. En caso de que la clave privada no pueda ser recuperada, el método lanzará una excepción *UnrecoverableKeyException*. Como parámetros recibe:

- **keystore:** Parámetro que representa la colección de bytes que se corresponden con el almacén de claves.
- **keystoreDecodedPass:** Parámetro que representa la contraseña para acceder al almacén de claves.
- **alias:** Parámetro que representa el alias de la clave privada a obtener.
- **keystoreType:** Parámetro que representa el tipo del almacén de claves.
- **privateKeyDecodedPass:** Parámetro que representa la contraseña para acceder a la clave privada.

```
public static List<X509Certificate> getListCertificates(byte[ ] keystore, String
keystoreDecodedPass, String keystoreType) throws KeyStoreException,
NoSuchAlgorithmException, CertificateException, IOException
```

Método que obtiene una lista con todos los certificados almacenados en un almacén de claves. En caso de que el tipo de almacén de claves a instanciar no esté reconocido, el método lanzará una excepción *KeyStoreException*. En caso de que el algoritmo usado para comprobar la integración del almacén de claves no esté reconocido, el método lanzará una excepción *NoSuchAlgorithmException*. En caso de que alguno de los certificados del almacén de claves no pueda ser cargado, el método lanzará una excepción *CertificateException*. En caso de que se produzca algún error de lectura del almacén de claves, el método lanzará una excepción *IOException*. Como parámetros recibe:

- **keystore:** Parámetro que representa la colección de bytes que se corresponden con el almacén de claves.
- **keystoreDecodedPass:** Parámetro que representa la contraseña para acceder al almacén de claves.

- **keystoreType:** Parámetro que representa el tipo del almacén de claves.

Clase UtilsResources

Esta clase proporciona funcionalidades para controlar el cierre de recursos. Está compuesta por los métodos:

```
public static void safeCloseInputStream(InputStream is)
```

Método que gestiona el cierre de un flujo de entrada. Como parámetros recibe:

- **is:** Parámetro que representa el flujo de entrada.

```
public static void safeCloseOutputStream(OutputStream os)
```

Método que gestiona el cierre de un flujo de salida. Como parámetros recibe:

- **os:** Parámetro que representa el flujo de salida.

```
public static void safeCloseSocket(Socket socket)
```

Método que gestiona el cierre de un *socket*. Como parámetros recibe:

- **socket:** Parámetro que representa el *socket*.

```
public static void safeClosePDFStamper(PdfStamper stamper)
```

Método que gestiona el cierre de un objeto que permite modificar un documento PDF. Como parámetros recibe:

- **stamper:** Parámetro que permite modificar un documento PDF.

Clase UtilsSignature

Esta clase proporciona funcionalidades criptográficas relacionadas con firmas electrónicas. Está compuesta por los métodos:

```
public static void validateCertificate(X509Certificate certificate, Date
validationDate, boolean isUpgradeOperation) throws SigningException
```

Método que valida el periodo de validez y el estado de revocación de un certificado. Si el certificado no es válido o se ha producido algún error durante la validación, se lanzará una excepción *SigningException.* Como parámetros recibe:

- **certificate:** Parámetro que representa el certificado a validar.
- **validationDate:** Paramétro que indica la fecha de validación.
- **isUpgradeOperation:** Bandera que indica si la operación original es de actualización o no.

```
public static PDFSignatureDictionary obtainLatestSignatureFromPDF(PdfReader
reader)
```

Método que recupera el diccionario de firma con el número de revisión mayor en una firma de tipo PAdES. Los parámetros recibidos son:

- **reader:** Parámetro que representa el objeto Reader del documento PDF.

```
public static boolean isNotPAdESEnhancedPDF(PdfDictionary pdfDic)
```

Método que indica si un diccionario de firma hace referencia a una firma PDF/PAdES-Basic o a una firma PAdES-BES/PAdES-EPES. Los parámetros recibidos son:

- **pdfDic:** Parámetro que representa el diccionario de firma.

```
public static CMSSignedData getCMSSignature(PDFSignatureDictionary
signatureDictionary) throws SigningException
```

Método que recupera el elemento signedData contenido en un diccionario de firma de un documento PDF. Si se produce algún error se lanzará una excepción *SigningException.* Los parámetros recibidos son:

- **signatureDictionary:** Parámetro que representa el diccionario de firma.

```
public static boolean isImplicit(CMSSignedData cmsSignedData)
```

Método que comprueba si la firma incluye el documento original o no.

- **cmsSignedData:** Parámetro que representa el mensaje de tipo pkcs7-signature.

```
public static boolean equalsHash(PdfArray pdfArrayByteRange, MessageDigest
messageDigestSignature, byte[ ] pdfDocument, byte[ ] hashSignature)
```

Método que compara dos bytes de arrays y comprueba si el hash de cada uno de ellos son iguales o no. Como parámetros recibe:

- **pdfArrayByteRange:** Array de bytes que representa los datos originales sobre los que se calculará el hash.
- **messageDigestSignature:** Algoritmo de resumen a utilizar.
- **pdfDocument:** Conjunto de bytes que representa el primer hash a comparar.
- **hashSignature:** Conjunto de bytes que representa el segundo hash a comparar.

```
public static void checkSubFilterConditionsISO320001(PDFSignatureDictionary
dictionarySignature, CMSSignedData signedData)
```

Método que comprueba que se cumpla la condición especificada en la sección 12.8.3.3.1 de la ISO 32000-1:

adbe.pkcs7.detached: The original signed message digest over the document's byte range shall be incorporated as the normal PKCS#7 SignedData field. No data shall be encapsulated in the PKCS#7 SignedData field.

adbe.pkcs7.sha1: The SHA1 digest of the document's byte range shall be encapsulated in the PKCS#7 SignedData field with ContentInfo of type Data. The digest of that SignedData shall be incorporated as the normal PKCS#7 digest.

Como parámetros recibe:

- **dictionarySignature:** Parámetro que representa el diccionario de firma.
- **signedData:** Parámetro que representa los datos firmados.

```
public static void validatePAdESEnhancedMandatoryAttributes(PDFSignatureDictionary
signatureDictionary, CMSSignedData signedData) throws SigningException
```

Método que valida los atributos obligatorios de una firma *PAdES enhanced*. Si se produce algún error durante el proceso el método lanzará una excepción *SigningException*. Como parámetros recibe:

- **signatureDictionary:** Parámetro que representa el diccionario de firma.
- **signedData:** Parámetro que representa los datos firmados.

```
public static void validatePAdESOptionalAttributes(PDFSignatureDictionary
signatureDictionary, CMSSignedData signedData, boolean isEPES, boolean isBasic)
throws SigningException
```

Método que valida los atributos opcionales de una firma PAdES. Si se produce algún error durante el proceso el método lanzará una excepción *SigningException*. Como parámetros recibe:

- **signatureDictionary:** Parámetro que representa el diccionario de firma.
- **signedData:** Parámetro que representa los datos firmados.
- **isEPES:** Bandera que indica si la firma es de tipo PAdES-EPES o PAdES-BES.
- **isBasic:** Bandera que indica si la firma es de tipo PAdES-Basic o PAdES enhanced.

```
public static X509CertificateHolder getX509CertificateHolderBySignerId(Store
certificatesStore, SignerId signerId)
```

Método que obtiene la estructura de un certificado almacenado. Como parámetros recibe:

- **certificatesStore:** Parámetro que representa el certificado almacenado.
- **signerId:** Parámetro que representa el identificador del firmante usado para buscar el certificado.

```
public static void validatePDFSigner(CMSSignedData signedData, SignerInformation
signerInformation, PdfDictionary pdfSignatureDictionary, Date validationDate)
throws SigningException
```

Método que valida la firma contenida en un documento PDF. Si se produce algún error durante el proceso el método lanzará una excepción *SigningException*. Como parámetros recibe:

- **signedData:** Parámetro que representa los datos firmados.
- **signerInformation:** Parámetro que representa la información del firmante.
- **pdfSignatureDictionary:** Parámetro que representa el diccionario de firma PDF.
- **validationDate:** Parámetro que representa la fecha de validación.

```
public static List<XAdESSignerInfo> getXAdESListSigners(Document doc)
```

Método que obtiene una lista con la principal información relativa a los firmantes de una firma XAdES. Como parámetros recibe:

- **doc:** Parámetro que representa el documento XML.

```
public static List<CAdESSignerInfo> getCAdESListSigners(CMSSignedData signedData)
```

Método que obtiene una lista con la principal información relativa a los firmantes de una firma CAdES. Como parámetros recibe:

- **signedData:** Parámetro que representa los datos firmados.

```
public static void
validateXAdESSigner(org.apache.xml.security.signature.XMLSignature xmlSignature,
X509Certificate signingCertificate, TimeStampToken tst, Element xmlTst, String
signingMode, byte[ ] signedFile, String signedFileName) throws SigningException
```

Método que valida el firmante de una firma XAdES. Si se produce algún error durante el proceso el método lanzará una excepción *SigningException*. Como parámetros recibe:

- **xmlSignature:** Parámetro que representa la firma XML.
- **signingCertificate**: Parámetro que representa el certificado firmante.
- **tst**: Parámetro que representa el sello de tiempo RFC3161 asociado al firmante.
- **xmlTst**: Parámetro que representa el sello de tiempo XML asociado al firmante.
- **signingMode**: Parámetro que representa el modo en el que se ha realizado la firma XAdES (detached, enveloped o enveloping).
- **signedFile**: Parámetro que representa el documento firmado cuando éstos no están incluidos en el XML.
- **signedFileName**: Parámetro que representa el nombre del documento firmado cuando éstos no están incluidos en el XML.

```
public static X509Certificate getSigningCertificate(CMSSignedData signedData,
SignerInformation signerInformation) throws SigningException
```

Método que obtiene el certificado firmante de un firmante incluido dentro de una forma. Si se produce algún error durante el proceso el método lanzará una excepción *SigningException*. Como parámetros recibe:

- **signedData:** Parámetro que representa los datos firmados.
- **signerInformation**: Parámetro que representa la información relativa al firmante sobre el que se quiere obtener el certificado.

```
public static Document getDocumentFromXML(byte[ ] xmlDocument) throws
SigningException
```

Método que obtiene un objeto java como representación de un documento XML. Si se produce algún error durante el proceso el método lanzará una excepción *SigningException*. Como parámetros recibe:

- **xmlDocument:** Parámetro que representa el documento XML.

```
public static PdfReader obtainLatestRevision(PdfReader reader) throws
SigningException
```

Método que obtiene el diccionario de firma con el número de revisión más reciente. Si se produce algún error durante el proceso el método lanzará una excepción SigningException. Como parámetros recibe:

- **reader:** Parámetro que representa el *reader* del documento PDF.

```
public static void checkPDFCertificationLevel(Map<Integer, InputStream>
mapRevisions) throws SigningException
```

Método que comprueba si se ha añadido alguna firma al documento PDF tras haber sido certificado. Si se produce algún error durante el proceso el método lanzará una excepción SigningException. Como parámetros recibe:

- **mapRevisions:** Parámetro que representa el conjunto de revisiones del documento PDF. Cada revisión representa un diccionario de firma.

Clase UtilsTimestamp

Esta clase proporciona funcionalidades criptográficas relacionadas con sellos de tiempo. Está compuesta por los métodos:

```
public static Object getTimestampFromDssService(byte[ ] dataToStamp, String
applicationID, String signatureType) throws SigningException
```

Método que obtiene un sello de tiempo de TS@. El sello de tiempo puede ser de tipo RFC 3161 (objeto java org.bouncycastle.tsp.TimeStampToken) o bien de tipo XML (objeto java org.w3c.dom.Element). Para poder obtener el sello de tiempo de TS@ será necesario que el fichero **tsaXXXXX.properties** esté configurado correctamente respecto a las propiedades asociadas a la comunicación con TS@. En caso de que se produzca un error durante el proceso y no sea posible obtener el sello de tiempo el método lanzará una excepción *SigningException*. Como parámetros recibe:

- **dataToStamp:** Parámetro que representa los datos a sellar.

- **applicationID:** Parámetro que representa el identificador de aplicación cliente para la comunicación con TS@.

- **signatureType:** Parámetro que representa la URI del tipo de sello de tiempo que generar. Los valores permitidos son:

  - **urn:ietf:rfc:3161** → Sello de tiempo RFC 3161.

  - **urn:oasis:names:tc:dss:1.0:core:schema:XMLTimeStampToken** → Sello de tiempo XML.

```
public static TimeStampToken getTimestampFromRFC3161Service(byte[ ] dataToStamp,
String applicationID, String tsaCommunicationMode) throws SigningException
```

Método que obtiene un sello de tiempo ASN.1 del servicio RFC 3161 de TS@. Para poder obtener el sello de tiempo de TS@ será necesario que el fichero **tsaXXXXX.properties** esté configurado correctamente respecto a las propiedades asociadas a la comunicación con TS@. En caso de que se produzca un error durante el proceso y no sea posible obtener el sello de tiempo el método lanzará una excepción *SigningException*. Como parámetros recibe:

- **dataToStamp:** Parámetro que representa los datos a sellar.

- **applicationID:** Parámetro que representa el identificador de aplicación cliente para la comunicación con TS@.

- **tsaCommunicationMode:** Parámetro que representa el protocolo definido para la comunicación con TS@. Los valores permitidos son:

  - **RFC3161-TCP** → Comunicación TCP.

  - **RFC3161-HTTPS** → Comunicación HTTPS.

  - **RFC3161-SSL** → Comunicación SSL.

```
public static X509Certificate getSigningCertificate(TimeStampToken tst) throws
SigningException
```

Método que obtiene el certificado firmante de un sello de tiempo ASN.1. En caso de que se produzca un error durante el proceso o no sea posible obtener el certificado el método lanzará una excepción *SigningException*. Como parámetros recibe:

- **tst:** Parámetro que representa el sello de tiempo ASN.1.

```
public static void validateXMLTimestamp(Element tst) throws SigningException
```

Método que valida estructuralmente un sello de tiempo XML (no valida el certificado firmante). En caso de que se produzca un error durante el proceso o si el sello de tiempo no es válido, el método lanzará una excepción *SigningException*. Como parámetros recibe:

- **tst:** Parámetro que representa el sello de tiempo XML que validar.

```
public static void validateASN1Timestamp(TimeStampToken tst) throws
SigningException
```

Método que valida estructuralmente un sello de tiempo ASN.1 (no valida el certificado firmante). En caso de que se produzca un error durante el proceso o si el sello de tiempo no es válido, el método lanzará una excepción *SigningException*. Como parámetros recibe:

- **tst:** Parámetro que representa el sello de tiempo ASN.1 que validar.

```
public static TimeStampToken getTimeStampToken(SignerInformation
signerInformation) throws SigningException
```

Método que obtiene un sello de tiempo ASN.1 de la información asociada al firmante de una firma, si es que ésta contiene un sello de tiempo. Si no contuviese un sello de tiempo se devolvería un valor nulo. En caso de que el sello de tiempo se encuentre mal formado el método lanzará una excepción *SigningException*. Como parámetros recibe:

- **signerInformation:** Parámetro que representa la información asociada al firmante.

```
public static Date getGenTimeXMLTimestamp(Element xmlTimestamp) throws
SigningException
```

Método que obtiene la fecha de generación de un sello de tiempo XML. En caso de que la fecha de generación del sello de tiempo no tenga un formato UTC el método lanzará una excepción *SigningException*. Como parámetros recibe:

- **xmlTimestamp:** Parámetro que representa el sello de tiempo XML.

```
public static void checkInputDocumentXMLTimeStamp(Element inputDocuments, Element
signature) throws TSAServiceInvokerException
```

Método que comprueba la integridad de un sello de tiempo XML, es decir, comprueba si el documento de entrada asociado al sello de tiempo es correcto. En caso de que se produzca algún error durante el proceso, o bien el documento de entrada no sea el correcto para el sello de tiempo, el método lanzará una excepción *TSAServiceInvokerException*. Como parámetros recibe:

- **inputDocuments:** Parámetro que representa el elemento *dss:InputDocuments* asociado al sello de tiempo XML que comprobar.

- **signature:** Parámetro que representa el elemento *ds:Signature* del sello de tiempo XML.

```
public static void checkInputDocumentRFC3161TimeStamp(Element inputDocuments,
TimeStampToken tst) throws TSAServiceInvokerException
```

Método que comprueba la integridad de un sello de tiempo RFC 3161, es decir, comprueba si el documento de entrada asociado al sello de tiempo es correcto. En caso de que se produzca algún error durante el proceso, o bien el documento de entrada no sea el correcto para el sello de tiempo, el método lanzará una excepción *TSAServiceInvokerException*. Como parámetros recibe:

- **inputDocuments:** Parámetro que representa el elemento *dss:InputDocuments* asociado al sello de tiempo RFC 3161 que comprobar.

- **tst:** Parámetro que representa el sello de tiempo RFC 3161 como un objeto ASN.1.

Clase UtilsXML

Esta clase proporciona funcionalidades relacionadas con el manejo de elementos XML. Está compuesta por los métodos:

```
public static Document parseDocument(Reader input) throws TransformersException
```

Método que obtiene un documento XML a partir de un flujo de entrada. En caso de que se produzca algún error durante el proceso el método lanzará una excepción *TransformersException*. Como parámetros recibe:

- **input:** Parámetro que representa los datos a sellar.

```
public static String getElementValue(Element e)
```

Método que obtiene el valor de un elemento XML. Como parámetros recibe:

- **e:** Parámetro que representa el elemento XML.

```
public static String getElementValue(Element e, String elementName)
```

Método que obtiene el valor de un elemento hijo de otro elemento XML. El elemento hijo debe estar localizado en el primer nivel de la estructura jerárquica del elemento XML padre. Como parámetros recibe:

- **e:** Parámetro que representa el elemento XML padre.
- **elementName:** Parámetro que representa el nombre del elemento hijo que obtener.

```
public static Element getElement(Element element, String elementName)
```

Método que obtiene un elemento hijo de otro elemento XML. El elemento hijo debe estar localizado en el primer nivel de la estructura jerárquica del elemento XML padre. Como parámetros recibe:

- **e:** Parámetro que representa el elemento XML padre.
- **elementName:** Parámetro que representa el nombre del elemento hijo que obtener.

```
public static Element searchChild(Element e, String childName)
```

Método que obtiene un elemento hijo de otro elemento XML padre. Como parámetros recibe:

- **e:** Parámetro que representa el elemento XML padre.
- **childName:** Parámetro que representa el nombre del elemento hijo que obtener.

```
public static List<Object> searchListChilds(Element e, String childElementName)
```

Método que obtiene una lista con los nodos hijos localizados en el primer nivel de la estructura jerárquica de hijos para un elemento XML padre. Como parámetros recibe:

- **e:** Parámetro que representa el elemento XML padre.
- **childElementName:** Parámetro que representa el nombre de los elementos hijo que obtener.

```
public static List<Element> searchChildElements(Element e)
```

Método que obtiene una lista con los elementos hijos de un elemento XML padre. Como parámetros recibe:

- **e:** Parámetro que representa el elemento XML padre.

```
public static Element createChild(Element e, String childName)
```

Método que crea un elemento hijo para un elemento XML padre. Como parámetros recibe:

- **e:** Parámetro que representa el elemento XML padre.

- **childName:** Parámetro que representa el nombre del elemento a crear.

```
public static boolean existsElement(Element e, String elementName)
```

Método que comprueba si existe un elemento XML como hijo de otro elemento XML. Devuelve un valor lógico que indica si el elemento XML padre contiene el elemento indicado (verdadero) o no (falso). Como parámetros recibe:

- **e:** Parámetro que representa el elemento XML padre.
- **elementName:** Parámetro que representa el nombre del elemento hijo a comprobar.

```
public static Element removeElement(Element e, String elementName)
```

Método que elimina un elemento hijo de otro elemento XML. Devuelve el elemento eliminado. Como parámetros recibe:

- **e:** Parámetro que representa el elemento XML padre.
- **elementName:** Parámetro que representa el nombre del elemento hijo a eliminar.

```
public static Element replaceElementValue(Element e, String elementName, String
elementValue)
```

Método que sustituye un elemento con un valor nuevo. Si el elemento no existe previamente, lo crea. Devuelve el elemento XML padre con el nuevo elemento hijo. Como parámetros recibe:

- **e:** Parámetro que representa el elemento XML padre.
- **elementName:** Parámetro que representa el nombre del elemento hijo a sustituir.
- **elementValue:** Parámetro que representa el nuevo valor del elemento hijo a sustituir.

```
public static Element createElementValue(Element e, String elementName, String
elementValue)
```

Método que crea un nuevo elemento hijo respecto a otro elemento XML padre. Devuelve el elemento padre actualizado con el nuevo elemento hijo. Como parámetros recibe:

- **e:** Parámetro que representa el elemento XML padre.
- **elementName:** Parámetro que representa el nombre del elemento hijo a crear.
- **elementValue:** Parámetro que representa el nuevo valor del elemento hijo a crear.

```
public static String toXMLString(Object xmlElement, String rootName, boolean
asAttributes) throws TransformersException
```

Método que obtiene una cadena de texto con los valores de un elemento XML. En caso de que se produzca un error durante el proceso el método lanzará una excepción *TransformersException*. Como parámetros recibe:

- **xmlElement:** Parámetro que representa el elemento XML.
- **rootName:** Parámetro que representa el nombre del elemento raíz.
- **asAttributes:** Parámetro que indica si los valores del elemento XML deben ser definidos como atributos (verdadero) o como elementos (falso).

```
public static String transformDOMtoString(Element xmlElement, boolean
omitXmlDeclaration) throws TransformersException
```

Método que genera una cadena de texto con formato XML a partir de una estructura DOM arbórea. En caso de que se produzca un error durante el proceso el método lanzará una excepción *TransformersException*. Como parámetros recibe:

- **xmlElement:** Parámetro que representa el elemento XML que procesar.
- **omitXmlDeclaration:** Parámetro que especifica si el procesador XSLT omitir la declaración XML (verdadero) o no (falso).

```
public static String transformDOMtoString(Document doc) throws
TransformersException
```

Método que genera una cadena de texto con formato XML a partir de un documento XML. En caso de que se produzca un error durante el proceso el método lanzará una excepción *TransformersException*. Como parámetros recibe:

- **doc:** Parámetro que representa el documento XML que procesar.

```
public static void deleteNodesNotUsed(Element xmlNode, String[ ]
optionalNodeTypes)
```

Método que elimina todos los nodos y etiquetas que no se usan para un elemento XML. El método busca todos los nodos de tipo *afirmaNodeType* y los elimina si el tipo es igual a alguno de los indicados como parámetro. Como parámetros recibe:

- **xmlNode:** Parámetro que representa el elemento XML que procesar.

- **optionalNodeTypes:** Parámetro que representa la colección con los tipos de nodos que eliminar.

```
public static void removeAfirmaAttribute(Element element)
```

Método que elimina todos los nodos de tipo *afirmaNodeType* contenidos dentro de un elemento XML. Como parámetros recibe:

- **element:** Parámetro que representa el elemento XML que procesar.

```
public static Element insertAttributeValue(Element xmlNode, String attributePath,
String value)
```

Método que añade un atributo a un nodo XML. Devuelve el nodo XML actualizado. Como parámetros recibe:

- **xmlNode:** Parámetro que representa el elemento XML que procesar.
- **attributePath:** Parámetro que representa la ruta que compone el nombre del atributo.
- **value:** Parámetro que representa el valor del atributo.

```
public static Element insertValueElement(Element element, String elementName,
String value)
```

Método que añade un nuevo elemento como hijo de otro elemento XML. Devuelve el elemento XML actualizado con el nuevo hijo. Como parámetros recibe:

- **element:** Parámetro que representa el elemento XML padre que procesar.
- **elementName:** Parámetro que representa el nombre del elemento que añadir.
- **value:** Parámetro que representa el valor del elemento que añadir.

```
public static String getAttributeValue(Element element, String elementPath, String
attributeName)
```

Método que obtiene el valor de un atributo perteneciente a un elemento XML. Como parámetros recibe:

- **element:** Parámetro que representa el elemento XML padre que procesar.
- **elementPath:** Parámetro que representa la ruta XPath del elemento XML que encontrar.
- **attributeName:** Parámetro que representa el nombre del atributo que encontrar.

```
public static String getAttributeValue(Element element, String attributeName)
```

Método que obtiene el valor de un atributo perteneciente a un elemento XML. Como parámetros recibe:

- **element:** Parámetro que representa el elemento XML que procesar.
- **attributeName:** Parámetro que representa el nombre del atributo que obtener.

```
public static String getRelativeXPath(Node nodeChild, Node parent)
```

Método que obtiene la ruta relativa a un nodo hijo a partir de su nodo padre. Como parámetros recibe:

- **nodeChild:** Parámetro que representa el nodo XML hijo.
- **parent:** Parámetro que representa el nodo XML padre.

```
public static String getNodeXPath(Node node)
```

Método que obtiene la ruta absoluta de un nodo. Como parámetros recibe:

- **node:** Parámetro que representa el nodo.

```
public static Element getFirstElementNode(Element element)
```

Método que obtiene el primer elemento XML hijo de otro elemento XML. Como parámetros recibe:

- **element:** Parámetro que representa el elemento XML que procesar.

```
public static Document getDocument(InputStream is) throws TransformerException
```

Método que obtiene un objeto que representa un documento XML a partir de un flujo de entrada. En caso de que se produzca algún error durante el proceso el método lanzará una excepción *TransformerException*. Como parámetros recibe:

- **is:** Parámetro que representa el flujo de entrada que procesar.

```
public static Document newDocument() throws ParserConfigurationException
```

Método que obtiene un nuevo documento XML. En caso de que se produzca algún error durante el proceso el método lanzará una excepción *TransformerException*.

```
public static Document getDocumentWithXsdValidation(File xsdSchema, InputStream
xml) throws TransformersException
```

Método que obtiene un documento XML a partir de un flujo de entrada y lo valida contra un esquema XSD. En caso de que se produzca un error durante el proceso o bien si el documento XML no es válido, el método lanzará una excepción *TransformersException*. Como parámetros recibe:

- **xsdSchema:** Parámetro que representa la definición del esquema XSD.
- **xml:** Parámetro que representa el flujo de entrada.

```
public static Document parseXMLDocument(String xml)
```

Método que obtiene un documento XML a partir de una cadena de texto. Como parámetros recibe:

- **xml:** Parámetro que representa el documento XML como cadena de texto.

Clase UtilsSignatureOp

Esta clase contiene un gran conjunto de mètodos utiles para el procesado y generación de firmas. Únicamente citaremos sus métodos más destacados:

```
public static Date getExpirationDate(byte[ ] signature)
```

Método que permite calcular la fecha de expiración de una firma a partir de su array de bytes. La fecha de expiración de una firma viene determinada por la fecha de caducidad más próxima del conjunto de certificados firmantes y certificados firmantes de sellos de tiempo. Como parámetro recibe:

- **signature:** Parámetro que representa el array de bytes de la firma a procesar.

9.5.4 Paquete es.gob.afirma.hsm

Este paquete contiene clases que gestionan el manejo de dispositivos HSM. Para poder hacer uso de los métodos definidos en este paquete debe estar configurado correctamente el archivo **hsm.properties**. De este paquete se describirán aquellas clases más destacadas.

Clase HSMKeystore

Esta clase maneja todas las operaciones relacionadas con almacenes de claves HSM. Está compuesta por los métodos:

```
public static PrivateKey getPrivateKey(String alias) throws HSMException
```

Método que obtiene una clave privada almacenada en un HSM a partir de su alias. En caso de que se produzca algún error durante el proceso, el método lanzará una excepción *HSMException*. Como parámetros recibe:

- **alias:** Parámetro que representa el alias de la clave privada a obtener.

```
public static X509Certificate getCertificate(String alias) throws HSMException
```

Método que obtiene un certificado almacenado en un HSM a partir de su alias. En caso de que se produzca algún error durante el proceso, el método lanzará una excepción *HSMException*. Como parámetros recibe:

- **alias:** Parámetro que representa el alias del certificado a obtener.

```
public static PrivateKeyEntry getPrivateKeyEntry(String alias) throws HSMException
```

Método que obtiene una entrada asociada a una clave privada almacenado en un HSM a partir de su alias. En caso de que se produzca algún error durante el proceso, el método lanzará una excepción *HSMException*. Como parámetros recibe:

- **alias:** Parámetro que representa el alias de la entrada a obtener.

9.6 Tipo de integración 6. Cifrado y descifrado de Datos

9.6.1 Paquete es.gob.afirma.encryption

Este paquete contiene las clases y constantes que permiten el cifrado y descifrado de datos.

Clase AlgorithmCipherEnum

Esta clase es una clase enumerada que representa los diferentes algoritmos, tanto simétricos como asimétricos, permitidos para realizar el cifrado y descifrado de datos. Está compuesta por los métodos.

```
public String getAlgorithm
```

Método que devuelve el nombre del algoritmo de cifrado.

```
public String getPaddingAlgorithm
```

Método que devuelve el modo de encadenamiento de bloques de cifrado con relleno.

Clase CipherIntegra

Esta clase gestiona la funcionalidad de cifrado y descifrado, tanto simétrico como asimétrico, de un conjunto de datos pasado como parámetro, seleccionando el algoritmo a utilizar entre uno de los especificados en el tipo AlgorithmCipherEnum y la clave a utilizar.

```
public CipherIntegra(AlgorithmCipherEnum algorithmCipherEnum, Key key) throws
CipherException
```

Constructor de la clase. Devuelve una instancia de la clase CipherIntegra configurada con el algoritmo y la clave correspondiente para el proceso de cifrado/descifrado. En el caso que se produzca algún error durante el proceso o los parámetros de entradas no son válidos, el método lanzará una excepción *CipherException,* indicando el motivo del error.

```
public String encrypt(String message) throws CipherException
```

Método que devuelve un objeto de tipo String que representa el conjunto de datos cifrados en base al algoritmo y la clave indicada al instanciar la clase. En el caso que se produzca algún error durante el proceso de encriptado, el método lanzará una excepción *CipherException*, indicando el motivo del error.

```
public String decrypt(String message) throws CipherException
```

Método que devuelve un objeto de tipo String que representa el conjunto de datos descifrado en base al algoritmo y la clave indicada al instanciar la clase. En el caso que se produzca algún error durante el proceso de desencriptado, el método lanzará una excepción *CipherException*, indicando el motivo del error.

9.6.2 Paquete es.gob.afirma.utils

Interfaz IEncryptionConstants

Esta interfaz contiene constantes utilizadas en el módulo Integra-encryption. Las constantes que se definen en esta interfaz representan cada uno de los algoritmos, simétricos y asimétricos, permitidos para el cifrado y descifrado de datos, así como los algoritmos de encadenamiento de bloques de cifrado con relleno asociado a cada uno de ellos.

9.7 Tipo de integración 7. Generación de informes de firma

9.7.1 Introducción XSL-FO

Antes de detallar las clases, métodos y parámetros que forman este tipo de intagación, es necesario abordar en qué consiste el proceso de generación de informes de firma.

La generación de informe PDF está basado en la utilización de XSL-FO (eXtensible Stylesheet Language Formatting Objects). XSL-FO es una recomendación W3C que mediante XML especifica cómo se van a formatear unos datos para presentarlos en pantalla, papel u otros medios (puede consultarse información sobre las especificaciones XSL-FO en http://www.w3.org/TR/xsl11/).

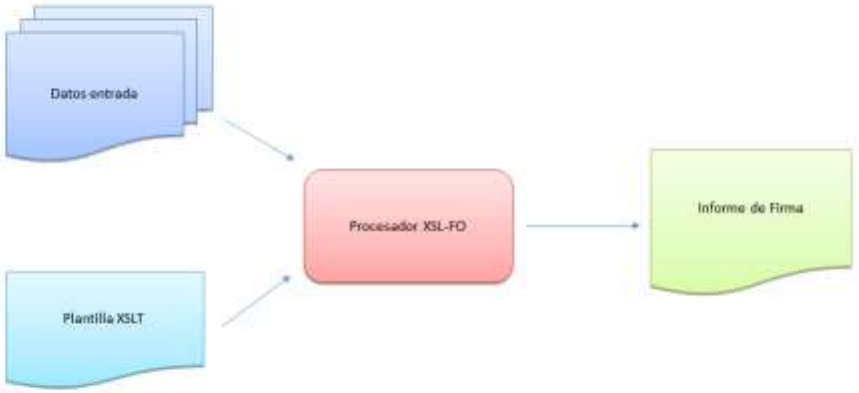
El proceso de generación es el siguiente:

1. Una aplicación solicita mediante invocación del API, la generación de un informe de firma. La petición incluye los datos firmados (documento original), una plantilla XSLT, información sobre cómo va a incluirse el documento e información adicional para la generación del informe de firma.

2. A partir de estos parámetros de entrada, el API construye un XML previamente definido.

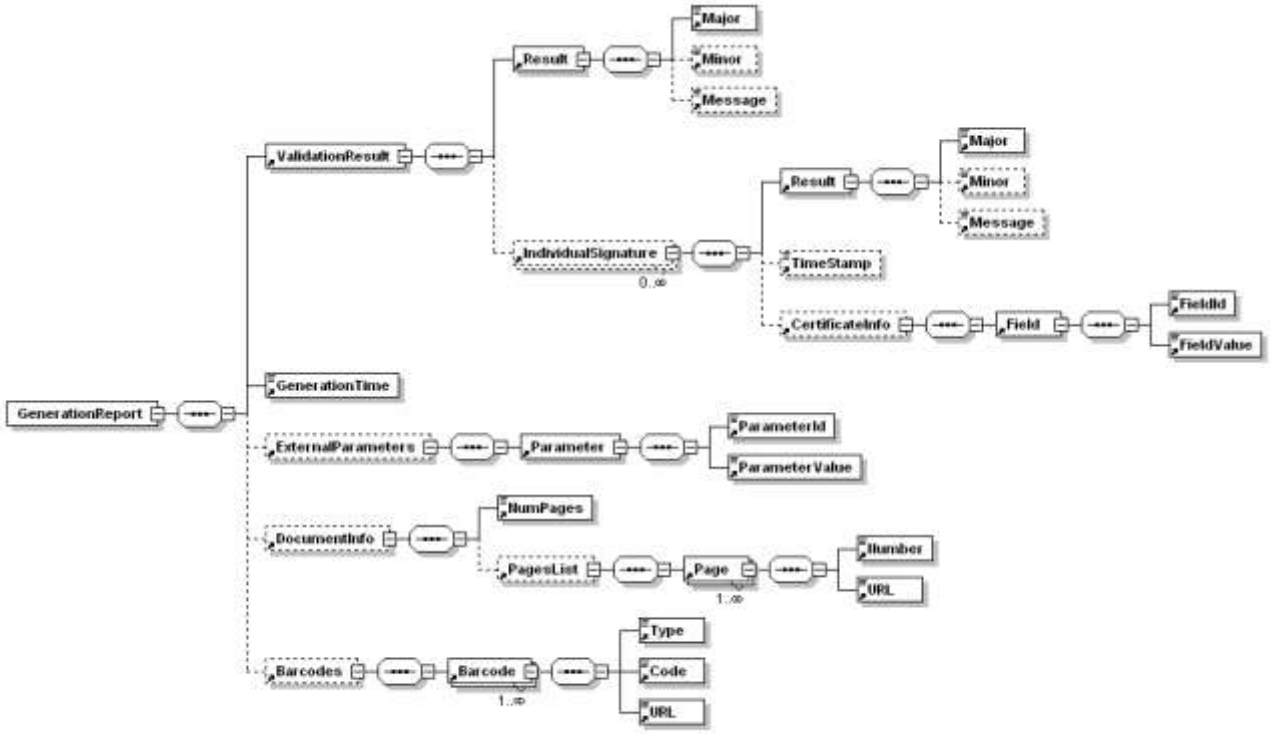
3. La aplicación obtiene un documento XSL-FO a partir del XML creado en el paso anterior y de la plantilla XSLT.

4. Una vez obtenido el documento XSL-FO podemos generar el documento PDF mediante el procesador XSL-FO "Apache-FOP" (http://xmlgraphics.apache.org/fop/).



9.7.2 XML de datos de entrada

Los datos que se desean mostrar en el informe de firma se insertan en un XML predefinido. Los tipos de datos que aceptará el API, reflejan en gran medida la estructura de este XML. Para el API de generación de informes de firma, la estructura de este documento está representada en la siguiente imagen



En un apartado posterior, cuando se defina el paquete que contiene las clases que representan estos datos, se entrará en el detalle de dichas estructuras.

La construcción de este XML, forma parte de un proceso interno del API que es transparente para la aplicación invocante, pero es útil conocerlo para construir adecuadamente los parámtros de entrada.

9.7.3 Plantilla XSLT

En este proceso interviene un elemento al que denominamos plantilla XSLT. Estas plantillas son un conjunto de reglas (expresadas mediante scripts XML) que se aplican cuando se hacen coincidir con nodos específicos del XML de entrada definido anteriormente. Estas reglas son lo que nos permitirá mostrar la información deseada con un diseño determinado. Por lo tanto, podemos decir que el qué y el cómo veremos la información en el informe PDF resultante, dependerá de dicha plantilla XSLT.

Para diseñar y obtener las plantillas XSLT, primero debemos diseñar el informe de firma. Puede consultarse más información acerca de cómo diseñar plantillas en el manual de eVisor "*@Firma-eVisor-Progs-Templates-MAN*".

Además, existe la posibilidad de obtener estas plantillas a través de la herramienta visual denominada "evisorTemplates" (editor de plantillas de eVisor) cuyo funcionamiento se detalla en el manual *"3114 - MINHAFP -eVisorTemplates-USER-MAN_1.2"*.

9.7.4 Modos de inclusión del documento original en el informe de firma

El documento original siempre aparece representado en el informe de firma. El API de generación de informes de firma contempla dos modos de inclusión:

- Embebido: El documento original se incrusta dentro del PDF generado. Para poder usar este modo de inclusión, será necesario indicar en la plantilla XSLT la zona del documento resultante donde se embeberá el documento origianl.

- Concatenado: El documento original se concatena con el PDF generado atendiendo a una determinada regla que indicará el orden y páginas de la concatenación. La cadena REP representa al informe de firma y la cadena DOC al documento originalmente firmado.

  Ejemplos:

  - REP + DOC: Se concatena el documento resultante de la trasformación XSL-FO + El documento originalmente firmado.

  - REP(1)+DOC(3-10)+REP(2): Se concatena la primera página resultante de la trasformación XSL-FO + las paginas de la 3 a 10 (ambas inclusives) + la segunda página resultante de la trasformación XSL-FO.

9.7.5 Elementos del tipo de integración

Para este tipo de integración, el paquete base está situado en "*es.gob.afirma.mreport*". Los integradores deberán hacer uso principalmente de tres elementos:

- Método de invocación para realizar la generación del informe de firma.

- Clases POJO de datos que representan algunos de los parámetros de entrada del método de generación de informes de firma.

- Clase de excepción para el control de errores desde la aplicación invocante.

9.7.6 Paquete es.gob.afirma.mreport.pdf

Este paquete contiene la clase principal del API donde se encuentra el método que deberá ser invocado para generar el informe de firma.

Clase PdfReportManager

La signatura de ficho método es la siguiente:

```
public byte[] createReport(ValidationData validationData, DocInclusionData docIncData, byte[] xsltTemplate,
byte[] document, ArrayList<Barcode> barcodes, ArrayList<FileAttachment> attachments, HashMap<String,
String> additionalParameters) throws SignatureReportException
```

Éste método realiza las transformaciones necesarias para obtener el informe de firma del documento original firmado, a partir de una serie de parámetros de entrada que se especifican a continuación:

- validationData: parámetro de tipo ValidationData que representa la información referente al resultado de la validación de la firma incluída en el documento original. Es un parámetro **opcional que puede formar parte del XML de entrada**.

- docIncData: parámetro de tipo DocInclusionData que representa a aquellos datos necesarios para que el API pueda incluir el documento original en el informe de firma de manera adecuada.. Es un parámetro **obligatorio que no forma parte del XML de entrada**.

- xsltTemplate: parámetro de tipo byte[] que representa la plantilla XSLT que se utilizará para realizar la transformación XSL-FO y determinará el diseño del informe de firma, incluyendo elementos como tablas o imágenes. Es un parámetro **obligatorio que no forma parte del XML de entrada**.

- document: parámetro de tipo byte[] que representa al documento original firmado **en formato PDF**. Es un parámetro **obligatorio**. No forma parte tal cual del XML de entrada, aunque información como el úmero de páginas y su orientación se extraen de él y se añadirán al XML

- barcodes: parámetro de tipo ArrayList<Barcode> que representa la lista de objetos Barcode, cada uno de los cuales contendrá información de un código de barras a incluir en el informe de firma. Es un parámetro **opcional que puede formar parte del XML de entrada**.

- attachments: parámetro de tipo ArrayList<FileArrachment> que representa la lista de objetos FileAttachment, cada uno de los cuales contendrá información de un fichero adjunto que se desea incluir en el informe de firma. Es un parámetro **opcional que no forma parte del XML de entrada**.

- additionalParameters: parámetro de tipo HashMap<String,String> que representa un conjunto de pares clave/valor para incluir en el informe datos libres mediante el uso de variables en la plantilla XSLT. Es un parámetro **opcional que puede formar parte del XML de entrada**.

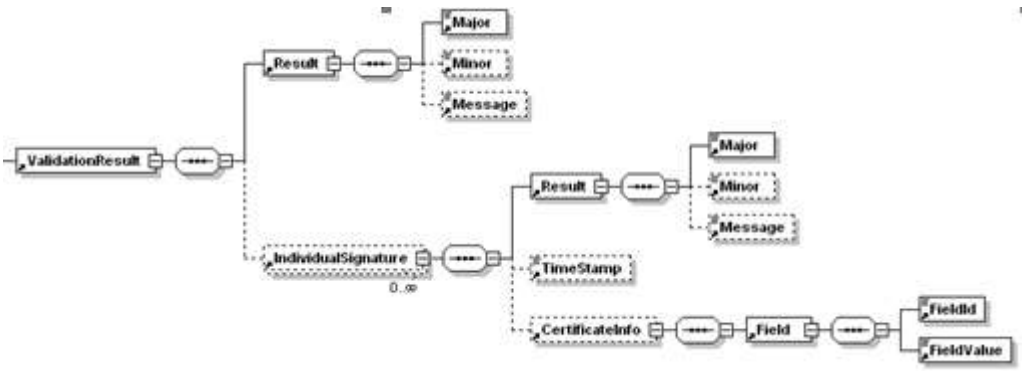
Si alguno de los parámetros de entrada es erróneo u ocurre algún problema durante el proceso de generación, el método lanzará una excepción *SignatureReportException*.

9.7.7 Paquete es.gob.afirma.mreport.items

Este paquete, contiene una serie de clases POJO que representan a las estructuras de datos que intervienen en los procesos de transformación. Algunas de estas clases son usadas de manera interna por el API. A continuación se detallan aquellas clases necesarias para construir algunos de los parámetros de entrada:

Clase ValidationData

El propósito de esta clase, es modelar como parámetro de entrada la estructura que contiene los datos de validación de firma del documento original y forman parte del XML de entrada



Los atributos de esta clase son:

```
/**
 * Attribute that represents the result major of the signature validation.
 */
private String resultMajor;
```

```
/**
 * Attribute that represents the result minor of the signature validation.
 */
private String resultMinor;
```

```
/**
 * Attribute that represents the result message of the signature validation.
 */
private String resultMessage;
```

```
/**
 * Attribute that represents the date format of each individual signature validation timestamp.
 */
private String timestampFormat;
```

```
/**
 * Attribute that represents the list of individual signature validations.
 */
private List<IndividualSignature> signatures;
```

Los métodos de esta clase son un constructor con parámetros y métodos de acceso para todos sus atributos.

Clase IndividualSignature

El propósito de esta clase, es modelar parte del parámetro de entrada que representa la inforamción de validación de la firma. Forma parte de la estructura de ValidationData y tiene una cardinalidad 0…n dentro de esta clase.

Los atributos de esta clase son:

```
/**
 * Attribute that represents the result major of the individual signature validation.
 */
private String resultMajor;
```

```
/**
 * Attribute that represents the result minor of the individual signature validation.
 */
private String resultMinor;
```

```
/**
 * Attribute that represents the result message of the individual signature validation.
```

```
 */
private String resultMessage;
```

```
/**
 * Attribute that represents the timestamp of the individual signature validation.
 */
private LocalDateTime timestamp;
```

```
/**
 * Attribute that represents the certificate information of the individual signature.
 */
LinkedHashMap<String, String> certInfo = new LinkedHashMap<String,String>();
```

Los métodos de esta clase son un constructor con parámetros y métodos de acceso para todos sus atributos.

Clase DocInclusionData

El propósito de esta clase es represetar al parámetro de entrada que indica la información referente a los modos de inclusión del documento original en el informe de firma.

Los atributos de esta clase son:

```
/**
 * Attribute that represents the inclusion mode of the original document
 * 0 - Embedded
 * 1 - Concatenated
 */
private int docInclusionMode;
```

```
/**
 * Attribute that represents the concatenation rule for including the original document in the report in the mode selected was 'concatenate'.
 */
private String docConcatRule;
```

Los métodos de esta clase son los propios de acceso a los atributos.

Clase Barcode

El propósito de esta clase es representar el parámetro de entrada que indica la información referente a cada código de barra diferente que va a incluirse en el informe de firma generado.

La estructura de esta clase, no se corresponde con la del XML de entrada expuesta en el apartdo XML de datos de entrada, ya que antes de crearse la estructura del XML, esta información se procesa de manera interna, generando la información necesaria. Desde el punto de vista del integrador, esta es la información que debe proporcionarse.

Atributos de esta clase:

```
/**
    * Attribute that represents the type of bar code to create.
```

```
    */
   private String type = null;
```

```
/**
 * Attribute that represents the message used to create the bar code.
 */
 private String message = null;
```

```
/**
* Attribute that represents additional parameters of configuration.
*/
private LinkedHashMap<String, String> configuration = new LinkedHashMap<String, String>();
```

Los métodos de esta clase son un constructor con parámetros y métodos de acceso para todos sus atributos.

Clase FileArrachment

El propósito de esta clase es representar el parámetro de entrada que indica los archivos adjuntos que van a insertarse al informe de firma.

Atributos de esta clase:

```
/**
* Attribute that represents the attachment name.
*/
private String name = null;
```

```
/**
* Attribute that represents the attachment description.
*/
private String description = null;
```

```
/**
* Attribute that represents the attachment content.
*/
private byte[] content = null;
```

Los métodos de esta clase son un constructor con parámetros y métodos de acceso para todos sus atributos.

9.7.8 Paquete es.gob.afirma.mreport.exceptions

Contiene las excepciones que utiliza el API. Algunas de estas excepciones son de uso interno, se indicará únicamente la clase relevante para el integrador.

Clase SignatureReportException

Esta clase representa la excepción que lanzará la API en caso de detectarse un problema al validar algún parámetro de entrada (por ejemplo, que el documento original no sea PDF o que la plantilla XSLT proporcionada no sea válida) o en caso de producirse un error durante el proceso de transformación.

9.8 Tipo de integración 8. Validación de Certificados mediante TSL

9.8.1 Paquete es.gob.afirma.tsl

Este paquete contiene las clases y constantes que permiten la obtención de la TSL a utilizar, y la validación de certificados mediante la TSL obtenida.

Interfaz ITslValidation y clase TslValidation

Esta clase gestiona la funcionalidad de validación de certificados frente a una TSL. Implementa los siguientes métodos:

```
ITSLObject getTSLObjectFromPath(String pathTsl)
```

Este método recibe como parámetro la ruta absoluta donde se localiza en local un fichero XML con la TSL a utilizar en la validación. Devuelve el objeto ITSLObject que representa dicha TSL mapeada.

```
ITSLObject downloadTSLbyHTTP(String uriTSL, int connectionTimeout, int
readTimeout)
```

Este método recibe como parámetros la URI donde se encuentra publicada la TSL que se utilizará para la validación del certificado, tiempo de espera de conexión definido para la comunicación con el servidor donde se descargará la TSL en milisegundos y tiempo de espera de lectura en milisegundos. Devuelve el objeto ITSLObject que representa dicha TSL mapeada.

```
DetectCertInTslInfoAndValidationResponse validateCertificateTsl(byte[]
certByteArrayB64, TSLObject tslObject, Date date, boolean getInfo, boolean
checkRevStatus)
```

Este método recibe los siguientes parámetros de entrada:

- certByteArrayB64: array de bytes con el certificado a validar. El certificado tiene que ser de tipo x.509v3. Parámetro obligatorio.
- tslObject: objeto TSLObject, que representa la TSL mapeada, este objeto será el devuelto por alguno de los dos métodos indicados anteriormente. Parámetro obligatorio.
- date: fecha a utilizar para validar el certificado a validar (formato permitido: dd/MM/aaaa HH:mm:ss). Parámetro opcional, si no se introduce se utilizará la fecha actual.
- getInfo: valor booleano para indicar si la respuesta debe contener la información sobre el certificado.

- checkRevStatus: valor booleano para indicar si es necesario verificar el estado de revocación del certificado de entrada.

Después de una serie de comprobaciones, el método devuelve un objeto de tipo DetectCertInTslInfoAndValidationResponse, clase que representa toda la información que se obitene de la TSL utilizada. A continuación, se detalla los distintos atributos que componen esta clase:

- **status: (Integer/int)** Código que representa el resultado general de la petición.

- **Description. (String)** En este campo se describe el estado que devuelve la petición.

En la siguiente tabla se indica los posibles estados que puede devolver el servicio ***detectCertInTslInfoAndValidation,*** junto a su descripción***:***

Código	Descripción
0	Ha ocurrido un error en los parámetros de entrada.
1	Ha ocurrido un error al ejecutar el servicio.
20	Indica que no se ha encontrado la TSL para detectar el certificado.
21	Indica que se ha encontrado la TSL para detectar el certificado.
22	Indica que el certificado no ha sido detectado en la TSL.
23	Indica que el certificado ha sido detectado en la TSL.
24	Indica que el certificado ha sido detectado en la TSL y no se ha obtenido la información de mapeos asociada a éste.
25	Indica que el certificado ha sido detectado en la TSL y se ha obtenido la información de mapeos asociada a este.
26	Indica que el certificado ha sido detectado en la TSL y se ha obtenido la información de revocación asociada a éste.
27	Indica que el certificado ha sido detectado en la TSL y NO se ha obtenido la información de revocación asociada a este.
28	Indica que el certificado ha sido detectado en la TSL, NO se ha obtenido la información de mapeos asociada a este y NO se ha obtenido la información de revocación asociada a este.
29	Indica que el certificado ha sido detectado en la TSL, NO se ha obtenido la información de mapeos asociada a este y se ha obtenido la información de

	revocación asociada a este.
30	Indica que el certificado ha sido detectado en la TSL, se ha obtenido la información de mapeos asociada a éste y NO se ha obtenido la información de revocación asociada a este.
31	Indica que el certificado ha sido detectado en la TSL, se ha obtenido la información de mapeos asociada a este y se ha obtenido la información de revocación asociada a éste.

- **resultTslInfVal:** Objeto que será opcional en la respuesta dependiendo de la ejecución del servicio y que contendrá lo siguiente.
  - **tslInformation:** objeto de tipo TslInformation, que contendrá la información básica de la TSL. Este objeto está compuesto de los siguientes atributos:
    - **etsiSpecificationAndVersion:** especificación ETSI y versión que implementa la TSL.
    - **countryRegion:** código del país/región al que pertenece la TSL.
    - **sequenceNumber:** el número de secuencia de la TLS.
    - **TslLocation:** URI que representa la localización de la TSL.
    - **Issued:** representa la fecha de emisión de la TSL.
    - **tslXMLData**: para la respuesta de este servicio, este parámetro siempre es nulo.
  - **certDetectedInTSL:** objeto de tipo CertDetedtedInTSL, que contendrá la información del certificado detectado en la TSL. Este objeto está compuesto por los atributos:
    - **tspName:** nombre del TSP en el que el certificado ha sido detectado.
    - **tspServiceInformation:** objeto de tipo TspServiceInformation que contendrá la información del servicio TSP que ha detectado el certificado. Este objeto está compuesto por el siguiente atributo:
      - **tspServiceHistoryInf:** objeto de tipo TspServiceHistoryInf que contendrá la información del historial del servicio TSP (si se usa). Este objeto está compuesto de los siguientes atributos:

- **tspServiceName:** nombre del servicio TSP en el que el certificado ha sido detectado.
- **tspServiceType:** identificador del tipo de servicio TSP en el que el certificado ha sido detectado. El identificador viene expresado mediante una URI. Los distintos valores que puede tomar este parámetro vienen descritos en el documento ETSI TS 119 612 V2.1.1 (2015-07), apartado 5.5.1.
- **tspServiceStatus:** URI que identifica el estado actual en el que se encuentra el servicio TSP en el que se ha detectado el certificado. Los distintos valores que puede tomar este parámetro vienen descritos en el documento ETSI TS 119 612 V2.1.1 (2015-07), apartado 5.5.4.
- **tspServiceStatusStartingDate:** fecha de inicio del estado del servicio TSP.

▪ **certInfo:** en este parámetro se muestra los distintos mapeos asignados para el certificado validado. Esta información se incluirá en el resultado, si en la petición se hubiera indicado el parámetro de entrada "getInfo" a true. Se mostrará los siguientes mapeos:

- **certQualified:** Mapeo que indica si se considera qualified el certificado validado:
  - NO: El certificado no es qualified.
  - YES: El certificado es qualified.
  - UNKNOWN: Se desconoce si el certificado es qualified.
- **certClassification:** Mapeo que determina el tipo del certificado:
  - NATURAL_PERSON: Certificado de persona física.
  - LEGAL_PERSON: Certificado de persona jurídica.
  - ESEAL: Certificado de sello electrónico (de tiempo).

- ESIG: Certificado para firma electrónica (persona física).
- WSA: Certificado para autenticación de servidor web (de componentes).
- UNKNOWN: Se desconoce el tipo del certificado.
- TSA: De sello de tiempo.

- qscd: Mapeo que determina si el certificado se encuentra almacenado en un SSCD/QSCD:

- NO: El certificado no se encuentra en un SSCD/QSCD.
- YES: El certificado se encuentra en un SSCD/QSCD.
- YES_MANAGED_ON_BEHALF: El certificado se encuentra en un SSCD/QSCD controlado por un tercero autorizado.
- UNKNOWN: Se desconoce si el certificado está en un SSCD/QSCD.

▪ **tslRevocationStatus:** objeto de tipo TslRevocationStatus, contendrá toda la información referente a la revocación. Este objeto será opcional en la respuesta dependiendo de la ejecución del servicio y si en la petición, el parámetro "checkRevocationStatus" es true. En el caso que se incluya, contendrá los siguientes atributos:

- revocationStatus: Valor entero, que identificará los distintos estados de revocación disponibles.
- revocationDesc: descripción del motivo de revocación.

A continuación, se enumeran los diferentes estados posibles:

Código	Descripción
1	No ha sido posible determinar el estado de revocación del certificado.
2	Se ha comprobado el estado de revocación del

	certificado: OK.
3	Se ha comprobado el estado de revocación del certificado: Revocado.
4	Se ha comprobado el estado de revocación del certificado: Caducado: Cadena de certificación no válida (ha expirado su periodo de validez).
5	Se ha comprobado el estado de revocación del certificado: Revocado: El servicio que lo reconoce está revocado.
6	Se ha comprobado el estado de revocación del certificado: Unknown: El servicio que reconoce el certificado ha dejado de estar supervisado.

- isFromServicesStatus: valor booleano
- url: URL de donde se ha obtenido la evidencia de revocación (la respuesta OCSP o la CRL). Esta información sólo se devuelve si el parámetro isFromServicesStatus es false.
- DP-AIA: Parámetro booleano que indica si la evidencia de revocación se ha obtenido desde DistributionPoint o de AIA del certificado, en ese caso tendría el valor "true", en caso contrario "false". Esta información solo se devuelve si el parámetro isFromServicesStatus es false.
- tslServiceInformation: objeto TspServiceInformation que será incluido en la respuesta en el caso en el que DP-AIA fuera false, es decir, la evidencia de revocación se obtenga a través de un servicio. Este parámetro contendrá la información del servicio TSP que ha validado el certificado. Este parámetro, viene definido por un atributo tspServiceHistoryInf, objeto de Tipo TspServiceHistoryInf que está compuesto por los siguientes atributos:
    - tslServiceName: Cadena con el nombre del servicio.
    - tspServiceType: Cadena con el identificador del tipo de servicio. Como indicamos anteriormente, los posibles valores

que pueden aparecer en este campo se detallan en el documento ETSI TS 119 612 V2.1.1 (2015-07), apartado 5.5.1.

- tspServiceStatus: Cadena con el estado actual del servicio. Como indicamos anteriormente, los posibles valores que puede tomar este parámetro se detallan en el documento ETSI TS 119 612 V2.1.1 (2015-07), apartado 5.5.4.

- evidenceType: valor entero, que tomará el valor 1, si el tipo de evidencia es una respuesta OCSP, y 2, si es CRL.

- evidence: array de bytes en Base64, que representa la evidencia del estado de revocación consultada.

- revocationReason: valor entero, que identificará el motivo de revocación recogido en la CRL o respuesta OCSP, en el caso en el que el certificado esté revocado. Los diferentes motivos por los que un certificado es revocado se detallan en el documento https://www.ietf.org/rfc/rfc5280.txt, en el apartado 5.3.1.

- revocationDate: Fecha de revocación, en el caso que el certificado esté revocado.

9.8.2 Paquete es.gob.afirma.tsl.access

Este paquete contiene la clase TSLManager, clase principal mediante la que se realizan todas las operaciones asociadas a una TSL, como puede ser el mapeo de una TSL desde un fichero XML, como la validación de un certificado X509v3.

ANEXO

A.1 Validación de Firmas

En este anexo se describirán las operaciones de validación de firma que se aplicarán en función de su formato.

A.1.1 Validación de firmas CAdES

Integr@ permite la validación de firmas electrónicas CAdES, desde su implementación propia de la interfaz ***es.gob.afirma.signature.Signer*** (***es.gob.afirma.signature.cades.CadesSigner***) y desde la fachada de firma, (***es.gob.afirma.integraFacade.IntegraFacade***) Así, se definen las siguientes tareas de validación a nivel general:

- ✓ **Validación de la Integridad:** Se comprobará que la firma indicada posee al menos un firmante, y que, en el caso de indicar los datos firmados, éstos coinciden con los datos contenidos en la firma.

Para cada uno de los firmantes contenidos en la firma, se definen las siguientes tareas de validación :

- ✓ **Validación del Núcleo de Firma:** Se comprobará que el firmante verifica la firma.
- ✓ **Validación de la Información de Clave Pública:** Se comprobará que el firmante incluye el atributo firmado signing-certificate o el atributo firmado signing-certificate-v2, y que dicho atributo identifica al certificado del firmante. Además, en el caso de que el atributo que incluya de los dos sea signing-certificate se comprobará que el algoritmo de firma utilizado ha sido SHA-1.
- ✓ **Validación del Instante de Firma:** Si el firmante incluye el atributo firmado signing-time se comprobará que dicho atributo está bien formado y que la fecha contenida en el mismo es anterior a la fecha de validación. Esta fecha de validación será la fecha del sello de tiempo menos reciente contenido en un atributo no firmado signature-time-stamp, si dicho atributo se incluye. En caso contrario, la fecha de validación será la fecha actual.
- ✓ **Validación de la Política de Firma:** Si el firmante incluye el atributo firmado signature-policy-identifier se comprobará si el OID de la política de firma definida en dicho atributo coincide con el OID de la política de firma definida para firmas ASN.1 en el fichero **integra.properties**, en cuyo caso, se comprobará que los datos de la firma y del firmante concreto son válidos respecto a las propiedades definidas en dicho fichero.
- ✓ **Validación del Certificado Firmante:** Se comprobará el estado del certificado firmante respecto a la fecha de validación respecto del método de validación definido para el mismo en el fichero **integra.properties**. La fecha de validación será la fecha del sello de tiempo menos

reciente contenido en un atributo no firmado signature-time-stamp, si dicho atributo se incluye. En caso contrario, la fecha de validación será la fecha actual.

- ✓ **Validación de los Atributos signature-time-stamp:** Si el firmante posee atributos signature-time-stamp se comprobará que todos ellos poseen una estructura correcta y que los sellos de tiempo que contienen están bien formados. Respecto a cada sello de tiempo se definen las siguientes tareas de validación:
  - **Validación de la Firma del Sello de Tiempo:** Se comprobará que la estructura del sello de tiempo así como su núcleo de firma son correctos.
  - **Validación de la Integridad del Sello de Tiempo:** Se comprobará que los datos sellados son correctos.
  - **Validación del Certificado Firmante del Sello de Tiempo:** Se comprobará el estado del certificado firmante del sello de tiempo respecto a la fecha de generación del siguiente sello de tiempo, utilizando el método de validación definido para los certificados firmantes en el fichero **integra.properties**. Cuando se esté procesando el certificado firmante del sello de tiempo más reciente (y por lo tanto el último) se utilizará como fecha de validación la fecha actual. Además, se verificará que el certificado posee la extensión id-kp-timestamp.

A.1.2 Validación de firmas CAdES Baseline

Integr@ permite la validación de firmas electrónicas CAdES Baseline, desde su implementación propia de la interfaz ***es.gob.afirma.signature.Signer*** (***es.gob.afirma.signature.cades.CAdESBaselineSigner***) y desde la fachada de firma, (***es.gob.afirma.integraFacade.IntegraFacade***) Así, se definen las siguientes tareas de validación a nivel general:

- ✓ **Validación de la Integridad:** Se comprobará que la firma indicada posee al menos un firmante, y que, en el caso de indicar los datos firmados, éstos coinciden con los datos contenidos en la firma.

Para cada uno de los firmantes contenidos en la firma, se definen las siguientes tareas de validación:

- ✓ **Validación del Núcleo de Firma:** Se comprobará que el firmante verifica la firma y que incluye el atributo firmado content-type y que éste posee el valor "id-data".
- ✓ **Validación de la Información de Clave Pública:** Se comprobará que el firmante incluye el atributo firmado signing-certificate o el atributo firmado signing-certificate-v2, y que dicho atributo identifica al certificado del firmante. En el caso de que el atributo que incluya de los dos sea signing-certificate se comprobará que el algoritmo de firma utilizado ha sido SHA-1. Además, se comprobará que el elemento SignedData.certificates incluye, al menos, el certificado firmante.

- ✓ **Validación del Instante de Firma:** Se comprobará que el firmante incluye el atributo firmado signing-time, que dicho atributo está bien formado y que la fecha contenida en el mismo es anterior a la fecha de validación. Esta fecha de validación será la fecha del sello de tiempo menos reciente contenido en un atributo no firmado signature-time-stamp, si dicho atributo se incluye. En caso contrario, la fecha de validación será la fecha actual.

- ✓ **Validación de la Política de Firma:** Si el firmante incluye el atributo firmado signature-policy-identifier se comprobará si el OID de la política de firma definida en dicho atributo coincide con el OID de la política de firma definida para firmas ASN.1 en el fichero **integra.properties**, en cuyo caso, se comprobará que los datos de la firma y del firmante concreto son válidos respecto a las propiedades definidas en dicho fichero.

- ✓ **Validación del Certificado Firmante:** Se comprobará el estado del certificado firmante respecto a la fecha de validación respecto del método de validación definido para el mismo en el fichero **integra.properties**. La fecha de validación será la fecha del sello de tiempo menos reciente contenido en un atributo no firmado signature-time-stamp, si dicho atributo se incluye. En caso contrario, la fecha de validación será la fecha actual.

- ✓ **Validación de los Atributos signature-time-stamp:** Si el firmante posee atributos signature-time-stamp se comprobará que todos ellos poseen una estructura correcta y que los sellos de tiempo que contienen están bien formados. Respecto a cada sello de tiempo se definen las siguientes tareas de validación:

    - **Validación de la Firma del Sello de Tiempo:** Se comprobará que la estructura del sello de tiempo así como su núcleo de firma son correctos.

    - **Validación de la Integridad del Sello de Tiempo:** Se comprobará que los datos sellados son correctos.

    - **Validación del Certificado Firmante del Sello de Tiempo:** Se comprobará el estado del certificado firmante del sello de tiempo respecto a la fecha de generación del siguiente sello de tiempo, utilizando el método de validación definido para los certificados firmantes en el fichero integra.properties. Cuando se esté procesando el certificado firmante del sello de tiempo más reciente (y por lo tanto el último) se utilizará como fecha de validación la fecha actual. Además, se verificará que el certificado posee la extensión id-kp-timestamp.

A.1.3 Validación de firmas XAdES

Integr@ permite la validación de firmas electrónicas XAdES, desde su implementación propia de la interfaz ***es.gob.afirma.signature.Signer*** (***es.gob.afirma.signature.xades.XadesSigner***) y desde la fachada de firma, (***es.gob.afirma.integraFacade.IntegraFacade***) Así, se definen las siguientes tareas de validación a nivel general:

- ✓ **Validación de la Integridad:** Se comprobará que el documento XML posee al menos una firma, y que los datos firmados incluídos en la misma son acordes respecto al modo en que se ha realizado la firma (*detached*, *enveloped*, o *enveloping*).

Para cada uno de los firmantes contenidos en el documento XML firmado, se definen las siguientes tareas de validación:

- ✓ **Validación del Núcleo de Firma:** Se realizarán las siguientes verificaciones
  - Se comprobará que el firmante verifica la firma.
  - Se comprobará que el elemento QualifyingProperties está presente y se encuentra correctamente formado.
  - Se comprobará que el elemento SignedSignatureProperties tiene una estructura correcta.
  - Se comprobará que existe una referencia que apunta al elemento SignedProperties.
- ✓ **Validación de la Información de Clave Pública:** Si se incluye el elemento firmado SigningCertificate, se comprobará que éste se corresponde con el certificado firmante. Si no se incluye dicho elemento, se comprobará que existe al menos un elemento KeyInfo y éste, a su vez, contiene el certificado firmante. Además, se comprobará que existe una referencia al elemento KeyInfo.
- ✓ **Validación del Instante de Firma:** Si se incluye el elemento firmado SigningTime se comprobará que está correctamente formado y que contiene una fecha anterior a la fecha de validación. Esta fecha de validación será la fecha del sello de tiempo menos reciente contenido en un elemento no firmado SignatureTimeStamp, si dicho elemento se incluye. En caso contrario, la fecha de validación será la fecha actual.
- ✓ **Validación de la Política de Firma:** Si se incluye el elemento firmado SignaturePolicyIdentifier se comprobará si el identificador (URN o URI) de la política de firma definida en dicho elemento coincide con el identificador de la política de firma definida para firmas XML en el fichero **integra.properties**, en cuyo caso, se comprobará que los datos de la firma y del firmante concreto son válidos respecto a las propiedades definidas en dicho fichero.
- ✓ **Validación del Certificado Firmante:** Se comprobará el estado del certificado firmante respecto al método de validación definido para el mismo en el fichero **integra.properties**. La fecha de validación será la fecha del sello de tiempo menos reciente contenido en un elemento no firmado SignatureTimeStamp, si dicho elemento se incluye. En caso contrario, la fecha de validación será la fecha actual.
- ✓ **Validación de los Elementos SignatureTimeStamp:** Si el firmante posee elementos SignatureTimeStamp se comprobará que todos ellos poseen una estructura correcta y que los

sellos de tiempo que contienen están bien formados. Respecto a cada sello de tiempo se definen las siguientes tareas de validación:

- **Validación de la Firma del Sello de Tiempo:** Se comprobará que la estructura del sello de tiempo así como su núcleo de firma son correctos.

- **Validación de la Integridad del Sello de Tiempo:** Se comprobará que los datos sellados son correctos.

- **Validación del Certificado Firmante:** Se comprobará el estado del certificado firmante del sello de tiempo respecto a la fecha de generación del siguiente sello de tiempo, utilizando el método de validación definido para los certificados firmantes en el fichero **integra.properties**. Cuando se esté procesando el certificado firmante del sello de tiempo más reciente (y por lo tanto el último) se utilizará como fecha de validación la fecha actual. Además, se verificará que el certificado posee la extensión id-kp-timestamp.

A.1.4 Validación de firmas XAdES Baseline

Integr@ permite la validación de firmas electrónicas XAdES Baseline, desde su implementación propia de la interfaz ***es.gob.afirma.signature.Signer*** (***es.gob.afirma.signature.xades.XAdESBaselineSigner***) y desde la fachada de firma, (***es.gob.afirma.integraFacade.IntegraFacade***) Así, se definen las siguientes tareas de validación a nivel general:

- ✓ **Validación de la Integridad:** Se comprobará que el documento XML posee al menos una firma, y que los datos firmados incluídos en la misma son acordes respecto al modo en que se ha realizado la firma (*detached*, *enveloped*, o *enveloping*).

Para cada uno de los firmantes contenidos en el documento XML firmado, se definen las siguientes tareas de validación:

- ✓ **Validación del Núcleo de Firma:** Se realizarán las siguientes verificaciones:

  - Se comprobará que el firmante verifica la firma.

  - Se verificará que el elemento QualifyingProperties está presente y se encuentra correctamente formado.

  - Se comprobará que el elemento SignedSignatureProperties tiene una estructura correcta.

  - Se verificará que se incluye, al menos, un elemento DataObjectFormat, y por cada uno de ellos, que está correctamente formado y que tiene asociada una referencia que no apunta hacia el elemento SignedProperties.

- Se comprobará que existe una referencia que apunta al elemento SignedProperties.

✓ **Validación de la Información de Clave Pública:** Se verificará que se incluye el elemento SigningCertificate y que dentro del elemento KeyInfo se incluye el certificado firmante.

✓ **Validación del Instante de Firma:** Se comprobará que se incluye el elemento firmado SigningTime, que está correctamente formado y que contiene una fecha anterior a la fecha de validación. Esta fecha de validación será la fecha del sello de tiempo menos reciente contenido en un elemento no firmado SignatureTimeStamp, si dicho elemento se incluye. En caso contrario, la fecha de validación será la fecha actual.

✓ **Validación de la Política de Firma:** Si se incluye el elemento firmado SignaturePolicyIdentifier se comprobará si el identificador (URN o URI) de la política de firma definida en dicho elemento coincide con el identificador de la política de firma definida para firmas XML en el fichero **integra.properties**, en cuyo caso, se comprobará que los datos de la firma y del firmante concreto son válidos respecto a las propiedades definidas en dicho fichero.

✓ **Validación del Certificado Firmante:** Se comprobará el estado del certificado firmante respecto al método de validación definido para el mismo en el fichero **integra.properties**. La fecha de validación será la fecha del sello de tiempo menos reciente contenido en un elemento no firmado SignatureTimeStamp, si dicho elemento se incluye. En caso contrario, la fecha de validación será la fecha actual.

✓ **Validación de los Elementos SignatureTimeStamp:** Si el firmante posee elementos SignatureTimeStamp se comprobará que todos ellos poseen una estructura correcta y que los sellos de tiempo que contienen están bien formados. Respecto a cada sello de tiempo se definen las siguientes tareas de validación:

- Validación de la Firma del Sello de Tiempo: Se comprobará que la estructura del sello de tiempo así como su núcleo de firma son correctos.

- Validación de la Integridad del Sello de Tiempo: Se comprobará que los datos sellados son correctos.

- Validación del Certificado Firmante: Se comprobará el estado del certificado firmante del sello de tiempo respecto a la fecha de generación del siguiente sello de tiempo, utilizando el método de validación definido para los certificados firmantes en el fichero **integra.properties**. Cuando se esté procesando el certificado firmante del sello de tiempo más reciente (y por lo tanto el último) se utilizará como fecha de validación la fecha actual. Además, se verificará que el certificado posee la extensión id-kp-timestamp.

A.1.5 Validación de Documentos PDF Firmados

Integr@ permite la validación de documentos PDF firmados que contengan diccionarios de firma y diccionarios de sello de tiempo. La validación puede realizarse desde las implementaciones de la interfaz ***es.gob.afirma.signature.Signer*** asociadas con firmas PAdES y PAdES Baseline (***es.gob.afirma.signature.pades.PadesSigner*** y ***es.gob.afirma.signature.pades.PAdESBaselineSigner***) y desde la fachada de firma, (***es.gob.afirma.integraFacade.IntegraFacade***) Así, se definen las siguientes tareas de validación a nivel general:

- ✓ **Validación de la Integridad:** Se comprobará que el documento PDF posee al menos una firma, y que el nivel de certificación del documento PDF es correcto. Esto es, que no existe ninguna revisión (diccionario de firma) que haya sido añadida al documento PDF después de que la última revisión haya sido definida como *Certified*.

Para cada uno de los diccionarios de firma contenidos en el documento PDF se determinará su formato de firma PAdES y se aplicarán las tareas de validación correspondientes:

- Si el diccionario de firma tiene asociado el formato PAdES-Basic se validará según el apartado A.1.6.
- Si el diccionario de firma tiene asociado el formato PAdES-BES se validará según el apartado A.1.7.
- Si el diccionario de firma tiene asociado el formato PAdES-EPES se validará según el apartado A.1.8.
- Si el diccionario de firma tiene asociado el formato PAdES B-Level se validará según el apartado A.1.9.
- Si el diccionario de firma tiene asociado el formato PAdES T-Level o superior (PAdES LT-Level o PAdES LTA-Level) se validará según el apartado A.1.10.

Cada uno de los diccionarios de sello de tiempo contenidos en el documento PDF se validará según el apartado A.1.11.

A.1.6 Validación de Diccionarios de Firma con formato PAdES-Basic

La validación de un diccionario de firma con formato PAdES-Basic se divide en las siguientes tareas:

- ✓ **Validación Estructural PDF:** Contemplará las siguientes verificaciones:
  - La clave /Contents del diccionario de firma deberá estar presente y su contenido corresponderse con una firma CMS.
  - La clave /ByteRange del diccionario de firma deberá estar presente y su contenido corresponderse con el resumen de la firma CMS.

- La clave /SubFilter del diccionario de firma deberá estar presente y su contenido corresponderse con el valor “adbe.pkcs7.detached” o “adbe.pkcs7.sha1”. En el caso de que el valor sea “adbe.pkcs7.detached” se comprobará que la firma CMS contenida es explícita y no incluye los datos firmados. En el caso de que el valor sea “adbe.pkcs7.sha1” se comprobará que el algoritmo de firma utilizado es SHA-1.

✓ **Validación del Núcleo de Firma:** Se comprobará que el primer firmante de la firma CMS contenida en el diccionario de firma verifica la propia firma CMS.

✓ **Validación de la Información de Clave Pública:** Si el primer firmante de la firma CMS posee el atributo firmado serial-number se comprobará que éste coincide con el número de serie del certificado firmante.

✓ **Validación del Instante de Firma:** Si el primer firmante de la firma CMS contenida en el diccionario de firma incluye el atributo firmado signing-time se comprobará que dicho atributo está bien formado y que la fecha contenida en el mismo es anterior a la fecha de validación. Igualmente, si el diccionario de firma incluye la clave /M validaremos que dicho campo posee un formato correcto según [PDF_Reference], sección 3.8.3 (Dates), así como que la fecha contenida no sea futura respecto a la fecha de validación. La fecha de validación en ambos casos será la fecha de generación del sello de tiempo menos reciente contenido en un atributo no firmado signature-time-stamp del primer firmante de la firma CMS contenida en el diccionario de firma. Si dicho atributo no se incluye, la fecha de validación será la fecha de generación del sello de tiempo menos reciente contenido en los diccionarios de sello de tiempo que incluyese el documento PDF con número de revisión posterior a la del diccionario de firma que se está validando. Si no se incluye ningún diccionario de sello de tiempo posterior al diccionario de firma que se está validando la fecha de validación será la fecha actual.

✓ **Validación del Certificado Firmante:** Se comprobará el estado del certificado del primer firmante de la firma CMS contenida en el diccionario de firma respecto a la fecha de validación utilizando el método de validación definido para el mismo en el fichero **integra.properties**. La fecha de validación en ambos casos será la fecha de generación del sello de tiempo menos reciente contenido en un atributo no firmado signature-time-stamp del primer firmante de la firma CMS contenida en el diccionario de firma. Si dicho atributo no se incluye, la fecha de validación será la fecha de generación del sello de tiempo menos reciente contenido en los diccionarios de sello de tiempo que incluyese el documento PDF con número de revisión posterior a la del diccionario de firma que se está validando. Si no se incluye ningún diccionario de sello de tiempo posterior al diccionario de firma que se está validando la fecha de validación será la fecha actual.

✓ **Validación de los Atributos signature-time-stamp:** Si el primer firmante de la firma CMS contenida en el diccionario de firma posee atributos signature-time-stamp se comprobará que todos ellos poseen una estructura correcta y que los sellos de tiempo que contienen están bien formados. Respecto a cada sello de tiempo se definen las siguientes tareas de validación:

- **Validación de la Firma del Sello de Tiempo:** Se comprobará que la estructura del sello de tiempo así como su núcleo de firma son correctos.

- **Validación de la Integridad del Sello de Tiempo:** Se comprobará que los datos sellados son correctos.

- **Validación del Certificado Firmante:** Se comprobará el estado del certificado firmante del sello de tiempo respecto a la fecha de generación del siguiente sello de tiempo, utilizando el método de validación definido para los certificados firmantes en el fichero **integra.properties**. Cuando se esté procesando el certificado firmante del sello de tiempo más reciente (y por lo tanto el último) se utilizará como fecha de validación la fecha de generación del sello de tiempo menos reciente contenido en los diccionarios de sello de tiempo con número de revisión posterior al diccionario de firma que se está validando que incluyese el documento PDF. Si no se incluye ningún diccionario de sello de tiempo la fecha de validación será la fecha actual. Además, se verificará que el certificado firmante del sello de tiempo posee la extensión id-kp-timestamp.

A.1.7 Validación de Diccionarios de Firma con formato PAdES-BES

La validación de un diccionario de firma con formato PAdES-BES se divide en las siguientes tareas:

✓ **Validación Estructural PDF:** Contemplará las siguientes verificaciones:

- La clave /Contents del diccionario de firma deberá estar presente y su contenido corresponderse con una firma CAdES.

- La clave /ByteRange del diccionario de firma deberá estar presente y su contenido corresponderse con el resumen de la firma CAdES.

- La clave /SubFilter del diccionario de firma deberá estar presente y su contenido corresponderse con el valor “ETSI.CAdES.detached”.

- El primer firmante de la firma CAdES que contiene el diccionario de firma deberá no tener el atributo no firmado counter-signature.

- El primer firmante de la firma CAdES que contiene el diccionario de firma deberá no tener el atributo no firmado content-reference.

- El primer firmante de la firma CAdES que contiene el diccionario de firma deberá no tener el atributo firmado content-identifier.

- El primer firmante de la firma CAdES que contiene el diccionario de firma deberá no tener el atributo firmado commitment-type-indication.

- El primer firmante de la firma CAdES que contiene el diccionario de firma deberá no tener el atributo firmado signer-location.

- El primer firmante de la firma CAdES que contiene el diccionario de firma deberá no tener el atributo firmado signing-time.

- El primer firmante de la firma CAdES que contiene el diccionario de firma deberá no tener el atributo firmado content-hints.

✓ **Validación del Núcleo de Firma:** Se comprobará que el primer firmante de la firma CAdES contenida en el diccionario de firma verifica la propia firma CAdES.

✓ **Validación de la Información de Clave Pública:** Se comprobará que el primer firmante de la firma CAdES contenida en el diccionario de firma incluye el atributo firmado signing-certificate o el atributo firmado signing-certificate-v2, y que dicho atributo identifica al certificado del firmante. Además, en el caso de que el atributo que incluya de los dos sea signing-certificate se comprobará que el algoritmo de firma utilizado ha sido SHA-1.

✓ **Validación del Instante de Firma:** Si el diccionario de firma incluye la clave /M validaremos que dicho campo posee un formato correcto según [PDF_Reference], sección 3.8.3 (Dates), así como que la fecha contenida no sea futura respecto a la fecha de validación. La fecha de validación en ambos casos será la fecha de generación del sello de tiempo menos reciente contenido en un atributo no firmado signature-time-stamp del primer firmante de la firma CAdES contenida en el diccionario de firma. Si dicho atributo no se incluye, la fecha de validación será la fecha de generación del sello de tiempo menos reciente contenido en los diccionarios de sello de tiempo que incluyese el documento PDF con número de revisión posterior a la del diccionario de firma que se está validando. Si no se incluye ningún diccionario de sello de tiempo posterior al diccionario de firma que se está validando la fecha de validación será la fecha actual.

✓ **Validación del Certificado Firmante:** Se comprobará el estado del certificado del primer firmante de la firma CAdES contenida en el diccionario de firma respecto a la fecha de validación utilizando el método de validación definido para el mismo en el fichero **integra.properties**. La fecha de validación en ambos casos será la fecha de generación del sello de tiempo menos reciente contenido en un atributo no firmado signature-time-stamp del primer firmante de la firma CAdES contenida en el diccionario de firma. Si dicho atributo no se incluye, la fecha de validación será la fecha de generación del sello de tiempo menos reciente contenido en los diccionarios de sello de tiempo que incluyese el documento PDF con número de revisión posterior a la del diccionario de firma que se está validando. Si no se incluye ningún diccionario de sello de tiempo posterior al diccionario de firma que se está validando la fecha de validación será la fecha actual.

✓ **Validación de los Atributos signature-time-stamp:** Si el primer firmante de la firma CAdES contenida en el diccionario de firma posee atributos signature-time-stamp se comprobará que todos ellos poseen una estructura correcta y que los sellos de tiempo que contienen están bien formados. Respecto a cada sello de tiempo se definen las siguientes tareas de validación:

- **Validación de la Firma del Sello de Tiempo:** Se comprobará que la estructura del sello de tiempo así como su núcleo de firma son correctos.

- **Validación de la Integridad del Sello de Tiempo:** Se comprobará que los datos sellados son correctos.

- **Validación del Certificado Firmante:** Se comprobará el estado del certificado firmante del sello de tiempo respecto a la fecha de generación del siguiente sello de tiempo, utilizando el método de validación definido para los certificados firmantes en el fichero **integra.properties**. Cuando se esté procesando el certificado firmante del sello de tiempo más reciente (y por lo tanto el último) se utilizará como fecha de validación la fecha de generación del sello de tiempo menos reciente contenido en los diccionarios de sello de tiempo con número de revisión posterior al diccionario de firma que se está validando que incluyese el documento PDF. Si no se incluye ningún diccionario de sello de tiempo la fecha de validación será la fecha actual. Además, se verificará que el certificado firmante del sello de tiempo posee la extensión id-kp-timestamp.

A.1.8 Validación de Diccionarios de Firma con formato PAdES-EPES

La validación de un diccionario de firma con formato PAdES-EPES se divide en las siguientes tareas:

✓ **Validación Estructural PDF:** Contemplará las siguientes verificaciones:

- La clave /Contents del diccionario de firma deberá estar presente y su contenido corresponderse con una firma CAdES.

- La clave /ByteRange del diccionario de firma deberá estar presente y su valor corresponderse con el resumen de la firma CAdES.

- La clave /SubFilter del diccionario de firma deberá estar presente y su valor corresponderse con “ETSI.CAdES.detached”.

- El primer firmante de la firma CAdES contenida en el diccionario de firma no deberá contener el atributo no firmado counter-signature.

- El primer firmante de la firma CAdES contenida en el diccionario de firma no deberá contener el atributo no firmado content-reference.

- El primer firmante de la firma CAdES contenida en el diccionario de firma no deberá contener el atributo firmado content-identifier.

- La clave /Reason del diccionario de firma no deberá estar presente.

- El primer firmante de la firma CAdES contenida en el diccionario de firma no deberá contener el atributo firmado signer-location.

- El primer firmante de la firma CAdES contenida en el diccionario de firma no deberá contener el atributo firmado signing-time.

- El primer firmante de la firma CAdES contenida en el diccionario de firma no deberá contener el atributo firmado content-hints.

✓ **Validación del Núcleo de Firma:** Se comprobará que el primer firmante de la firma CAdES contenida en el diccionario de firma verifica la propia firma CAdES.

✓ **Validación de la Información de Clave Pública:** Se comprobará que el primer firmante de la firma CAdES contenida en el diccionario de firma incluye el atributo firmado signing-certificate o el atributo firmado signing-certificate-v2, y que dicho atributo identifica al certificado del firmante. Además, en el caso de que el atributo que incluya de los dos sea signing-certificate se comprobará que el algoritmo de firma utilizado ha sido SHA-1.

✓ **Validación del Instante de Firma:** Si el diccionario de firma incluye la clave /M validaremos que dicho campo posee un formato correcto según [PDF_Reference], sección 3.8.3 (Dates), así como que la fecha contenida no sea futura respecto a la fecha de validación. La fecha de validación en ambos casos será la fecha de generación del sello de tiempo menos reciente contenido en un atributo no firmado signature-time-stamp del primer firmante de la firma CAdES contenida en el diccionario de firma. Si dicho atributo no se incluye, la fecha de validación será la fecha de generación del sello de tiempo menos reciente contenido en los diccionarios de sello de tiempo que incluyese el documento PDF con número de revisión posterior a la del diccionario de firma que se está validando. Si no se incluye ningún diccionario de sello de tiempo posterior al diccionario de firma que se está validando la fecha de validación será la fecha actual.

✓ **Validación de la Política de Firma:** Se comprobará si el OID de la política de firma definida en el atributo firmado signature-policy-identifier para el primer firmante de la firma CAdES contenida en el diccionario de firma coincide con el OID de la política de firma definida para firmas PDF en el fichero **integra.properties**, en cuyo caso, se comprobará que los datos de la firma y del firmante concreto son válidos respecto a las propiedades definidas en dicho fichero.

✓ **Validación del Certificado Firmante:** Se comprobará el estado del certificado del primer firmante de la firma CAdES contenida en el diccionario de firma respecto a la fecha de validación utilizando el método de validación definido para el mismo en el fichero **integra.properties**. La fecha de validación en ambos casos será la fecha de generación del sello de tiempo menos reciente contenido en un atributo no firmado signature-time-stamp del primer firmante de la firma CAdES contenida en el diccionario de firma. Si dicho atributo no se incluye, la fecha de validación será la fecha de generación del sello de tiempo menos reciente contenido en los diccionarios de sello de tiempo que incluyese el documento PDF con número de revisión posterior a la del diccionario de firma que se está validando. Si no se incluye ningún diccionario de sello de tiempo posterior al diccionario de firma que se está validando la fecha de validación será la fecha actual.

- ✓ **Validación de los Atributos signature-time-stamp:** Si el primer firmante de la firma CAdES contenida en el diccionario de firma posee atributos signature-time-stamp se comprobará que todos ellos poseen una estructura correcta y que los sellos de tiempo que contienen están bien formados. Respecto a cada sello de tiempo se definen las siguientes tareas de validación:

  - **Validación de la Firma del Sello de Tiempo:** Se comprobará que la estructura del sello de tiempo así como su núcleo de firma son correctos.

  - **Validación de la Integridad del Sello de Tiempo:** Se comprobará que los datos sellados son correctos.

  - **Validación del Certificado Firmante:** Se comprobará el estado del certificado firmante del sello de tiempo respecto a la fecha de generación del siguiente sello de tiempo, utilizando el método de validación definido para los certificados firmantes en el fichero **integra.properties**. Cuando se esté procesando el certificado firmante del sello de tiempo más reciente (y por lo tanto el último) se utilizará como fecha de validación la fecha de generación del sello de tiempo menos reciente contenido en los diccionarios de sello de tiempo con número de revisión posterior al diccionario de firma que se está validando que incluyese el documento PDF. Si no se incluye ningún diccionario de sello de tiempo la fecha de validación será la fecha actual. Además, se verificará que el certificado firmante del sello de tiempo posee la extensión id-kp-timestamp.

A.1.9 Validación de Diccionarios de Firma con formato PAdES B-Level

La validación de un diccionario de firma con formato PAdES B-Level se divide en las siguientes tareas:

- ✓ **Validación Estructural PDF:** Contemplará las siguientes verificaciones:

  - La clave /Contents del diccionario de firma deberá estar presente y su contenido corresponderse con una firma CAdES.

  - La clave /ByteRange del diccionario de firma deberá estar presente y su contenido corresponderse con el resumen de la firma CAdES.

  - La clave /SubFilter del diccionario de firma deberá estar presente y su valor corresponderse con “ETSI.CAdES.detached”.

  - El primer firmante de la firma CAdES contenida en el diccionario de firma no deberá contener el atributo firmado signing-time.

  - El primer firmante de la firma CAdES contenida en el diccionario de firma no deberá contener el atributo no firmado counter-signature.

- El primer firmante de la firma CAdES contenida en el diccionario de firma no deberá contener el atributo no firmado content-reference.
- El primer firmante de la firma CAdES contenida en el diccionario de firma no deberá contener el atributo firmado content-identifier.
- El primer firmante de la firma CAdES contenida en el diccionario de firma no deberá contener el atributo firmado content-hints.
- Si el primer firmante de la firma CAdES contenida en el diccionario de firma posee el atributo firmado signature-policy-id entonces no deberá poseer el atributo firmado commitment-type-indication.
- El primer firmante de la firma CAdES contenida en el diccionario de firma no deberá contener el atributo firmado signer-location.
- Si el primer firmante de la firma CAdES contienida en el diccionario de firma posee el atributo firmado content-type y éste tiene el valor "id-data".
- La clave /Cert del diccionario de firma no deberá estar presente.

✓ **Validación del Núcleo de Firma:** Se comprobará que el primer firmante de la firma CAdES contenida en el diccionario de firma verifica la propia firma CAdES y que dicha firma es explícita.

✓ **Validación de la Información de Clave Pública:** Se comprobará que el primer firmante de la firma CAdES contenida en el diccionario de firma incluye el atributo firmado signing-certificate o el atributo firmado signing-certificate-v2, y que dicho atributo identifica al certificado del firmante. Además, en el caso de que el atributo que incluya de los dos sea signing-certificate se comprobará que el algoritmo de firma utilizado ha sido SHA-1.

✓ **Validación de la Política de Firma:** Si el primer firmante de la firma CAdES contenida en el diccionario de firma incluye el atributo firmado signature-policy-identifier se comprobará si el OID de la política de firma definida en dicho atributo coincide con el OID de la política de firma definida para firmas PDF en el fichero **integra.properties**, en cuyo caso, se comprobará que los datos de la firma y del firmante concreto son válidos respecto a las propiedades definidas en dicho fichero.

✓ **Validación del Instante de Firma:** Se comprobará que el diccionario de firma incluye la clave /M y que dicho campo posee un formato correcto según [PDF_Reference], sección 3.8.3 (Dates), así como que la fecha contenida no sea futura respecto a la fecha de validación. La fecha de validación será la fecha de generación del sello de tiempo menos reciente contenido en los diccionarios de sello de tiempo que incluyese el documento PDF con número de revisión posterior a la del diccionario de firma que se está validando. Si no se incluye ningún diccionario de sello de tiempo posterior al diccionario de firma que se está validando la fecha de validación será la fecha actual.

- ✓ **Validación del Certificado Firmante:** Se comprobará el estado del certificado del primer firmante de la firma CAdES contenida en el diccionario de firma respecto a la fecha de validación utilizando el método de validación definido para el mismo en el fichero **integra.properties**. La fecha de validación será la fecha de generación del sello de tiempo menos reciente contenido en los diccionarios de sello de tiempo que incluyese el documento PDF con número de revisión posterior a la del diccionario de firma que se está validando. Si no se incluye ningún diccionario de sello de tiempo posterior al diccionario de firma que se está validando la fecha de validación será la fecha actual.

A.1.10 Validación de Diccionarios de Firma con formato igual o superior a PAdES T-Level

La validación de un diccionario de firma con formato igual o superior a PAdES T-Level se divide en las siguientes tareas:

- ✓ **Validación Estructural PDF:** Contemplará las siguientes verificaciones:
  - La clave /Contents del diccionario de firma deberá estar presente y su contenido corresponderse con una firma CAdES.
  - La clave /ByteRange del diccionario de firma deberá estar presente y su contenido corresponderse con el resumen de la firma CAdES.
  - La clave /SubFilter del diccionario de firma deberá estar presente y su valor corresponderse con "ETSI.CAdES.detached".
  - El primer firmante de la firma CAdES contenida en el diccionario de firma no deberá contener el atributo firmado signing-time.
  - El primer firmante de la firma CAdES contenida en el diccionario de firma no deberá contener el atributo no firmado counter-signature.
  - El primer firmante de la firma CAdES contenida en el diccionario de firma no deberá contener el atributo no firmado content-reference.
  - El primer firmante de la firma CAdES contenida en el diccionario de firma no deberá contener el atributo firmado content-identifier.
  - El primer firmante de la firma CAdES contenida en el diccionario de firma no deberá contener el atributo firmado content-hints.
  - Si el primer firmante de la firma CAdES contenida en el diccionario de firma contiene el atributo firmado signature-policy-id entonces no deberá incluir el atributo firmado commitment-type-indication.
  - El primer firmante de la firma CAdES contenida en el diccionario de firma no deberá contener el atributo firmado signer-location.

- El primer firmante de la firma CAdES contenida en el diccionario de firma deberá contener el atributo firmado content-type y el valor de éste corresponderse con "id-data".

- La clave /Cert del diccionario de firma no deberá estar presente.

✓ **Validación del Núcleo de Firma:** Se comprobará que el primer firmante de la firma CAdES contenida en el diccionario de firma verifica la propia firma CAdES y que dicha firma es explícita.

✓ **Validación de la Información de Clave Pública:** Se comprobará que el primer firmante de la firma CAdES contenida en el diccionario de firma incluye el atributo firmado signing-certificate o el atributo firmado signing-certificate-v2, y que dicho atributo identifica al certificado del firmante. Además, en el caso de que el atributo que incluya de los dos sea signing-certificate se comprobará que el algoritmo de firma utilizado ha sido SHA-1.

✓ **Validación de la Política de Firma:** Si el primer firmante de la firma CAdES contenida en el diccionario de firma incluye el atributo firmado signature-policy-identifier se comprobará si el OID de la política de firma definida en dicho atributo coincide con el OID de la política de firma definida para firmas PDF en el fichero **integra.properties**, en cuyo caso, se comprobará que los datos de la firma y del firmante concreto son válidos respecto a las propiedades definidas en dicho fichero.

✓ **Validación del Instante de Firma:** Se comprobará que el diccionario de firma incluye la clave /M y que dicho campo posee un formato correcto según [PDF_Reference], sección 3.8.3 (Dates), así como que la fecha contenida no sea futura respecto a la fecha de validación. La fecha de validación será la fecha de generación del sello de tiempo menos reciente contenido en un atributo no firmado signature-time-stamp del primer firmante de la firma CAdES contenida en el diccionario de firma.

✓ **Validación del Certificado Firmante:** Se comprobará el estado del certificado del primer firmante de la firma CAdES contenida en el diccionario de firma respecto a la fecha de validación utilizando el método de validación definido para el mismo en el fichero **integra.properties**. La fecha de validación será la fecha de generación del sello de tiempo menos reciente contenido en un atributo no firmado signature-time-stamp del primer firmante de la firma CAdES contenida en el diccionario de firma.

✓ **Validación de los Atributos signature-time-stamp:** Si el primer firmante de la firma CAdES contenida en el diccionario de firma posee atributos signature-time-stamp se comprobará que todos ellos poseen una estructura correcta y que los sellos de tiempo que contienen están bien formados. Respecto a cada sello de tiempo se definen las siguientes tareas de validación:

- **Validación de la Firma del Sello de Tiempo:** Se comprobará que la estructura del sello de tiempo así como su núcleo de firma son correctos.

- **Validación de la Integridad del Sello de Tiempo:** Se comprobará que los datos sellados son correctos.

- **Validación del Certificado Firmante:** Se comprobará el estado del certificado firmante del sello de tiempo respecto a la fecha de generación del siguiente sello de tiempo, utilizando el método de validación definido para los certificados firmantes en el fichero **integra.properties**. Cuando se esté procesando el certificado firmante del sello de tiempo más reciente (y por lo tanto el último) se utilizará como fecha de validación la fecha de generación del sello de tiempo menos reciente contenido en los diccionarios de sello de tiempo con número de revisión posterior al del diccionario de firma que se está validando que incluyese el documento PDF. Si no se incluye ningún diccionario de sello de tiempo posterior la fecha de validación será la fecha actual. Además, se verificará que el certificado firmante del sello de tiempo posee la extensión id-kp-timestamp.

A.1.11 Validación de Diccionarios de Sello de Tiempo

La validación de un diccionario de sello de tiempo se divide en las siguientes tareas:

✓ **Validación Estructural PDF:** Contemplará las siguientes verificaciones:

- La clave /Contents del diccionario de sello de tiempo deberá estar presente y corresponderse con un sello de tiempo ASN.1.

- La clave /ByteRange del diccionario de sello de tiempo deberá estar presente y su valor corresponderse con el valor del atributo message-imprint del sello de tiempo.

- La clave /SubFilter del diccionario de sello de tiempo deberá estar presente presente y su valor corresponderse con “ETSI.RFC3161”.

- La clave /Cert del diccionario de sello de tiempo no deberá estar presente.

- La clave /Reference del diccionario de sello de tiempo no deberá estar presente.

- La clave /Changes del diccionario de sello de tiempo no deberá estar presente.

- La clave /R del diccionario de sello de tiempo no deberá estar presente.

- La clave /Prop_AuthTime del diccionario de sello de tiempo no deberá estar presente.

- La clave /Prop_AuthType del diccionario de sello de tiempo no deberá estar presente.

- La clave /V del diccionario de sello de tiempo deberá estar presente y su valor corresponderse con el valor 0.

- ✓ **Validación del Instante de Firma:** La fecha de generación del sello de tiempo contenido en el diccionario de sello de tiempo será anterior a la fecha de validación. La fecha de validación se corresponderá con la fecha de generación del sello de tiempo contenido en el diccionario de sello de tiempo incluído en el documento PDF como la revisión inmediatamente superior a la revisión del diccionario de sello de tiempo que se está procesando. En caso de que no existan ningún diccionario de sello de tiempo con número de revisión mayor la fecha de validación se corresponderá con la fecha actual.

- ✓ **Validación del Núcleo de Firma:** Se comprobará que el sello de tiempo contenido en el diccionario de sello de tiempo está bien formado y que sus datos sellados se corresponden con los que deberían haber sellado.

- ✓ **Validación del Certificado Firmante:** Se comprobará el estado del certificado del primer firmante del sello de tiempo contenido en el diccionario de sello de tiempo respecto a la fecha de validación utilizando el método de validación definido para los certificados firmantes en el fichero **integra.properties**. La fecha de validación se corresponderá con la fecha de generación del sello de tiempo contenido en el diccionario de sello de tiempo incluído en el documento PDF como la revisión inmediatamente superior a la revisión del diccionario de sello de tiempo que estamos procesando. En caso de que no existan ningún diccionario de sello de tiempo con número de revisión mayor la fecha de validación se corresponderá con la fecha actual.

A.1.12 Validación de Firmas ASiC-S Baseline

Integr@ permite la validación de firmas ASiC-S Baseline que contengan una firma CAdES Baseline o una firma XAdES Baseline. No está soportada la validación de firmas ASiC-S Baseline que contengan únicamente un sello de tiempo. La validación puede realizarse desde la implementación de la interfaz ***es.gob.afirma.signature.Signer*** asociada con firmas ASiC-S Baseline (***es.gob.afirma.signature.asic.ASiCSBaselineSigner***) y desde la fachada de firma, (***es.gob.afirma.integraFacade.IntegraFacade***) Así, se definen las siguientes tareas de validación a nivel general:

- ✓ **Validación de la Integridad:** Se llevarán a cabo las siguientes verificaciones:

  - Los 4 primeros octetos del fichero ZIP deberán tener el valor '504B0304' en hexadecimal.

  - El fichero ZIP deberá contener al menos dos elementos: El fichero que se corresponde con los datos firmados, y una firma (ASN.1 o XML) o un sello de tiempo ASN.1.

  - Si dentro del fichero ZIP se incluye un fichero “mimetype” se comprobará que el valor indicado en dicho fichero es el asociado a firmas ASiC-S, esto es, 'application/vnd.etsi.asic-s+zip'. Si no posee dicho valor, deberá tener el mimetype asociado al fichero firmado.

- Si dentro del fichero ZIP se incluye una firma ASN.1 como un fichero con nombre "signature.p7s" se comprobará que la firma indicada es de tipo CAdES, posee al menos un firmante, y que los datos firmados se corresponden con el fichero incluído dentro de la firma ASiC-S Baseline como datos firmados.

- Si dentro del fichero ZIP se incluye un documento XML como un fichero con nombre "signature.xml" Se comprobará que dicho documento contiene al menos una firma, que el primero elemento ds:Signature posee como elemento padre asic:XAdESSignatures, y que todas las firmas contenidas en el documento XML son *detached*.

En lo que se refiere a la validación de los firmantes incluidos en la firma contenida dentro de la propia firma ASiC-S, si la firma es ASN.1 se validarán según el apartado XXXX, mientras que si lo que contiene es un documento XML firmado se validarán según el apartado YYYY.

A.1.13 Validación de Firma ASN.1 Contenida en Firma ASiC-S Baseline

Para cada uno de los firmantes que componen la firma ASN.1 contenida en una firma ASiC-S se ejecutan las siguientes tareas de validación:

- ✓ **Validación del Núcleo de Firma:** Se comprobará que el firmante verifica la firma y que incluye el atributo firmado content-type y que éste posee el valor "id-data".

- ✓ **Validación de la Información de Clave Pública:** Se comprobará que el firmante incluye el atributo firmado signing-certificate o el atributo firmado signing-certificate-v2, y que dicho atributo identifica al certificado del firmante. En el caso de que el atributo que incluya de los dos sea signing-certificate se comprobará que el algoritmo de firma utilizado ha sido SHA-1. Además, se comprobará que el elemento SignedData.certificates incluye, al menos, el certificado firmante.

- ✓ **Validación del Instante de Firma:** Se comprobará que el firmante incluye el atributo firmado signing-time, que dicho atributo está bien formado y que la fecha contenida en el mismo es anterior a la fecha de validación. Esta fecha de validación será la fecha del sello de tiempo menos reciente contenido en un atributo no firmado signature-time-stamp, si dicho atributo se incluye. En caso contrario, la fecha de validación será la fecha actual.

- ✓ **Validación de la Política de Firma:** Si el firmante incluye el atributo firmado signature-policy-identifier se comprobará si el OID de la política de firma definida en dicho atributo coincide con el OID de la política de firma definida para firmas ASN.1 en el fichero **integra.properties**, en cuyo caso, se comprobará que los datos de la firma y del firmante concreto son válidos respecto a las propiedades definidas en dicho fichero.

- ✓ **Validación del Certificado Firmante:** Se comprobará el estado del certificado firmante respecto a la fecha de validación respecto del método de validación definido para el mismo en el fichero **integra.properties**. La fecha de validación será la fecha del sello de tiempo menos

reciente contenido en un atributo no firmado signature-time-stamp, si dicho atributo se incluye. En caso contrario, la fecha de validación será la fecha actual.

- ✓ **Validación de los Atributos signature-time-stamp:** Si el firmante posee atributos signature-time-stamp se comprobará que todos ellos poseen una estructura correcta y que los sellos de tiempo que contienen están bien formados. Respecto a cada sello de tiempo se definen las siguientes tareas de validación:
  - **Validación de la Firma del Sello de Tiempo:** Se comprobará que la estructura del sello de tiempo así como su núcleo de firma son correctos.
  - **Validación de la Integridad del Sello de Tiempo:** Se comprobará que los datos sellados son correctos.
  - **Validación del Certificado Firmante del Sello de Tiempo:** Se comprobará el estado del certificado firmante del sello de tiempo respecto a la fecha de generación del siguiente sello de tiempo, utilizando el método de validación definido para los certificados firmantes en el fichero **integra.properties**. Cuando se esté procesando el certificado firmante del sello de tiempo más reciente (y por lo tanto el último) se utilizará como fecha de validación la fecha actual. Además, se verificará que el certificado posee la extensión id-kp-timestamp.

A.1.14 Validación de Documento XML Firmado Contenido en Firma ASiC-S Baseline

Para cada uno de los firmantes que componen el documento XML firmado contenido en una firma ASiC-S se ejecutan las siguientes tareas de validación:

- ✓ **Validación del Núcleo de Firma:** Se realizarán las siguientes verificaciones:
  - Se comprobará que el firmante verifica la firma.
  - Se verificará que el elemento QualifyingProperties está presente y se encuentra correctamente formado.
  - Se comprobará que el elemento SignedSignatureProperties tiene una estructura correcta.
  - Se verificará que se incluye, al menos, un elemento DataObjectFormat, y por cada uno de ellos, que está correctamente formado y que tiene asociada una referencia que no apunta hacia el elemento SignedProperties.
  - Se comprobará que existe una referencia que apunta al elemento SignedProperties.
  - Se comprobará que se incluye una referencia al fichero incluído dentro de la firma ASiC-S Baseline como datos firmados.

- ✓ **Validación de la Información de Clave Pública:** Se verificará que se incluye el elemento SigningCertificate y que dentro del elemento KeyInfo se incluye el certificado firmante.

- ✓ **Validación del Instante de Firma:** Se comprobará que se incluye el elemento firmado SigningTime, que está correctamente formado y que contiene una fecha anterior a la fecha de validación. Esta fecha de validación será la fecha del sello de tiempo menos reciente contenido en un elemento no firmado SignatureTimeStamp, si dicho elemento se incluye. En caso contrario, la fecha de validación será la fecha actual.

- ✓ **Validación de la Política de Firma:** Si se incluye el elemento firmado SignaturePolicyIdentifier se comprobará si el identificador (URN o URI) de la política de firma definida en dicho elemento coincide con el identificador de la política de firma definida para firmas XML en el fichero **integra.properties**, en cuyo caso, se comprobará que los datos de la firma y del firmante concreto son válidos respecto a las propiedades definidas en dicho fichero.

- ✓ **Validación del Certificado Firmante:** Se comprobará el estado del certificado firmante respecto al método de validación definido para el mismo en el fichero **integra.properties**. La fecha de validación será la fecha del sello de tiempo menos reciente contenido en un elemento no firmado SignatureTimeStamp, si dicho elemento se incluye. En caso contrario, la fecha de validación será la fecha actual.

- ✓ **Validación de los Elementos SignatureTimeStamp:** Si el firmante posee elementos SignatureTimeStamp se comprobará que todos ellos poseen una estructura correcta y que los sellos de tiempo que contienen están bien formados. Respecto a cada sello de tiempo se definen las siguientes tareas de validación:

    - **Validación de la Firma del Sello de Tiempo:** Se comprobará que la estructura del sello de tiempo, así como su núcleo de firma son correctos.

    - **Validación de la Integridad del Sello de Tiempo:** Se comprobará que los datos sellados son correctos.

    - **Validación del Certificado Firmante:** Se comprobará el estado del certificado firmante del sello de tiempo respecto a la fecha de generación del siguiente sello de tiempo, utilizando el método de validación definido para los certificados firmantes en el fichero integra.properties. Cuando se esté procesando el certificado firmante del sello de tiempo más reciente (y por lo tanto el último) se utilizará como fecha de validación la fecha actual. Además, se verificará que el certificado posee la extensión id-kp-timestamp.

A.2 Ejemplos de Uso de los Servicios para el tipo de integración 1

En este anexo se incluyen algunos ejemplos de uso relacionados con los accesos a servicios OCSP de @Firma y RFC 3161 de TS@.

A.2.1 Configuración general.

Para este tipo de integración es necesario hacer uso de las librerías de Integr@ y librerías de terceros enumeradas en el apartado 7.1.

Por otro lado, será necesario disponer en una ruta accesible desde la aplicación en la que se usa Integr@ de los ficheros de configuración necesarios para este caso concreto. Puenden verse los ficheros de propiedades necesarios en el apartado 8.1.

Para el caso que nos ocupa damos por hecho que se encuentran desplegadas las plataformas de @Firma y TS@ con aplicaciones "afirmaTest" y "pruebasTSARFC3161SHA1" dadas de alta respectivamente y correctamente configuradas.

A continuación, se procederá a configurar el lenguaje de los mensajes de Integr@ editando el archivo de propiedades **Language.properties**:

```
LANGUAGE = es_ES
```

Con esta configuración la plataforma emitirá los mensajes en español.

Configuramos a continuación el acceso al servicio OCSP de validación de certificados de @firma añadiendo al archivo de propiedades generales **integra.properties** la configuración del acceso ocsp. Esta configuración sólo será necesaria si se va a utilizar el servicio OCSP para validación de certificados. Para el caso concreto que nos ocupa los datos serían como los siguientes.

```
OCSP_URL = http://localhost:8080/servidorOcsp/servidorOCSP
OCSP_ISSUER_KEYSTORE_PATH =D:/workspace-baseJdk1.8/Integra-parent/Integra-ocsp-
rfc3161/src/test/resources/truststoreWS.jks
OCSP_ISSUER_KEYSTORE_TYPE = JKS
OCSP_ISSUER_KEYSTORE_PASSWORD = 12345
OCSP_RESPONSE_CERTIFICATE_PATH = D:/workspace-baseJdk1.8/Integra-parent/Integra-
ocsp-rfc3161/src/test/resources/confianzaocsp.crt
OCSP_TIMEOUT = 200000
OCSP_SIGN_REQUEST = true
OCSP_REQUEST_KEYSTORE_PATH = D:/workspace-baseJdk1.8/Integra-parent/Integra-ocsp-
rfc3161/src/test/resources/SoapSigner-2012.p12
OCSP_REQUEST_KEYSTORE_TYPE = PKCS12
OCSP_REQUEST_KEYSTORE_PASSWORD = 12345
OCSP_REQUEST_PRIVATE_KEY_ALIAS = confianzaocsp
OCSP_REQUEST_PRIVATE_KEY_PASSWORD = 12345
OCSP_APP_ID = afirmaTest
OCSP_USE_GET = false
```

Si se van a realizar peticiones de sello de tiempo a TS@ mediante RFC 3161 será necesario también añadir al archivo **integra.properties** los datos generales de acceso a TS@. Para el ejemplo que nos ocupa la configuración sería la siguiente.

```
com.trustedstorePath = D:/workspace-baseJdk1.8/Integra-parent/Integra-ocsp-
rfc3161/src/test/resources/truststoreWS.jks

com.trustedstorePassword = 12345

com.serviceWSDLPath = file:/D:/workspace-baseJdk1.8/Integra-parent/Integra-ocsp-
rfc3161/src/test/resources/TimeStampWS.wsdl
```

Si se van a realizar peticiones de sello de tiempo a TS@ mediante RFC 3161 será necesario también añadir el archivo de propiedades para la conexión con TS@ para la aplicación pruebasTSARFC3161SHA1 **tsapruebasTSARFC3161SHA1.properties**. Para el ejemplo que nos ocupa la configuración sería la siguiente.

```
rfc3161.host = localhost
rfc3161.timestampPolicyOID = 1.2.3.4.5
rfc3161.applicationOID = 1.3.4.6.1.3.4.6
rfc3161.Timeout = 50000
rfc3161.shaAlgorithm = SHA-1
rfc3161.portNumber = 318
rfc3161HTTPS.portNumber = 443
rfc3161HTTPS.context = /tsamap/TspHttpServer
rfc3161HTTPS.useAuthClient = true
rfc3161HTTPS.keystorePath = D:/workspace-baseJdk1.8/Integra-parent/Integra-ocsp-
rfc3161/src/test/resources/autenticacionAfirma.p12
rfc3161HTTPS.keystoreType = PKCS12
rfc3161HTTPS.keystorePassword = 12345
rfc3161SSL.portNumber = 10318
rfc3161SSL.keystorePath =D:/workspace-baseJdk1.8/Integra-parent/Integra-ocsp-
rfc3161/src/test/resources/autenticacionAfirma.p12
rfc3161SSL.keystoreType = PKCS12
rfc3161SSL.keystorePassword = 12345
```

Finalmente, sera necesario mapear los ficheros de configuración dinámicos en **mappingFiles.properties**.

```
tsapruebasTSARFC3161SHA1 = tsapruebasTSARFC3161SHA1.properties
integra = integra.properties
```

A.2.2 Validación de Certificado por OCSP.

Se hará uso de un código como el siguiente si se desea realizar la validación de un certificado mediante OCSP.

```
OCSPEnhancedResponse ocspResponse = OCSPClient.validateCertificate("certificado de
tipo X509Certificate");
```

A.2.3 Obtención de sello de tiempo desde TS@ a través de RFC 3161.

Se hará uso de un código como el siguiente si se desea solicitar un sello de tiempo a la TS@ mediante RFC 3161. Se plantean tres modos de acceso (TCP, HTTPS, SSL).

```
String APPLICATION = "pruebasTSARFC3161SHA1";
byte[ ] file = UtilsFileSystemCommons.readFile("ficheroAfirmar.txt", true);

//Otención de sello de tiempo mediante TCP.
RFC3161TSAServiceInvoker invoker = new RFC3161TSAServiceInvoker();
byte[ ] response =
invoker.generateTimeStampToken(TSAServiceInvokerConstants.RFC3161Protocol.TCP,
APPLICATION, file);
TimeStampResponse tsp = new TimeStampResponse(response);

//Otención de sello de tiempo mediante HTTPS.
RFC3161TSAServiceInvoker invoker = new RFC3161TSAServiceInvoker();
byte[ ] response =
invoker.generateTimeStampToken(TSAServiceInvokerConstants.RFC3161Protocol.HTTPS,
APPLICATION, file);
TimeStampResponse tsp = new TimeStampResponse(response);

//Otención de sello de tiempo mediante SSL.
RFC3161TSAServiceInvoker invoker = new RFC3161TSAServiceInvoker();
byte[ ] response =
invoker.generateTimeStampToken(TSAServiceInvokerConstants.RFC3161Protocol.SSL,
APPLICATION, file);
TimeStampResponse tsp = new TimeStampResponse(response);
```

A.3 Ejemplos de Uso de los Servicios para el tipo de integración 2

En este anexo se incluyen algunos ejemplos de uso relacionados con los accesos a servicios WS y DSS de @Firma, TS@ y eVisor.

A.3.1 Configuración general.

Para este tipo de integración es necesario hacer uso de las librerías de Integr@ y librerías de terceros enumeradas en el apartado 0.

Por otro lado, será necesario disponer en una ruta accesible desde la aplicación en la que se usa Integr@ de los ficheros y carpetas de configuración necesarios para este caso concreto y que pueden encontrarse en el entregable. Puenden verse las carpetas y los ficheros de propiedades necesarios en el apartado 8.2.

Para el caso que nos ocupa damos por hecho que se encuentran desplegadas las plataformas de @Firma, TS@ y eVisor con aplicaciones “afirmaTest”, “pruebasTest” y “afirmaTestEvisor” dadas de alta respectivamente y correctamente configuradas.

A continuación, se procederá a configurar el lenguaje de los mensajes de Integr@ editando el archivo de propiedades **Language.properties**:

```
LANGUAGE = es_ES
```

Con esta configuración la plataforma emitirá los mensajes en español.

Configuramos a continuación el archivo **integra.properties** con los datos generales de acceso a TS@, eVisor y @firma. Para el ejemplo que nos ocupa la configuración sería la siguiente.

```
com.trustedstorePath = D:/workspace-baseJdk1.8/Integra-parent/Integra-ocsp-
rfc3161/src/test/resources/truststoreWS.jks
com.trustedstorepassword=12345
com.certificatesCache.use = false
com.certificatesCache.entries = 2
com.certificatesCache.lifeTime = 120
```

Configuramos a continuación los parámetros concretos de acceso a los WS de @Firma creando el archivo de propiedades **afirmaafirmaTest.properties**. Esta configuración sólo será necesaria si se va a utilizar el servicio WS de @Firma. Para el caso concreto que nos ocupa los datos serían como los siguientes.

```
secureMode=false
endPoint=localhost:8080
servicePath=afirmaws/services
callTimeout=20000
authorizationMethod=none
authorizationMethod.passwordType=clear
response.validate=false
response.certificateAlias=alias
```

Configuramos a continuación los parámetros concretos de acceso a los WS de eVisor creando el archivo de propiedades **evisorafirmaTestEVisor.properties**. Esta configuración sólo será necesaria si se va a utilizar el servicio WS de eVisor. Para el caso concreto que nos ocupa los datos serían como los siguientes.

```
secureMode=false
endPoint=localhost:8089
servicePath=eVisor-2.1/services
authorizationMethod=none
callTimeout=200000
```

Si se van a realizar peticiones de sello de tiempo a TS@ mediante DSS será necesario crear el archivo de propiedades **tsapruebasTest.properties**. Para el ejemplo que nos ocupa la configuración sería la siguiente.

```
callTimeout =20000
authorizationMethod =UserNameToken
UserNameToken.userName = user
UserNameToken.userPassword = 12345
request.symmetricKey.use =false
renewTimeStampWS.validationLevel = 0
```

A continuación se realizará la configuración del componente Transformers, configurando las rutas de las plantillas que utilizará el componente. Accedemos al fichero **transformers.properties** y configuramos las siguientes propiedades:

```
TransformersTemplatesPath = D:/workspace-baseJdk1.8/Integra-parent/Integra-
commons/src/test/resources/transformersTemplates
```

Finalmente sera necesario mapear los ficheros de configuración dinámicos en **mappingFiles.properties**.

```
tsapruebasTest = tsapruebasTest.properties
afirmaafirmaTest = afirmaafirmaTest.properties
evisorafirmaTestEVisor = evisorafirmaTestEVisor.properties
integra = integra.properties
```

A.3.2 Generación de firmas CAdES y XAdES contra el servicio DSS de @Firma.

Se hará uso de un código como el siguiente para generar firmas contra @Firma. En el ejemplo concreto se generan una firma CAdES, una XAdES detached y una XAdES enveloping

```
String APPLICATION_NAME = "afirmaTest";
String SERVER_SIGNER_NAME = "raul conde";
String signatureB64XML = null;
String signXMLEnveloping = null;
String signatureB64 = null;
String archiveIdentifier = null;

String documentB64 =
UtilsFileSystemCommons.readFileBase64Encoded("ficheroAfirmar.txt", true);

Map<String, Object> inParams = new HashMap<String, Object>();
inParams.put(DSSTagsRequest.BASE64DATA, documentB64);
inParams.put(DSSTagsRequest.KEY_SELECTOR, SERVER_SIGNER_NAME);
inParams.put(DSSTagsRequest.SIGNATURE_TYPE, SignTypesURIs.CMS);
inParams.put(DSSTagsRequest.HASH_ALGORITHM, AlgorithmTypes.SHA1);
inParams.put(DSSTagsRequest.ADDITIONAL_DOCUMENT_NAME, "ficheroAfirmar.txt");
inParams.put(DSSTagsRequest.ADDITIONAL_DOCUMENT_TYPE, "txt");

// ->CADES
inParams.put(DSSTagsRequest.SIGNATURE_TYPE, SignTypesURIs.CADES);
inParams.put(DSSTagsRequest.SIGNATURE_FORM, SignatureForm.BES);
xmlInput = TransformersFacade.getInstance().generateXml(inParams,
GeneralConstants.DSS_AFIRMA_SIGN_REQUEST, GeneralConstants.DSS_AFIRMA_SIGN_METHOD,
TransformersConstants.VERSION_10);
xmlOutput = Afirma5ServiceInvokerFacade.getInstance().invokeService(xmlInput,
GeneralConstants.DSS_AFIRMA_SIGN_REQUEST, GeneralConstants.DSS_AFIRMA_SIGN_METHOD,
APPLICATION_NAME);
propertiesResult = TransformersFacade.getInstance().parseResponse(xmlOutput,
GeneralConstants.DSS_AFIRMA_SIGN_REQUEST, GeneralConstants.DSS_AFIRMA_SIGN_METHOD,
TransformersConstants.VERSION_10);

signatureB64 =
propertiesResult.get(TransformersFacade.getInstance().getParserParameterValue("Sig
natureBase64")).toString();
```

```
// ->XADES DETACHED
inParams.put(DSSTagsRequest.SIGNATURE_TYPE, SignTypesURIs.XADES_V_1_3_2);
inParams.put(DSSTagsRequest.SIGNATURE_FORM, SignatureForm.BES);
inParams.put(DSSTagsRequest.XML_SIGNATURE_MODE, XmlSignatureModeEnum.DETACHED);
xmlInput = TransformersFacade.getInstance().generateXml(inParams,
GeneralConstants.DSS_AFIRMA_SIGN_REQUEST, GeneralConstants.DSS_AFIRMA_SIGN_METHOD,
TransformersConstants.VERSION_10);
xmlOutput = Afirma5ServiceInvokerFacade.getInstance().invokeService(xmlInput,
GeneralConstants.DSS_AFIRMA_SIGN_REQUEST, GeneralConstants.DSS_AFIRMA_SIGN_METHOD,
APPLICATION_NAME);
propertiesResult = TransformersFacade.getInstance().parseResponse(xmlOutput,
GeneralConstants.DSS_AFIRMA_SIGN_REQUEST, GeneralConstants.DSS_AFIRMA_SIGN_METHOD,
TransformersConstants.VERSION_10);
signatureB64XML =
propertiesResult.get(TransformersFacade.getInstance().getParserParameterValue("Sig
natureBase64XML")).toString();
archiveIdentifier =
propertiesResult.get(TransformersFacade.getInstance().getParserParameterValue("Arc
hiveIdentifier")).toString();


// ->XADES ENVELOPING
inParams.put(DSSTagsRequest.CLAIMED_IDENTITY, APPLICATION_NAME);

inParams.put(DSSTagsRequest.SIGNATURE_TYPE, SignTypesURIs.XADES_V_1_3_2);
inParams.put(DSSTagsRequest.SIGNATURE_FORM, SignatureForm.BES);
inParams.put(DSSTagsRequest.XML_SIGNATURE_MODE, XmlSignatureModeEnum.ENVELOPING);
xmlInput = TransformersFacade.getInstance().generateXml(inParams,
GeneralConstants.DSS_AFIRMA_SIGN_REQUEST, GeneralConstants.DSS_AFIRMA_SIGN_METHOD,
TransformersConstants.VERSION_10);
xmlOutput = Afirma5ServiceInvokerFacade.getInstance().invokeService(xmlInput,
GeneralConstants.DSS_AFIRMA_SIGN_REQUEST, GeneralConstants.DSS_AFIRMA_SIGN_METHOD,
APPLICATION_NAME);
propertiesResult = TransformersFacade.getInstance().parseResponse(xmlOutput,
GeneralConstants.DSS_AFIRMA_SIGN_REQUEST, GeneralConstants.DSS_AFIRMA_SIGN_METHOD,
TransformersConstants.VERSION_10);
signXMLEnveloping =
propertiesResult.get(TransformersFacade.getInstance().getParserParameterValue("Sig
natureXML")).toString();
```

Tras ejecutar el código anterior se dispondrá en signatureB64XML, signingXMLEnveloping, signatureB64 y archiveIdentifier el resultado de las operaciones que podrán ser utilizados para el resto de los ejemplos.

A.3.3 Generación de co-firma contra el servicio DSS de @Firma.

Se hará uso de un código como el siguiente si se desea solicitar una co-firma al servicio DSS de @Firma sobre una firma previamente realizada. El identificador de la firma previa puede ser obtenido del ejemplo anterior A.3.2.

```
String APPLICATION_NAME = "afirmaTest";
String SERVER_SIGNER_NAME = "raul conde";

Map<String, Object> inParams = new HashMap<String, Object>();
inParams.put(DSSTagsRequest.CLAIMED_IDENTITY, APPLICATION_NAME);
```

```
inParams.put(DSSTagsRequest.KEY_SELECTOR, SERVER_SIGNER_NAME);
inParams.put(DSSTagsRequest.HASH_ALGORITHM, AlgorithmTypes.SHA1);
inParams.put(DSSTagsRequest.DOCUMENT_ARCHIVE_ID, archiveIdentifier);
inParams.put(DSSTagsRequest.PARALLEL_SIGNATURE, "");

String xmlInput = TransformersFacade.getInstance().generateXml(inParams,
GeneralConstants.DSS_AFIRMA_SIGN_REQUEST, GeneralConstants.DSS_AFIRMA_SIGN_METHOD,
TransformersConstants.VERSION_10);
String xmlOutput =
Afirma5ServiceInvokerFacade.getInstance().invokeService(xmlInput,
GeneralConstants.DSS_AFIRMA_SIGN_REQUEST, GeneralConstants.DSS_AFIRMA_SIGN_METHOD,
APPLICATION_NAME);
Map<String, Object> propertiesResult =
TransformersFacade.getInstance().parseResponse(xmlOutput,
GeneralConstants.DSS_AFIRMA_SIGN_REQUEST, GeneralConstants.DSS_AFIRMA_SIGN_METHOD,
TransformersConstants.VERSION_10);
```

A.3.4 Generación de contra-firma contra el servicio DSS de @Firma.

Se hará uso de un código como el siguiente si se desea solicitar una contra-firma al servicio DSS de @Firma sobre una firma previamente realizada. El identificador de la firma previa puede ser obtenido del ejemplo anterior A.3.2.

```
String APPLICATION_NAME = "afirmaTest";
String SERVER_SIGNER_NAME = "raul conde";

inParams.put(DSSTagsRequest.CLAIMED_IDENTITY, APPLICATION_NAME);
inParams.put(DSSTagsRequest.KEY_SELECTOR, SERVER_SIGNER_NAME);
inParams.put(DSSTagsRequest.HASH_ALGORITHM, AlgorithmTypes.SHA1);
inParams.put(DSSTagsRequest.DOCUMENT_ARCHIVE_ID, archiveIdentifier);
inParams.put(DSSTagsRequest.COUNTER_SIGNATURE, "");

String xmlInput = TransformersFacade.getInstance().generateXml(inParams,
GeneralConstants.DSS_AFIRMA_SIGN_REQUEST, GeneralConstants.DSS_AFIRMA_SIGN_METHOD,
TransformersConstants.VERSION_10);
String xmlOutput =
Afirma5ServiceInvokerFacade.getInstance().invokeService(xmlInput,
GeneralConstants.DSS_AFIRMA_SIGN_REQUEST, GeneralConstants.DSS_AFIRMA_SIGN_METHOD,
APPLICATION_NAME);
Map<String, Object> propertiesResult =
TransformersFacade.getInstance().parseResponse(xmlOutput,
GeneralConstants.DSS_AFIRMA_SIGN_REQUEST, GeneralConstants.DSS_AFIRMA_SIGN_METHOD,
TransformersConstants.VERSION_10);
```

A.3.5 Validación de firmas contra el servicio DSS de @Firma.

Se hará uso de un código como el siguiente si se desea enviar una solicitud de validación al servicio DSS de @Firma sobre una firma previamente realizada. Sn el ejemplo que nos ocupa se realizarán validaciones sobre las firmas obtendias en el ejemplo anterior A.3.2.

```
String APPLICATION_NAME = "afirmaTest";
String SERVER_SIGNER_NAME = "raul conde";

inParams.put(DSSTagsRequest.INCLUDE_REVOCATION, Boolean.TRUE.toString());
```

```
inParams.put(DSSTagsRequest.REPORT_DETAIL_LEVEL, ReportDetailLevel.ALL_DETAILS);
inParams.put(DSSTagsRequest.RETURN_PROCESSING_DETAILS, "");
inParams.put(DSSTagsRequest.RETURN_READABLE_CERT_INFO, "");
inParams.put(DSSTagsRequest.ADDICIONAL_REPORT_OPT_SIGNATURE_TIMESTAMP,
"urn:afirma:dss:1.0:profile:XSS:SignatureProperty:SignatureTimeStamp");

// pruebas con firmas de tipo XML (XAdES Detached o Enveloped)
Map<String, Object> xadesParams = new HashMap<String, Object>(inParams);
xadesParams.put(DSSTagsRequest.SIGNATURE_PTR_ATR_WHICH, archiveIdentifier);
xadesParams.put(DSSTagsRequest.DOCUMENT_ATR_ID, archiveIdentifier);
xadesParams.put(DSSTagsRequest.BASE64XML, signatureB64XML);

String xmlInput = TransformersFacade.getInstance().generateXml(xadesParams,
GeneralConstants.DSS_AFIRMA_VERIFY_REQUEST,
GeneralConstants.DSS_AFIRMA_VERIFY_METHOD, TransformersConstants.VERSION_10);
String xmlOutput =
Afirma5ServiceInvokerFacade.getInstance().invokeService(xmlInput,
GeneralConstants.DSS_AFIRMA_VERIFY_REQUEST,
GeneralConstants.DSS_AFIRMA_VERIFY_METHOD, APPLICATION_NAME);
Map<String, Object> propertiesResult =
TransformersFacade.getInstance().parseResponse(xmlOutput,
GeneralConstants.DSS_AFIRMA_VERIFY_REQUEST,
GeneralConstants.DSS_AFIRMA_VERIFY_METHOD, TransformersConstants.VERSION_10);

// pruebas con firma de tipo XML (XAdES Enveloping)
xadesParams = new HashMap<String, Object>(inParams);
xadesParams.put(DSSTagsRequest.SIGNATURE_OBJECT, signXMLEnveloping);
xmlInput = TransformersFacade.getInstance().generateXml(xadesParams,
GeneralConstants.DSS_AFIRMA_VERIFY_REQUEST,
GeneralConstants.DSS_AFIRMA_VERIFY_METHOD, TransformersConstants.VERSION_10);
xmlOutput = Afirma5ServiceInvokerFacade.getInstance().invokeService(xmlInput,
GeneralConstants.DSS_AFIRMA_VERIFY_REQUEST,
GeneralConstants.DSS_AFIRMA_VERIFY_METHOD, APPLICATION_NAME);
propertiesResult = TransformersFacade.getInstance().parseResponse(xmlOutput,
GeneralConstants.DSS_AFIRMA_VERIFY_REQUEST,
GeneralConstants.DSS_AFIRMA_VERIFY_METHOD, TransformersConstants.VERSION_10);

// pruebas con firmas de tipo CAdES
Map<String, Object> cadesParams = new HashMap<String, Object>(inParams);
cadesParams.put(DSSTagsRequest.BASE64DATA,
UtilsFileSystemCommons.readFileBase64Encoded("ficheroAfirmar.txt", true));
cadesParams.put(DSSTagsRequest.SIGNATURE_BASE64, signatureB64);
cadesParams.put(DSSTagsRequest.SIGNATURE_BASE64_ATR_TYPE, SignTypesURIs.CADES);

xmlInput = TransformersFacade.getInstance().generateXml(cadesParams,
GeneralConstants.DSS_AFIRMA_VERIFY_REQUEST,
GeneralConstants.DSS_AFIRMA_VERIFY_METHOD, TransformersConstants.VERSION_10);
xmlOutput = Afirma5ServiceInvokerFacade.getInstance().invokeService(xmlInput,
GeneralConstants.DSS_AFIRMA_VERIFY_REQUEST,
GeneralConstants.DSS_AFIRMA_VERIFY_METHOD, APPLICATION_NAME);
propertiesResult = TransformersFacade.getInstance().parseResponse(xmlOutput,
GeneralConstants.DSS_AFIRMA_VERIFY_REQUEST,
GeneralConstants.DSS_AFIRMA_VERIFY_METHOD, TransformersConstants.VERSION_10);

// extraemos la información de la firma
Map<String, Object>[ ] signReports = (Map[ ])
propertiesResult.get(TransformersFacade.getInstance().getParserParameterValue("Ind
ividualSignatureReport"));
Map<String, Object> individualSignReport = signReports[0];
```

```
// comprobamos valores de la información del certificado (tipo mapa)
Map<String, String> certificateInfo = (Map<String, String>)
individualSignReport.get(TransformersFacade.getInstance().getParserParameterValue(
"ReadableCertificateInfo"));
```

A.3.6 Actualización de firmas contra el servicio DSS de @Firma.

Se hará uso de un código como el siguiente si se desea enviar una solicitud de actualización (en este caso a formato T) al servicio DSS de @Firma sobre una firma previamente realizada. En el ejemplo que nos ocupa se realizarán actualizaciones sobre las firmas obtendias en el ejemplo anterior A.3.2.

```
String APPLICATION_NAME = "afirmaTest";
String SERVER_SIGNER_NAME = "raul conde";

Map<String, Object> inParams = new HashMap<String, Object>();
inParams.put(DSSTagsRequest.CLAIMED_IDENTITY, APPLICATION_NAME);
inParams.put(DSSTagsRequest.RETURN_UPDATED_SIGNATURE_ATR_TYPE, SignatureForm.T);

// pruebas con firmas de tipo CAdES
inParams.put(DSSTagsRequest.SIGNATURE_BASE64, signatureB64);
inParams.put(DSSTagsRequest.SIGNATURE_PTR_ATR_WHICH, "1298045604559");

String xmlInput = TransformersFacade.getInstance().generateXml(inParams,
GeneralConstants.DSS_AFIRMA_VERIFY_REQUEST,
GeneralConstants.DSS_AFIRMA_VERIFY_METHOD, TransformersConstants.VERSION_10);
String xmlOutput =
Afirma5ServiceInvokerFacade.getInstance().invokeService(xmlInput,
GeneralConstants.DSS_AFIRMA_VERIFY_REQUEST,
GeneralConstants.DSS_AFIRMA_VERIFY_METHOD, APPLICATION_NAME);
Map<String, Object> propertiesResult =
TransformersFacade.getInstance().parseResponse(xmlOutput,
GeneralConstants.DSS_AFIRMA_VERIFY_REQUEST,
GeneralConstants.DSS_AFIRMA_VERIFY_METHOD, TransformersConstants.VERSION_10);

// pruebas con firmas de tipo XAdES Detached
inParams.remove(DSSTagsRequest.SIGNATURE_BASE64);
inParams.put(DSSTagsRequest.BASE64XML, signatureB64XML);
inParams.put(DSSTagsRequest.DOCUMENT_ATR_ID, archiveIdentifier);
inParams.put(DSSTagsRequest.SIGNATURE_PTR_ATR_WHICH, archiveIdentifier);

xmlInput = TransformersFacade.getInstance().generateXml(inParams,
GeneralConstants.DSS_AFIRMA_VERIFY_REQUEST,
GeneralConstants.DSS_AFIRMA_VERIFY_METHOD, TransformersConstants.VERSION_10);
xmlOutput = Afirma5ServiceInvokerFacade.getInstance().invokeService(xmlInput,
GeneralConstants.DSS_AFIRMA_VERIFY_REQUEST,
GeneralConstants.DSS_AFIRMA_VERIFY_METHOD, APPLICATION_NAME);
propertiesResult = TransformersFacade.getInstance().parseResponse(xmlOutput,
GeneralConstants.DSS_AFIRMA_VERIFY_REQUEST,
GeneralConstants.DSS_AFIRMA_VERIFY_METHOD, TransformersConstants.VERSION_10);

// pruebas con firmas de tipo XAdES Enveloping
inParams.remove(DSSTagsRequest.BASE64XML);
inParams.remove(DSSTagsRequest.DOCUMENT_ATR_ID);
inParams.remove(DSSTagsRequest.SIGNATURE_PTR_ATR_WHICH);
inParams.put(DSSTagsRequest.RETURN_UPDATED_SIGNATURE_ATR_TYPE, SignatureForm.T);
inParams.put(DSSTagsRequest.SIGNATURE_OBJECT, signXMLEnveloping);
```

```
xmlInput = TransformersFacade.getInstance().generateXml(inParams,
GeneralConstants.DSS_AFIRMA_VERIFY_REQUEST,
GeneralConstants.DSS_AFIRMA_VERIFY_METHOD, TransformersConstants.VERSION_10);
xmlOutput = Afirma5ServiceInvokerFacade.getInstance().invokeService(xmlInput,
GeneralConstants.DSS_AFIRMA_VERIFY_REQUEST,
GeneralConstants.DSS_AFIRMA_VERIFY_METHOD, APPLICATION_NAME);

propertiesResult = TransformersFacade.getInstance().parseResponse(xmlOutput,
GeneralConstants.DSS_AFIRMA_VERIFY_REQUEST,
GeneralConstants.DSS_AFIRMA_VERIFY_METHOD, TransformersConstants.VERSION_10);
String signXml =
propertiesResult.get(TransformersFacade.getInstance().getParserParameterValue("Upd
atedSignatureXML")).toString();
```

A.3.7 Validación de certificados contra el servicio DSS de @Firma.

Se hará uso de un código como el siguiente si se desea solicitar una validación de certificado al servicio DSS de @Firma.

```
String APPLICATION_NAME = "afirmaTest";
String CERTIFICATE_NAME = "confianzaocsp.crt";

Map<String, Object> inParams = new HashMap<String, Object>();

inParams.put(DSSTagsRequest.CLAIMED_IDENTITY, APPLICATION_NAME);
inParams.put(DSSTagsRequest.INCLUDE_CERTIFICATE, "true");
inParams.put(DSSTagsRequest.INCLUDE_REVOCATION, "true");
inParams.put(DSSTagsRequest.REPORT_DETAIL_LEVEL, ReportDetailLevel.ALL_DETAILS);
inParams.put(DSSTagsRequest.CHECK_CERTIFICATE_STATUS, "true");
inParams.put(DSSTagsRequest.RETURN_READABLE_CERT_INFO, "");
inParams.put(DSSTagsRequest.X509_CERTIFICATE,
UtilsFileSystemCommons.readFileBase64Encoded(CERTIFICATE_NAME, true));
String xmlInput = TransformersFacade.getInstance().generateXml(inParams,
GeneralConstants.DSS_AFIRMA_VERIFY_CERTIFICATE_REQUEST,
GeneralConstants.DSS_AFIRMA_VERIFY_METHOD, TransformersConstants.VERSION_10);
String xmlOutput =
Afirma5ServiceInvokerFacade.getInstance().invokeService(xmlInput,
GeneralConstants.DSS_AFIRMA_VERIFY_CERTIFICATE_REQUEST,
GeneralConstants.DSS_AFIRMA_VERIFY_METHOD, APPLICATION_NAME);
Map<String, Object> propertiesResult =
TransformersFacade.getInstance().parseResponse(xmlOutput,
GeneralConstants.DSS_AFIRMA_VERIFY_CERTIFICATE_REQUEST,
GeneralConstants.DSS_AFIRMA_VERIFY_METHOD, TransformersConstants.VERSION_10);

Map<String, String> certificateInfo = (Map<String, String>)
propertiesResult.get(TransformersFacade.getInstance().getParserParameterValue("Cer
tificateInfo"));
```

A.3.8 Sellado de tiempo y validación contra el servicio DSS de TS@.

Se hará uso de un código como el siguiente si se desea solicitar un sello de tiempo al servicio DSS de TS@.

Ejemplo 1. Tipo de sello de tiempo RFC 3161

```
String TSA_APPLICATION_NAME = "pruebasTest";
```

```
/*
* INICIO SELLADO
*/
file = UtilsFileSystemCommons.readFile("ficheroAfirmar.txt", true);
Map<String, Object> inParams = new HashMap<String, Object>();
String base64Data = new String(Base64CoderCommons.encodeBase64(file));
inParams.put(DSSTagsRequest.BASE64DATA, base64Data);
inParams.put(DSSTagsRequest.CLAIMED_IDENTITY_TSA, TSA_APPLICATION_NAME);
inParams.put(DSSTagsRequest.SIGNATURE_TYPE, DSSConstants.TimestampForm.RFC_3161);

String xmlInput = TransformersFacade.getInstance().generateXml(inParams,
GeneralConstants.DSS_TSA_REQUEST, GeneralConstants.TSA_TIMESTAMP_SERVICE,
TransformersConstants.VERSION_10);
String xmlOutput = TSAServiceInvokerFacade.getInstance().invokeService(xmlInput,
GeneralConstants.TSA_TIMESTAMP_SERVICE, TSA_APPLICATION_NAME);

Map<String, Object> propertiesResult =
TransformersFacade.getInstance().parseResponse(xmlOutput,
GeneralConstants.DSS_TSA_REQUEST, GeneralConstants.TSA_TIMESTAMP_SERVICE,
TransformersConstants.VERSION_10);

String rfcTimeStamp = (String)
propertiesResult.get(TransformersFacade.getInstance().getParserParameterValue("RFC
3161Timestamp"));
/*
* FIN SELLADO
*/

/*
* INICIO VALIDACIÓN
*/
inParams = new HashMap<String, Object>();
inParams.put(DSSTagsRequest.BASE64DATA, base64Data);
inParams.put(DSSTagsRequest.CLAIMED_IDENTITY_TSA, TSA_APPLICATION_NAME);
inParams.put(DSSTagsRequest.TIMESTAMP_RFC3161_TIMESTAMPTOKEN, rfcTimeStamp);

xmlInput = TransformersFacade.getInstance().generateXml(inParams,
GeneralConstants.DSS_TSA_REQUEST,
GeneralConstants.TSA_TIMESTAMP_VALIDATION_SERVICE,
TransformersConstants.VERSION_10);
xmlOutput = TSAServiceInvokerFacade.getInstance().invokeService(xmlInput,
GeneralConstants.TSA_TIMESTAMP_VALIDATION_SERVICE, TSA_APPLICATION_NAME);

propertiesResult = TransformersFacade.getInstance().parseResponse(xmlOutput,
GeneralConstants.DSS_TSA_REQUEST,
GeneralConstants.TSA_TIMESTAMP_VALIDATION_SERVICE,
TransformersConstants.VERSION_10);
/*
* FIN VALIDACIÓN */
```

Ejemplo 2. Tipo de sello de tiempo XML

```
String TSA_APPLICATION_NAME = "pruebasTest";

/*
* INICIO SELLADO
*/
file = UtilsFileSystemCommons.readFile("ficheroAfirmar2.xml", true);
```

```
String inputDocumentProcessed = new String(Base64CoderCommons.encodeBase64(file));
Map<String, Object> inParams = new HashMap<String, Object>();
inParams.put(DSSTagsRequest.BASE64XML, inputDocumentProcessed);
inParams.put(DSSTagsRequest.CLAIMED_IDENTITY_TSA, TSA_APPLICATION_NAME);
inParams.put(DSSTagsRequest.SIGNATURE_TYPE, DSSConstants.TimestampForm.XML);

String xmlInput = TransformersFacade.getInstance().generateXml(inParams,
GeneralConstants.DSS_TSA_REQUEST, GeneralConstants.TSA_TIMESTAMP_SERVICE,
TransformersConstants.VERSION_10);
String xmlOutput = TSAServiceInvokerFacade.getInstance().invokeService(xmlInput,
GeneralConstants.TSA_TIMESTAMP_SERVICE, TSA_APPLICATION_NAME);

Map<String, Object> propertiesResult =
TransformersFacade.getInstance().parseResponse(xmlOutput,
GeneralConstants.DSS_TSA_REQUEST, GeneralConstants.TSA_TIMESTAMP_SERVICE,
TransformersConstants.VERSION_10);
String xmlTimeStamp = (String)
propertiesResult.get(TransformersFacade.getInstance().getParserParameterValue("XML
Timestamp"));
/*
* FIN SELLADO
*/

/*
* INICIO VALIDACIÓN
*/
inParams = new HashMap<String, Object>();
inParams.put(DSSTagsRequest.BASE64XML, inputDocumentProcessed);
inParams.put(DSSTagsRequest.CLAIMED_IDENTITY_TSA, TSA_APPLICATION_NAME);
inParams.put(DSSTagsRequest.TIMESTAMP_XML_TIMESTAMPTOKEN, xmlTimeStamp);

xmlInput = TransformersFacade.getInstance().generateXml(inParams,
GeneralConstants.DSS_TSA_REQUEST,
GeneralConstants.TSA_TIMESTAMP_VALIDATION_SERVICE,
TransformersConstants.VERSION_10);
xmlOutput = TSAServiceInvokerFacade.getInstance().invokeService(xmlInput,
GeneralConstants.TSA_TIMESTAMP_VALIDATION_SERVICE, TSA_APPLICATION_NAME);

propertiesResult = TransformersFacade.getInstance().parseResponse(xmlOutput,
GeneralConstants.DSS_TSA_REQUEST,
GeneralConstants.TSA_TIMESTAMP_VALIDATION_SERVICE,
TransformersConstants.VERSION_10);
/*
* FIN VALIDACIÓN
*/
```

A.3.9 Servicios Nativos @Firma. Validación de certificados.

En este punto se expone solamente un ejemplo de acceso a servicios nativos de @Firma dado el carácter deprecado de estos servicios. En concreto se representa un ejemplo de validación de certificados.

Se hará uso de un código como el siguiente si se desea acceder al Servicio Nativo de Validación de Certificado de @Firma, se incluirá el siguiente código para una validación de certificado donde se validará la caducidad, integridad, confianza, estado de revocación y cadena de confianza al completo del certificado (modo de validación *2*):

```
String APPLICATION_NAME = "afirmaTest";
String certB64 = UtilsFileSystem.readFileBase64Encoded("serversigner.cer", true);
java.util.Map<String, Object> inParams = new HashMap<String, Object>();
inParams.put(NativeTagsRequest.ID_APLICACION, APPLICATION_NAME);
inParams.put(NativeTagsRequest.CERTIFICADO, certB64);
inParams.put(NativeTagsRequest.MODO_VALIDACION, "2");
inParams.put(NativeTagsRequest.OBTENER_INFO, Boolean.TRUE.toString());

String xmlInput = TransformersFacade.getInstance().generateXml(inParams,
GeneralConstants.VALIDACION_CERTIFICADO_REQUEST,
GeneralConstants.VALIDACION_CERTIFICADO_REQUEST,
TransformersConstants.VERSION_10);

String xmlOutput =
Afirma5ServiceInvokerFacade.getInstance().invokeService(xmlInput,
GeneralConstants.VALIDACION_CERTIFICADO_REQUEST,
GeneralConstants.VALIDACION_CERTIFICADO_REQUEST, APPLICATION_NAME);

java.util.Map<String, Object> propertiesResult =
TransformersFacade.getInstance().parseResponse(xmlOutput,
GeneralConstants.VALIDACION_CERTIFICADO_REQUEST,
GeneralConstants.VALIDACION_CERTIFICADO_REQUEST,
TransformersConstants.VERSION_10);


String validationResult =
GenericUtils.getValueFromMapsTree("ResultadoValidacion/resultado",
propertiesResult);
```

En este caso, como se ha indicado que se desea obtener información del certificado, será posible acceder en la respuesta a valores propios del certificado, como por ejemplo, el número de serie, accediendo a la clave correspondiente del mapa de parámetros:

```
String certificateSerialNumber =
GenericUtils.getValueFromMapsTree("InfoCertificado/serialNumber",
propertiesResult);
```

A.3.10 Generación de firma mediante Fachada de Integración de @Firma.

Se hará uso de un código como el siguiente para generar firmas CAdES mediante la Fachada de Integración de @Firma.

```
String APPLICATION_NAME = "afirmaTest";
String SERVER_SIGNER_NAME = "raul conde";
```

```
ServerSignerRequest serSigReq = new ServerSignerRequest();

byte[ ] documentB64 =
UtilsFileSystemCommons.getArrayByteFileBase64Encoded("ficheroAfirmar.txt", true);
serSigReq.setDocument(documentB64);

serSigReq.setKeySelector(SERVER_SIGNER_NAME);
serSigReq.setApplicationId(APPLICATION_NAME);
serSigReq.setSignatureFormat(SignatureFormatEnum.CAdES);
serSigReq.setIgnoreGracePeriod(false);
ServerSignerResponse serSigRes = IntegraFacadeWSDSS.getInstance().sign(serSigReq);
```

A.3.11 Generación de co-firma mediante Fachada de Integración de @Firma.

Se hará uso de un código como el siguiente si se desea solicitar una co-firma mediante Fachada de Integración de @Firma sobre una firma previamente realizada. En este ejemplo, el identificador de la firma previa, se supone conocido.

```
String APPLICATION_NAME = "afirmaTest";
String SERVER_SIGNER_NAME = "raul conde";

String idTransaction = "145346464856296185";
CoSignRequest coSigReq = new CoSignRequest();
coSigReq.setTransactionId(idTransaction);
coSigReq.setKeySelector(SERVER_SIGNER_NAME);
coSigReq.setApplicationId(APPLICATION_NAME);
coSigReq.setHashAlgorithm(HashAlgorithmEnum.SHA1);

ServerSignerResponse ssr = IntegraFacadeWSDSS.getInstance().coSign(coSigReq);
```

A.3.12 Generación de contra-firma mediante Fachada de Integración de @Firma.

Se hará uso de un código como el siguiente si se desea solicitar una contra-firma mediante Fachada de Integración de @Firma sobre una firma previamente realizada. En este ejemplo, el identificador de la firma previa, se supone conocido.

```
String APPLICATION_NAME = "afirmaTest";
String SERVER_SIGNER_NAME = "raul conde";

String idTransaction = "145346464856296185";
CounterSignRequest couSigReq = new CounterSignRequest();
couSigReq.setTransactionId(idTransaction);
couSigReq.setKeySelector(SERVER_SIGNER_NAME);
couSigReq.setApplicationId(APPLICATION_NAME);
couSigReq.setHashAlgorithm(HashAlgorithmEnum.SHA1);

ServerSignerResponse ssr =
IntegraFacadeWSDSS.getInstance().counterSign(couSigReq);
```

A.3.13 Validación de firmas mediante Fachada de Integración de @Firma.

Se hará uso de un código como el siguiente si se desea enviar una solicitud de validación de firma mediante Fachada de Integración de @Firma sobre una firma previamente realizada. La firma realizada previamente se supone estará alamacenada en la variable SIGNATURE_XADES.

```
String APPLICATION_NAME = "afirmaTest";

VerifySignatureRequest verSigReq = new VerifySignatureRequest();
verSigReq.setApplicationId(APPLICATION_NAME);

byte[ ] signB64 = UtilsFileSystemCommons.readFile(SIGNATURE_XADES, true);
verSigReq.setSignature(signB64);

OptionalParameters optParam = new OptionalParameters();
optParam.setReturnProcessingDetails(true);
optParam.setReturnReadableCertificateInfo(true);
verSigReq.setOptionalParameters(optParam);

VerificationReport verRep = new VerificationReport();
verRep.setReportDetailLevel(DetailLevelEnum.ALL_DETAILS);
verRep.setIncludeRevocationValues(true);
verRep.setIncludeCertificateValues(true);

verSigReq.setVerificationReport(verRep);

VerifySignatureResponse vsr =
IntegraFacadeWSDSS.getInstance().verifySignature(verSigReq);
```

A.3.14 Actualización de firmas mediante Fachada de Integración de @Firma.

Se hará uso de un código como el siguiente si se desea enviar una solicitud de actualización de firma mediante Fachada de Integración de @Firma sobre una firma previamente realizada. La firma realizada previamente se supone estará alamacenada en la variable SIGNATURE_XADES. La actualización se realiza al formato T.

```
String APPLICATION_NAME = "afirmaTest";

UpgradeSignatureRequest upgSigReq = new UpgradeSignatureRequest();
// firma sin codificar
byte[ ] signature = UtilsFileSystemCommons.readFile(SIGNATURE_XADES, true);

upgSigReq.setSignature(signature);
upgSigReq.setApplicationId(APPLICATION_NAME);
upgSigReq.setSignatureFormat(SignatureFormatEnum.XAdES_T);

ServerSignerResponse serSigRes =
IntegraFacadeWSDSS.getInstance().upgradeSignature(upgSigReq);
```

A.3.15 Validación de certificados firmas mediante Fachada de Integración de @Firma.

Se hará uso de un código como el siguiente si se desea solicitar una validación de certificado firmas mediante Fachada de Integración de @Firma.

```
String APPLICATION_NAME = "afirmaTest";
```

```
String CERTIFICATE_NAME = "confianzaocsp.crt";

VerifyCertificateRequest verCerReq = new VerifyCertificateRequest();

verCerReq.setApplicationId(APPLICATION_NAME);
byte[ ] certificate = UtilsFileSystemCommons.readFile(CERTIFICATE_NAME, true);
verCerReq.setCertificate(certificate);
verCerReq.setReturnReadableCertificateInfo(true);
VerificationReport verRep = new VerificationReport();
verRep.setIncludeCertificateValues(true);
verRep.setCheckCertificateStatus(true);
verRep.setIncludeRevocationValues(true);
verRep.setReportDetailLevel(DetailLevelEnum.ALL_DETAILS);
verCerReq.setReturnVerificationReport(verRep);
VerifyCertificateResponse verCerRes =
IntegraFacadeWSDSS.getInstance().verifyCertificate(verCerReq);
```

A.3.16 Generación de reportes eVisor.

Se hará uso de un código como el siguiente si se desea solicitar un reporte de firma a eVisor.

```
String APPLICATION_NAME = "afirmaTestEVisor";
String TEMPLATE_ID = "pdf_escalado";

Map<String, Object> inputParams = new HashMap<String, Object>();

String signB64 =
UtilsFileSystemCommons.readFileBase64Encoded("evisor/PADES_Signature.pdf", true);

inputParams.put(EVisorTagsRequest.APPLICATION_ID, APPLICATION_NAME);
inputParams.put(EVisorTagsRequest.TEMPLATE_ID, TEMPLATE_ID);
inputParams.put(EVisorTagsRequest.ENCODED_SIGNATURE, signB64);
inputParams.put(EVisorTagsRequest.INCLUDE_SIGNATURE, "true");

Map<String, String> qRCodeParams = new HashMap<String, String>(2);
qRCodeParams.put("QRCodeWidth", "600");
qRCodeParams.put("QRCodeHeight", "600");
qRCodeParams.put("Rotation", "90");

// introducimos un conjunto de códigos de barra en los parámetros de
// entrada.
inputParams.put(EVisorTagsRequest.BARCODE, new Map<?, ?>[ ] {
EVisorUtilCommons.newBarcodeMap("Prueba código barra tipo QRCode", "QRCode",
qRCodeParams), EVisorUtilCommons.newBarcodeMap("986656487", "EAN128", null),
EVisorUtilCommons.newBarcodeMap("Prueba código barra tipo DataMatrix",
"DataMatrix", null) });

String inputXml = TransformersFacade.getInstance().generateXml(inputParams,
EVisorConstants.SIGNATURE_REPORT_SERVICE, EVisorConstants.GENERATE_REPORT_METHOD,
TransformersConstants.VERSION_10);

String outputXml =
Afirma5ServiceInvokerFacade.getInstance().invokeService(inputXml,
EVisorConstants.SIGNATURE_REPORT_SERVICE, EVisorConstants.GENERATE_REPORT_METHOD,
APPLICATION_NAME);
```

```
Map<String, Object> result =
TransformersFacade.getInstance().parseResponse(outputXml,
EVisorConstants.SIGNATURE_REPORT_SERVICE, EVisorConstants.GENERATE_REPORT_METHOD,
TransformersConstants.VERSION_10);
```

A.3.17 Generación, verificación y renovación de un sello de tiempo mediante Fachada de Integración de TS@ con un documento de tipo "DocumentHash".

Se hará uso de un código como el siguiente para generar un sello de tiempo de tipo XML a partir de un fichero de tipo "DocumentHash" con la Fachada de Integración de TS@. Cambiando el tipo de sello de tiempo por RFC_3161 lo obtendríamos del tipo indicado.

```
String APPLICATION_NAME = "pruebasTest";

TimestampRequest timestampReq = new TimestampRequest();
DocumentHash docH = new DocumentHash();
byte[ ] file;

// Prueba de DocumentHash
file = UtilsFileSystemCommons.readFile("ficheroAfirmar.txt", true);
timestampReq.setDocumentType(DocumentTypeEnum.DOCUMENT_HASH);
docH.setDigestValue(file);
docH.setDigestMethod(DSSConstants.AlgorithmTypes.SHA1);
timestampReq.setDocumentHash(docH);
timestampReq.setTimestampType(TimestampTypeEnum.XML);
timestampReq.setApplicationId(APPLICATION_NAME);

// Sellado
TimestampResponse timestampRes =
TsaIntegraFacadeWSDSS.getInstance().generateTimestamp(timestampReq);

// Verificacion sellado
timestampReq.setTimestampTimestampToken(timestampRes.getTimestamp());
TimestampResponse timestampResVerify =
TsaIntegraFacadeWSDSS.getInstance().verifyTimestamp(timestampReq);

// Resellado
timestampReq.setTimestampPreviousTimestampToken(timestampRes.getTimestamp());
TimestampResponse timestampResRenew =
TsaIntegraFacadeWSDSS.getInstance().renewTimestamp(timestampReq);
```

A.3.18 Generación, verificación y renovación de un sello de tiempo mediante Fachada de Integración de TS@ con un documento de tipo "DocumentHashTransformedData".

Se hará uso de un código como el siguiente para generar un sello de tiempo de tipo XML a partir de un fichero de tipo "DocumentHashTransformedData" con la Fachada de Integración de TS@. Cambiando el tipo de sello de tiempo por XML lo obtendríamos del tipo indicado.

```
String APPLICATION_NAME = "pruebasTest";

TimestampRequest timestampReq = new TimestampRequest();
DocumentHash docH = new DocumentHash();
byte[ ] file;
List<String> canonicalizer;
```

```
// Prueba de DocumentHashTransformedData
file = UtilsFileSystemCommons.readFile("ficheroAfirmar2.xml", true);

timestampReq.setDocumentType(DocumentTypeEnum.DOCUMENT_HASH_TRANSFORMED_DATA);
docH = new DocumentHash();
canonicalizer = new ArrayList<>();
canonicalizer.add(Canonicalizer.ALGO_ID_C14N_EXCL_OMIT_COMMENTS);
TransformData td = new TransformData("SHA1", canonicalizer);
docH.setTransform(td);

docH.setDigestMethod(CryptoUtilXML.translateDigestAlgorithmToXMLURI(td.getAlgorith
m()));
docH.setDigestValue(file);
timestampReq.setDocumentHash(docH);
timestampReq.setTimestampType(TimestampTypeEnum.RFC_3161);
timestampReq.setApplicationId(APPLICATION_NAME);

// Sellado
TimestampResponse timestampRes =
TsaIntegraFacadeWSDSS.getInstance().generateTimestamp(timestampReq);

// Verificacion sellado
timestampReq.setTimestampTimestampToken(timestampRes.getTimestamp());
TimestampResponse timestampResVerify =
TsaIntegraFacadeWSDSS.getInstance().verifyTimestamp(timestampReq);

// Resellado
timestampReq.setTimestampPreviousTimestampToken(timestampRes.getTimestamp());
TimestampResponse timestampResRenew =
TsaIntegraFacadeWSDSS.getInstance().renewTimestamp(timestampReq);
```

A.3.19 Generación, verificación y renovación de un sello de tiempo mediante Fachada de Integración de TS@ con un documento de tipo “TransformedData”.

Se hará uso de un código como el siguiente para generar un sello de tiempo de tipo XML a partir de un fichero de tipo “TransformedData” con la Fachada de Integración de TS@.

```
String APPLICATION_NAME = "pruebasTest";

TimestampRequest timestampReq = new TimestampRequest();
List<String> canonicalizer;
byte[ ] file;

// Prueba de TransformedData
file = UtilsFileSystemCommons.readFile("ficheroAfirmar2.xml", true);
timestampReq.setDocumentType(DocumentTypeEnum.TRANSFORMED_DATA);
canonicalizer = new ArrayList<>();
canonicalizer.add(Canonicalizer.ALGO_ID_C14N_EXCL_OMIT_COMMENTS);
TransformData td = new TransformData("SHA1", canonicalizer);
timestampReq.setTransformData(td);
timestampReq.setDataToStamp(file);

timestampReq.setTimestampType(TimestampTypeEnum.XML);
timestampReq.setApplicationId(APPLICATION_NAME);

// Sellado
```

```
TimestampResponse timestampRes =
TsaIntegraFacadeWSDSS.getInstance().generateTimestamp(timestampReq);

// Verificacion sellado
timestampReq.setTimestampTimestampToken(timestampRes.getTimestamp());
TimestampResponse timestampResVerify =
TsaIntegraFacadeWSDSS.getInstance().verifyTimestamp(timestampReq);

// Resellado
timestampReq.setTimestampPreviousTimestampToken(timestampRes.getTimestamp());
TimestampResponse timestampResRenew =
TsaIntegraFacadeWSDSS.getInstance().renewTimestamp(timestampReq);
```

A.3.20 Generación, verificación y renovación de un sello de tiempo mediante Fachada de Integración de TS@ con un documento de tipo "Base64Data".

Se hará uso de un código como el siguiente para generar un sello de tiempo de tipo XML a partir de un fichero de tipo "Base64Data" con la Fachada de Integración de TS@. Cambiando el tipo de sello de tiempo por XML lo obtendríamos del tipo indicado.

```
String APPLICATION_NAME = "pruebasTest";

TimestampRequest timestampReq = new TimestampRequest();
byte[ ] file;

// Prueba de Base64Data
file = UtilsFileSystemCommons.readFile("ficheroAfirmar.txt", true);
timestampReq.setDocumentType(DocumentTypeEnum.BASE64_DATA);
timestampReq.setDataToStamp(file);
timestampReq.setTimestampType(TimestampTypeEnum.RFC_3161);
timestampReq.setApplicationId(APPLICATION_NAME);

// Sellado
TimestampResponse timestampRes =
TsaIntegraFacadeWSDSS.getInstance().generateTimestamp(timestampReq);

// Verificacion sellado
timestampReq.setTimestampTimestampToken(timestampRes.getTimestamp());
TimestampResponse timestampResVerify =
TsaIntegraFacadeWSDSS.getInstance().verifyTimestamp(timestampReq);

// Resellado
timestampReq.setTimestampPreviousTimestampToken(timestampRes.getTimestamp());
TimestampResponse timestampResRenew =
TsaIntegraFacadeWSDSS.getInstance().renewTimestamp(timestampReq);
```

A.3.21 Generación, verificación y renovación de un sello de tiempo mediante Fachada de Integración de TS@ con un documento de tipo "Base64XML".

Se hará uso de un código como el siguiente para generar un sello de tiempo de tipo XML a partir de un fichero de tipo "Base64XML" con la Fachada de Integración de TS@.

```
String APPLICATION_NAME = "pruebasTest";
```

```
TimestampRequest timestampReq = new TimestampRequest();
byte[ ] file;

// Prueba de Base64XML
file = UtilsFileSystemCommons.readFile("ficheroAfirmar2.xml", true);
timestampReq.setDocumentType(DocumentTypeEnum.BASE64_XML);
timestampReq.setDataToStamp(file);
timestampReq.setTimestampType(TimestampTypeEnum.XML);
timestampReq.setApplicationId(APPLICATION_NAME);

// Sellado
TimestampResponse timestampRes =
TsaIntegraFacadeWSDSS.getInstance().generateTimestamp(timestampReq);

// Verificacion sellado
timestampReq.setTimestampTimestampToken(timestampRes.getTimestamp());
TimestampResponse timestampResVerify =
TsaIntegraFacadeWSDSS.getInstance().verifyTimestamp(timestampReq);

// Resellado
timestampReq.setTimestampPreviousTimestampToken(timestampRes.getTimestamp());
TimestampResponse timestampResRenew =
TsaIntegraFacadeWSDSS.getInstance().renewTimestamp(timestampReq);
```

A.3.22 Generación, verificación y renovación de un sello de tiempo mediante Fachada de Integración de TS@ con un documento de tipo “InlineXML”.

Se hará uso de un código como el siguiente para generar un sello de tiempo de tipo XML a partir de un fichero de tipo “InlineXML” con la Fachada de Integración de TS@.

```
String APPLICATION_NAME = "pruebasTest";

TimestampRequest timestampReq = new TimestampRequest();
byte[ ] file;

// Prueba de InlineXML
file = UtilsFileSystemCommons.readFile("ficheroAfirmar2.xml", true);
timestampReq.setDocumentType(DocumentTypeEnum.INLINE_XML);
timestampReq.setDataToStamp(file);

timestampReq.setTimestampType(TimestampTypeEnum.XML);
timestampReq.setApplicationId(APPLICATION_NAME);

// Sellado
TimestampResponse timestampRes =
TsaIntegraFacadeWSDSS.getInstance().generateTimestamp(timestampReq);

// Verificacion sellado
timestampReq.setTimestampTimestampToken(timestampRes.getTimestamp());
TimestampResponse timestampResVerify =
TsaIntegraFacadeWSDSS.getInstance().verifyTimestamp(timestampReq);

// Resellado
timestampReq.setTimestampPreviousTimestampToken(timestampRes.getTimestamp());
```

```
TimestampResponse timestampResRenew =
TsaIntegraFacadeWSDSS.getInstance().renewTimestamp(timestampReq);
```

A.3.23 Generación, verificación y renovación de un sello de tiempo mediante Fachada de Integración de TS@ con un documento de tipo "EscapedXML".

Se hará uso de un código como el siguiente para generar un sello de tiempo de tipo XML a partir de un fichero de tipo "EscapedXML" con la Fachada de Integración de TS@.

```
String APPLICATION_NAME = "pruebasTest";

TimestampRequest timestampReq = new TimestampRequest();
byte[ ] file;

// Prueba de EscapedXML
file = UtilsFileSystemCommons.readFile("ficheroAfirmarEscapado.xml",true);
timestampReq.setDocumentType(DocumentTypeEnum.ESCAPED_XML);
timestampReq.setDataToStamp(file);

timestampReq.setTimestampType(TimestampTypeEnum.XML);
timestampReq.setApplicationId(APPLICATION_NAME);

// Sellado
TimestampResponse timestampRes =
TsaIntegraFacadeWSDSS.getInstance().generateTimestamp(timestampReq);

// Verificacion sellado
timestampReq.setTimestampTimestampToken(timestampRes.getTimestamp());
TimestampResponse timestampResVerify =
TsaIntegraFacadeWSDSS.getInstance().verifyTimestamp(timestampReq);

// Resellado
timestampReq.setTimestampPreviousTimestampToken(timestampRes.getTimestamp());
TimestampResponse timestampResRenew =
TsaIntegraFacadeWSDSS.getInstance().renewTimestamp(timestampReq);
```

A.4 Ejemplos de Uso de los Servicios para el tipo de integración 3

El conjunto de ejemplos para este tipo de integración será la suma de los ejemplos de los dos tipos de integración anteriores A.1 y A.3.

A.5 Ejemplos de Uso de los Servicios para el tipo de integración 4

En este anexo se incluyen algunos ejemplos de uso relacionados con la generación, validación y actualización de firmas CAdES (Baseline o no) y PAdES (Baseline o no) realizadas por Integr@, así como la obtención de información de los datos originalmente firmados.

Hay que señalar que todas las operaciones y por lo tanto **los ejemplos expuestos para el tipo de integración 1 (Acceso a servicios OCSP y RFC 3161) pueden realizarse también con este tipo de integración**.

A.5.1 Configuración general.

Para este tipo de integración es necesario hacer uso de las librerías de Integr@ y librerías de terceros enumeradas en el apartado 7.5.

Por otro lado, será necesario disponer en una ruta accesible desde la aplicación en la que se usa Integr@ de los ficheros de configuración necesarios para este caso concreto y que pueden encontrarse en el entregable. Puenden verse los ficheros de propiedades necesarios en el apartado 8.4.

Para el caso que nos ocupa damos por hecho que se encuentra desplegada la plataforma de TS@ con una aplicación "pruebasTest" dada de alta y correctamente configurada. De igual forma, si en el fichero **integra.properties** cuya configuración se detalla a continuación se opta por realizar validaciones de estado de revocación de certificados (CERTIFICATE_VALIDATION_LEVEL = 2), será necesario tener desplegada la plataforma de @Firma con una aplicación "afirmaTest" creada y correctamente configurada con el acceso mediante ocsp.

A continuación, se procederá a configurar el lenguaje de los mensajes de Integr@ editando el archivo de propiedades **Language.properties**:

```
LANGUAGE = es_ES
```

Con esta configuración la plataforma emitirá los mensajes en español.

Se procederá a configurar el archivo **integra.properties** añadiendo:

```
CERTIFICATE_VALIDATION_LEVEL = 1
TSA_APP_ID = pruebasTest
TSA_COMMUNICATION_TYPE = RFC3161-TCP
TSA_TIMESTAMP_TYPE = ASN1
```

Con esta configuración se extrae que, tanto el certificado firmante a usar en la generación de la firma CAdES (Baseline o no) o PAdES (Baseline o no), como el certificado firmante del sello de tiempo, se validarán respecto al periodo de validez y caducidad; y que Integr@ se comunicará con el Servicio de Generación de Sello de Tiempo con protocolo RFC 3161 de TS@ para obtener un sello de tiempo ASN.1. Si se configura el parámetro `CERTIFICATE_VALIDATION_LEVEL` con valor 2 será necesario añadir al archivo **integra.properties** con los datos de acceso al servicio ocsp de validación de certificados de @Firma (la misma configuración que se detalla en el ejemplo A.2.1).

También se añadirá al archivo **integra.properties** las propiedades de la política de firma a utilizar en las firmas ASN.1 y PDF en la generación de firmas con política de firma:

```
ContentType = 1.2.840.113549.1.9.3
MessageDigest = 1.2.840.113549.1.9.4
SigningCertificate = 1.2.840.113549.1.9.16.2.12
SigningCertificateV2 = 1.2.840.113549.1.9.16.2.47
SigningTime = 1.2.840.113549.1.9.5
SignaturePolicyIdentifier = 1.2.840.113549.1.9.16.2.15
```

```
ContentHints = 1.2.840.113549.1.9.16.2.4
ContentReference = 1.2.840.113549.1.9.16.2.10
ContentIdentifier = 1.2.840.113549.1.9.16.2.7
SignerLocation = 1.2.840.113549.1.9.16.2.17
SignerAttributes = 1.2.840.113549.1.9.16.2.18
ContentTimeStamp = 1.2.840.113549.1.9.16.2.20
CounterSignature = 1.2.840.113549.1.9.6
CommitmentTypeIndication = 1.2.840.113549.1.9.16.2.16
ArchiveTimeStamp = 1.2.840.113549.1.9.16.2.48
CompleteCertificateRefs = 1.2.840.113549.1.9.16.2.21
CompleteRevocationRefs = 1.2.840.113549.1.9.16.2.22
TimestampedCertsCRLs = 1.2.840.113549.1.9.16.2.26
Data = 1.2.840.113549.1.7.1
#Listado de OIDs para algoritmos de hash en firmas ASN.1 y PDF.
ASN1_HASH-SHA1 = 1.3.14.3.2.26
ASN1_HASH-SHA256 = 2.16.840.1.101.3.4.2.1
ASN1_HASH-SHA512 = 2.16.840.1.101.3.4.2.3
#Listado de OIDs para algoritmos de firma en firmas ASN.1 y PDF.
ASN1_SIGN_HASH-SHA1WithRSA = 1.2.840.113549.1.1.5
ASN1_SIGN_HASH-SHA256WithRSA = 1.2.840.113549.1.1.11
ASN1_SIGN_HASH-SHA512WithRSA = 1.2.840.113549.1.1.13
#Identificador de la política de firma a usar en la generación de firmas ASN.1.
ASN1_POLICY_ID = ASN1_AGE_1.9
#Identificador de la política de firma.
ASN1_AGE_1.9-IDENTIFIER_ASN1 = 2.16.724.1.3.1.1.2.1.9
#Algoritmo de resumen a usar para calcular la huella digital del documento legible
de política de firma.
ASN1_AGE_1.9-HASH_ALGORITHM = SHA-1
#Valor del hash de la política de firma.
ASN1_AGE_1.9-HASH_VALUE = 7SxX3erFuH31TvAw9LZ70N7p1vA=
#Algoritmos de resumen válidos.
ASN1_AGE_1.9-ALLOWED_HASH_ALGORITHM = ASN1_HASH-SHA1,ASN1_HASH-SHA256,ASN1_HASH-
SHA512
#Algoritmos de firma válidos.
ASN1_AGE_1.9-ALLOWED_SIGN_ALGORITHM = ASN1_SIGN_HASH-SHA1WithRSA,ASN1_SIGN_HASH-
SHA256WithRSA,ASN1_SIGN_HASH-SHA512WithRSA
#Descripción de la política de firma
ASN1_AGE_1.9-DESCRIPTION = Política de Firma Electrónica y de Certificados de la
Administración General del Estado 1.9 para firmas CAdES
#Elementos firmados obligatorios.
ASN1_AGE_1.9-MANDATORY_SIGNED_ELEMENTS =
ContentType,MessageDigest,SigningCertificate|SigningCertificateV2,SigningTime,Sign
aturePolicyIdentifier,ContentHints
#Elementos firmados opcionales.
ASN1_AGE_1.9-OPTIONAL_SIGNED_ELEMENTS =
ContentReference,ContentIdentifier,CommitmentTypeIndication,SignerLocation,SignerA
ttributes,ContentTimeStamp
#Elementos no firmados opcionales.
ASN1_AGE_1.9-OPTIONAL_UNSIGNED_ELEMENTS = CounterSignature
#Modos de firma permitidos.
ASN1_AGE_1.9-ALLOWED_SIGNING_MODES = Implicit,Explicit


PDF_POLICY_ID =PDF_AGE_1.9
#Identificador de la política de firma.
PDF_AGE_1.9-IDENTIFIER_PDF = 2.16.724.1.3.1.1.2.1.9
#Algoritmo de resumen a usar para calcular la huella digital del documento legible
de política de firma.
PDF_AGE_1.9-HASH_ALGORITHM = SHA-1
```

```
#Valor del hash de la política de firma.
PDF_AGE_1.9-HASH_VALUE = 7SxX3erFuH31TvAw9LZ70N7p1vA=
#Algoritmos de resumen válidos.
PDF_AGE_1.9-ALLOWED_HASH_ALGORITHM = ASN1_HASH-SHA1,ASN1_HASH-SHA256,ASN1_HASH-
SHA512
#Algoritmos de firma válidos.
PDF_AGE_1.9-ALLOWED_SIGN_ALGORITHM = ASN1_SIGN_HASH-SHA1WithRSA,ASN1_SIGN_HASH-
SHA256WithRSA,ASN1_SIGN_HASH-SHA512WithRSA
#Descripción de la política de firma
PDF_AGE_1.9-DESCRIPTION = Política de Firma Electrónica y de Certificados de la
Administración General del Estado 1.9 para firmas PAdES
#Elementos firmados obligatorios.
PDF_AGE_1.9-MANDATORY_SIGNED_ELEMENTS =
ContentType,MessageDigest,SigningCertificate|SigningCertificateV2,SignaturePolicyI
dentifier
#Elementos firmados opcionales.
PDF_AGE_1.9-OPTIONAL_SIGNED_ELEMENTS =
CommitmentTypeIndication,SignerAttributes,ContentTimeStamp
#Elementos firmados no permitidos.
PDF_AGE_1.9-NOT_ALLOWED_SIGNED_ELEMENTS = ContentHints,SigningTime
#Elementos no firmados no permitidos.
PDF_AGE_1.9-NOT_ALLOWED_UNSIGNED_ELEMENTS = CounterSignature
```

Con esta configuración se aplicarían las restricciones asociadas a la Política de Firma Electrónica y de Certificados de la Administración General del Estado 1.9 en lo que se refiere a firmas CAdES (Baseline o no) y PAdES (Baseline o no).

Si se va a hacer uso de la fachada de Integr@, como en el caso del ejemplo A.5.2, será necesario editar el fichero **integra.properties** añadiendo valores como los siguientes.

```
FACADE_SIGNATURE_ALGORITHM = SHA256withRSA
FACADE_SIGNATURE_TYPE = CAdES
```

Si se van a realizar peticiones de sello de tiempo a TS@ mediante RFC 3161 será necesario añadir al archivo de propiedades **integra.properties**, los parámetros generales de acceso a TS@. Para el ejemplo que nos ocupa la configuración sería la siguiente.

```
com.trustedstorePath = D:/workspace-baseJdk1.8/Integra-parent/Integra-
ws/src/test/resources/truststoreWS.jks

com.trustedstorePassword = 12345

com.serviceWSDLPath = file:/D:/workspace-baseJdk1.8/Integra-parent/Integra-
ws/src/test/resources/TimeStampWS.wsdl
```

También será necesario crear y configurar el fichero **tsapruebasTest.properties** con la siguiente configuración para el ejemplo que nos ocupa.

```
callTimeout =20000
authorizationMethod =UserNameToken
UserNameToken.userName = user
```

```
UserNameToken.userPassword = 12345
request.symmetricKey.use =false
renewTimeStampWS.validationLevel = 0
```

Finalmente sera necesario mapear los ficheros de configuración dinámicos en **mappingFiles.properties**.

```
tsapruebasTest = tsapruebasTest.properties
integra = integra.properties
```

A.5.2 Fachada de Integr@: Generación, validación y actualización de firma, co-firma y contrafirma.

En el ejemplo que se expone a continuación se realizan operaciones de firma, co-firma, contra-firma, actualización y validación:

```
byte[ ] dataToSign =
UtilsFileSystemCommons.getArrayByteFileBase64Encoded("ficheroAfirmar.txt", true);
PrivateKeyEntry privateKey = "clave privada del certificado a utilizar";

// Generación y validación de firma AdES-BES
byte[ ] signatureAdESBES =
es.gob.afirma.integraFacade.IntegraFacade.generateSignature(dataToSign,
privateKey, false, false);
es.gob.afirma.signature.ValidationResult vr =
(es.gob.afirma.signature.validation.ValidationResult)es.gob.afirma.integraFacade.I
ntegraFacade.verifySignature(signatureAdESBES, dataToSign);

// Generación y validación de co-firma AdES-BES sobre firma AdES-BES
byte[ ] coSignatureAdESBES =
es.gob.afirma.integraFacade.IntegraFacade.generateCoSignature(signatureAdESBES,
dataToSign, privateKey, false, false);
vr =
(es.gob.afirma.signature.validation.ValidationResult)es.gob.afirma.integraFacade.I
ntegraFacade.verifySignature(coSignatureAdESBES, dataToSign);

// Generación y validación de contra-firma AdES-BES sobre firma AdES-BES
byte[ ] counterSignatureAdESBES =
es.gob.afirma.integraFacade.IntegraFacade.generateCounterSignature(signatureAdESBE
S, privateKey, false, false);
vr =
(es.gob.afirma.signature.validation.ValidationResult)es.gob.afirma.integraFacade.I
ntegraFacade.verifySignature(counterSignatureAdESBES, dataToSign);

// Generación y validación de firma AdES-T a partir de una firma AdES-BES
byte[ ] signatureAdESBEST =
es.gob.afirma.integraFacade.IntegraFacade.upgradeSignature(signatureAdESBES,
null);

// Validación de firma AdES-T generada a partir de una firma AdES-BES
vr =
(es.gob.afirma.signature.validation.ValidationResult)es.gob.afirma.integraFacade.I
ntegraFacade.verifySignature(signatureAdESBEST, dataToSign);
```

```
// Generación y validación de firma AdES-EPES
byte[ ] signatureAdESEPES =
es.gob.afirma.integraFacade.IntegraFacade.generateSignature(dataToSign,
privateKey, true, false);
vr =
(es.gob.afirma.signature.validation.ValidationResult)es.gob.afirma.integraFacade.I
ntegraFacade.verifySignature(signatureAdESEPES, dataToSign);

// Generación y validación de co-firma AdES-EPES sobre firma AdES-EPES
byte[ ] coSignatureAdESEPES =
es.gob.afirma.integraFacade.IntegraFacade.generateCoSignature(signatureAdESEPES,
dataToSign, privateKey, true, false);
vr =
(es.gob.afirma.signature.validation.ValidationResult)es.gob.afirma.integraFacade.I
ntegraFacade.verifySignature(coSignatureAdESEPES, dataToSign);

// Generación y validación de contra-firma AdES-EPES sobre firma AdES-EPES
byte[ ] counterSignatureAdESEPES =
es.gob.afirma.integraFacade.IntegraFacade.generateCounterSignature(signatureAdESEP
ES, privateKey, true, false);
vr =
(es.gob.afirma.signature.validation.ValidationResult)es.gob.afirma.integraFacade.I
ntegraFacade.verifySignature(counterSignatureAdESEPES, dataToSign);

// Generación y validación de firma AdES-T a partir de una firma AdES-EPES
byte[ ] signatureAdESEPEST =
es.gob.afirma.integraFacade.IntegraFacade.upgradeSignature(signatureAdESEPES,
null);

// Validación de firma AdES-T generada a partir de una firma AdES-EPES
vr =
(es.gob.afirma.signature.validation.ValidationResult)es.gob.afirma.integraFacade.I
ntegraFacade.verifySignature(signatureAdESEPEST, dataToSign);

// Generación y validación de firma AdES-T sin política de firma
byte[ ] signatureAdEST =
es.gob.afirma.integraFacade.IntegraFacade.generateSignature(dataToSign,
privateKey, false, true);
vr =
(es.gob.afirma.signature.validation.ValidationResult)es.gob.afirma.integraFacade.I
ntegraFacade.verifySignature(signatureAdEST, dataToSign);

// Generación y validación de co-firma AdES-T sin política de firma sobre firma
AdES-T sin política de firma
byte[ ] coSignatureAdEST =
es.gob.afirma.integraFacade.IntegraFacade.generateCoSignature(signatureAdEST,
dataToSign, privateKey, false, true);
vr =
(es.gob.afirma.signature.validation.ValidationResult)es.gob.afirma.integraFacade.I
ntegraFacade.verifySignature(coSignatureAdEST, dataToSign);

// Generación y validación de contra-firma AdES-T sin política de firma sobre
firma AdES-T sin política de firma
byte[ ] counterSignatureAdEST =
es.gob.afirma.integraFacade.IntegraFacade.generateCounterSignature(signatureAdEST,
privateKey, false, true);
```

```
vr =
(es.gob.afirma.signature.validation.ValidationResult)es.gob.afirma.integraFacade.I
ntegraFacade.verifySignature(counterSignatureAdEST, dataToSign);

// Generación y validación de firma AdES-T con política de firma
byte[ ] signatureAdESTWithPolicy =
es.gob.afirma.integraFacade.IntegraFacade.generateSignature(dataToSign,
privateKey, true, true);
vr =
(es.gob.afirma.signature.validation.ValidationResult)es.gob.afirma.integraFacade.I
ntegraFacade.verifySignature(signatureAdESTWithPolicy, dataToSign);

// Generación y validación de co-firma AdES-T con política de firma sobre firma
AdES-T con política de firma
byte[ ] coSignatureAdESTWithPolicy =
es.gob.afirma.integraFacade.IntegraFacade.generateCoSignature(signatureAdESTWithPo
licy, dataToSign, privateKey, true, true);
vr =
(es.gob.afirma.signature.validation.ValidationResult)es.gob.afirma.integraFacade.I
ntegraFacade.verifySignature(coSignatureAdESTWithPolicy, dataToSign);

// Generación y validación de contra-firma AdES-T con política de firma sobre
firma AdES-T con política de firma
byte[ ] counterSignatureAdESTWithPolicy =
es.gob.afirma.integraFacade.IntegraFacade.generateCounterSignature(signatureAdESTW
ithPolicy, privateKey, true, true);
vr =
(es.gob.afirma.signature.validation.ValidationResult)es.gob.afirma.integraFacade.I
ntegraFacade.verifySignature(counterSignatureAdESTWithPolicy, dataToSign);
```

Como puede verse en el fichero de configuración integraFacade.properties, el tipo de firma configurado (FACADE_SIGNATURE_TYPE = CAdES), con lo que el ejemplo realizará firmas CAdES. Puede modificarse ese valor a PAdES, XAdES, CAdES Baseline, PAdES Baseline o XAdES Baseline, si son este tipo de firmas las que se quieren realizar.

Se hará uso de un código como el siguiente si se desea generar una firma PAdES con rúbrica.

```
//coordenada horizontal inferior izquierda de la posición de la rúbrica en la pág.
 int lower_left_x = 200
//coordenada vertical inferior izquierda de la posición de la rúbrica en la pág.
 int lower_left_y = 40

//coordenada horizontal superior derecha de la posición de la rúbrica en la pág.
int upper_right_x = 310
//coordenada vertical superior derecha de la posición de la rúbrica en la pág.
int upper_right_y = 80

byte[ ] signaturePAdESRubric =
es.gob.afirma.integraFacade.IntegraFacade.generateSignaturePAdESRubric("array de
bytes del document a firmar", "clave privada del certificado a utilizar", false,
false, "array de byte con la imagen a inserter como rúbrica", "número de página
donde se va a inserter la rúbrica", lower_left_x, lower_left_y, upper_right_x,
upper_right_y);

es.gob.afirma.signature.validation.PDFValidationResultvr =
(es.gob.afirma.signature.validation.PDFValidationResult)IntegraFacade.verifySignat
ure(signaturePAdESRubric, dataToSign);
```

```
                assertTrue(vr.isCorrect());
```

Se hará uso de un código como el siguiente si se desea generar una multi-firma PAdES con rúbrica.

```
//coordenada horizontal inferior izquierda de la posición de la rúbrica en la pág.
 int lower_left_x = 200
//coordenada vertical inferior izquierda de la posición de la rúbrica en la pág.
 int lower_left_y = 40

//coordenada horizontal superior derecha de la posición de la rúbrica en la pág.
int upper_right_x = 310
//coordenada vertical superior derecha de la posición de la rúbrica en la pág.
int upper_right_y = 80

byte[ ] multiSignaturePAdESRubric =
es.gob.afirma.integraFacade.IntegraFacade.generateMultiSignaturePAdESRubric("array
de bytes del document a realizer la multifirma", "clave privada del certificado a
utilizar", false, false, "array de byte con la imagen a inserter como rúbrica",
"número de página donde se va a inserter la rúbrica", lower_left_x, lower_left_y,
upper_right_x, upper_right_y);

es.gob.afirma.signature.validation.PDFValidationResult vr =
(es.gob.afirma.signature.validation.PDFValidationResult)IntegraFacade.verifySignat
ure(multiSignaturePAdESRubric, dataToSign);
              assertTrue(vr.isCorrect());
```

A.5.3 Uso de la interfaz Signer.

A.5.3.1 Formato CAdES

A continuación se muestran ejemplos asociados al formato de firma CAdES mediante el uso de la interfaz Signer de Integr@.
En el siguiente ejemplo se muestra la generación y validación de firmas CAdES-BES y CAdES-EPES:

```
CadesSigner cs = new CadesSigner();

// test con valores válidos (firma explícita)
byte[ ] result = cs.sign("array de bytes del documento a firmar",
SignatureConstants.SIGN_ALGORITHM_SHA512WITHRSA,
SignatureConstants.SIGN_MODE_EXPLICIT, "clave privada del certificado a utilizar",
null, false, SignatureFormatDetector.FORMAT_CADES_BES, null);
// validamos firma
boolean firmaValida = cs.verifySignature(result, "array de bytes del documento a
firmar").isCorrect();

// test con valores válidos (firma implícita)
result = cs.sign("array de bytes del documento a firmar",
SignatureConstants.SIGN_ALGORITHM_SHA512WITHRSA,
SignatureConstants.SIGN_MODE_IMPLICIT, "clave privada del certificado a utilizar",
null, false, SignatureFormatDetector.FORMAT_CADES_BES, null);
// validamos firma
firmaValida = cs.verifySignature(result, null).isCorrect();
```

```
// test con valores válidos (firma implícita con política de firma de AGE)
result = cs.sign("array de bytes del documento a firmar",
SignatureConstants.SIGN_ALGORITHM_SHA1WITHRSA,
SignatureConstants.SIGN_MODE_IMPLICIT, "clave privada del certificado a utilizar",
null, false, SignatureFormatDetector.FORMAT_CADES_EPES, null);
// validamos firma
firmaValida = cs.verifySignature(result, "array de bytes del documento a
firmar").isCorrect();
```

Si se desea obtener firmas con sello de tiempo puede usarse un código como el que se expone a continuación.

```
CadesSigner cadesSigner = new CadesSigner();

/*
* Generación de firma CAdES-T explícita
*/
byte[ ] signature = cadesSigner.sign(getTextDocument(),
SignatureConstants.SIGN_ALGORITHM_SHA1WITHRSA,
SignatureConstants.SIGN_MODE_EXPLICIT, getCertificatePrivateKey(), null, true,
SignatureFormatDetector.FORMAT_CADES_BES, null);


/*
* Generación de firma CAdES-T implícita
*/
byte[ ] signature = cadesSigner.sign(getTextDocument(),
SignatureConstants.SIGN_ALGORITHM_SHA256WITHRSA,
SignatureConstants.SIGN_MODE_IMPLICIT, getCertificatePrivateKey(), null, true,
SignatureFormatDetector.FORMAT_CADES_BES, null);

/*
* Generación de firma CAdES-T implícita con política de firma
*/
cadesSigner.sign(getTextDocument(),
SignatureConstants.SIGN_ALGORITHM_SHA384WITHRSA,
SignatureConstants.SIGN_MODE_IMPLICIT, getCertificatePrivateKey(), null, true,
SignatureFormatDetector.FORMAT_CADES_EPES, "ASN1_AGE_1.9”);
```

Se hará uso de un código como el siguiente si se desea generar una co-firma:

```
CadesSigner cadesSigner = new CadesSigner();

byte[ ] result = cadesSigner.coSign("array de bytes con la firma a co-firmar",
"array de bytes con el documento originalmente firmado",
SignatureConstants.SIGN_ALGORITHM_SHA256WITHRSA, "clave privada del certificado a
utilizar", null, false, SignatureFormatDetector.FORMAT_CADES_BES, null);
```

Se hará uso de un código como el siguiente si se desea generar una contra-firma:

```
CadesSigner cadesSigner = new CadesSigner();

byte[ ] result = cs.counterSign(("array de bytes con la firma a contra-firmar",
SignatureConstants.SIGN_ALGORITHM_SHA512WITHRSA, "clave privada del certificado a
utilizar", null, false, SignatureFormatDetector.FORMAT_CADES_BES, null);
```

Se hará uso de un código como el siguiente si se desea actualizar una firma:

```
CadesSigner cadesSigner = new CadesSigner();
```

```
/*
* Actualización de una firma CAdES-BES implícita sin indicar firmante
*/
byte[ ] upgradedSignature = cadesSigner.upgrade("array de bytes con la firrma a
actualizar", null);

/*
* Actualización de una firma CAdES-BES explícita indicando firmante
*/
List<X509Certificate> listCertificates = new ArrayList<X509Certificate>();
listCertificates.add("certificado a utilizar");

byte[ ] upgradedSignature = cadesSigner.upgrade("array de bytes con la firrma a
actualizar", listCertificates);
```

Se hará uso de un código como el siguiente si se desea obtener información sobre los datos firmados.

```
CadesSigner cadesSigner = new CadesSigner();

/*
* Obtención de los datos firmados de una firma CAdES.
*/
try{
          OriginalSignedData osd = cadesSigner.getSignedData("array de bytes con
la firma de la que se quiere obtener la información");
} catch(SigningException e) {
          e.printStackTrace();
}
```

A.5.3.2 Formato CAdES Baseline

A continuación de describen diferentes ejemplos asociados a la generación, actualización y validación de firmas CAdES Baseline realizados haciendo uso de la fachada Signer:

```
byte[ ] dataToSign = getDataToSign();

es.gob.afirma.signature.cades.CAdESBaselineSigner signer = new
es.gob.afirma.signature.cades.CAdESBaselineSigner();
byte[ ] cadesBLevelSignature = null;
byte[ ] cadesBLevelCoSignature = null;
byte[ ] cadesBLevelCounterSignature = null;
byte[ ] cadesTLevelSignature = null;
byte[ ] cadesTLevelCoSignature = null;
byte[ ] cadesTLevelCounterSignature = null;
java.security.KeyStore.PrivateKeyEntry privateKey = getCertificatePrivateKey();

/*
 * Generación y Validación de firma CAdES B-Level explícita sin política de firma
y algoritmo SHA-256
 */
try {
          cadesBLevelSignature = signer.sign(dataToSign,
es.gob.afirma.signature.SignatureConstants.SIGN_ALGORITHM_SHA256WITHRSA,
es.gob.afirma.signature.SignatureConstants.SIGN_MODE_EXPLICIT, privateKey, null,
```

```
false, es.gob.afirma.signature.ISignatureFormatDetector.FORMAT_CADES_B_LEVEL,
null);
          es.gob.afirma.signature.ValidationResult vr =
signer.verifySignature(cadesBLevelSignature, dataToSign);
} catch (Exception e) {
          e.printStackTrace();
}

/*
 * Actualización de todos los firmantes de una firma CAdES B-Level explícita sin
política de firma y algoritmo SHA-256 a CAdES T-Level, y validación de la misma
 */
try {
          cadesTLevelSignature = signer.upgrade(cadesBLevelSignature, null);
          es.gob.afirma.signature.ValidationResult vr =
signer.verifySignature(cadesTLevelSignature, dataToSign);
} catch (Exception e) {
          e.printStackTrace();
}

/*
 *  Generación y Validación de co-firma CAdES B-Level explícita sin política de
firma y algoritmo SHA-1
 */
try {
          cadesBLevelCoSignature = signer.coSign(cadesBLevelSignature, dataToSign,
es.gob.afirma.signature.SignatureConstants.SIGN_ALGORITHM_SHA1WITHRSA, privateKey,
null, false,
es.gob.afirma.signature.ISignatureFormatDetector.FORMAT_CADES_B_LEVEL, null);
          es.gob.afirma.signature.ValidationResult vr =
signer.verifySignature(cadesBLevelCoSignature, dataToSign);
} catch (Exception e) {
          e.printStackTrace();
}

/*
 *  Generación y Validación de contra-firma CAdES B-Level explícita sin política
de firma y algoritmo SHA-512
 */
try {
          cadesBLevelCounterSignature = signer.counterSign(cadesBLevelSignature,
es.gob.afirma.signature.SignatureConstants.SIGN_ALGORITHM_SHA512WITHRSA,
privateKey, null, false,
es.gob.afirma.signature.ISignatureFormatDetector.FORMAT_CADES_B_LEVEL, null);
          es.gob.afirma.signature.ValidationResult vr =
signer.verifySignature(cadesBLevelCounterSignature, dataToSign);
} catch (Exception e) {
          e.printStackTrace();
}

/*
 * Generación y Validación de firma CAdES B-Level explícita con política de firma
y algoritmo SHA-256
 */
try {
          cadesBLevelSignature = signer.sign(dataToSign,
es.gob.afirma.signature.SignatureConstants.SIGN_ALGORITHM_SHA256WITHRSA,
es.gob.afirma.signature.SignatureConstants.SIGN_MODE_EXPLICIT, privateKey, null,
```

```
false, es.gob.afirma.signature.ISignatureFormatDetector.FORMAT_CADES_B_LEVEL,
"ASN1_AGE_1.9");
          es.gob.afirma.signature.ValidationResult vr =
signer.verifySignature(cadesBLevelSignature, dataToSign);
} catch (Exception e) {
          e.printStackTrace();
}

/*
 * Actualización de todos los firmantes de una firmaCAdES B-Level explícita con
política de firma y algoritmo SHA-256 a CAdES T-Level, y validación de la misma
 */
try {
          cadesTLevelSignature = signer.upgrade(cadesBLevelSignature, null);
          es.gob.afirma.signature.ValidationResult vr =
signer.verifySignature(cadesTLevelSignature, dataToSign);
} catch (Exception e) {
          e.printStackTrace();
}

/*
 *  Generación y Validación de co-firma CAdES B-Level explícita con política de
firma y algoritmo SHA-1
 */
try {
          cadesBLevelCoSignature = signer.coSign(cadesBLevelSignature, dataToSign,
es.gob.afirma.signature.SignatureConstants.SIGN_ALGORITHM_SHA1WITHRSA, privateKey,
null, false,
es.gob.afirma.signature.ISignatureFormatDetector.FORMAT_CADES_B_LEVEL,
"ASN1_AGE_1.9");
          es.gob.afirma.signature.ValidationResult vr =
signer.verifySignature(cadesBLevelCoSignature, dataToSign);
} catch (Exception e) {
          e.printStackTrace();
}

/*
 *  Generación y Validación de contra-firma CAdES B-Level explícita con política
de firma y algoritmo SHA-512
 */
try {
          cadesBLevelCounterSignature = signer.counterSign(cadesBLevelSignature,
es.gob.afirma.signature.SignatureConstants.SIGN_ALGORITHM_SHA512WITHRSA,
privateKey, null, false,
es.gob.afirma.signature.ISignatureFormatDetector.FORMAT_CADES_B_LEVEL,
"ASN1_AGE_1.9");
          es.gob.afirma.signature.ValidationResult vr =
signer.verifySignature(cadesBLevelCoSignature, dataToSign);
} catch (Exception e) {
          e.printStackTrace();
}

/*
 * Generación y Validación de firma CAdES T-Level explícita sin política de firma
y algoritmo SHA-256
 */
try {
          cadesTLevelSignature = signer.sign(dataToSign,
es.gob.afirma.signature.SignatureConstants.SIGN_ALGORITHM_SHA256WITHRSA,
```

```
es.gob.afirma.signature.SignatureConstants.SIGN_MODE_EXPLICIT, privateKey, null,
true, es.gob.afirma.signature.ISignatureFormatDetector.FORMAT_CADES_B_LEVEL,
null);
          es.gob.afirma.signature.ValidationResult vr =
signer.verifySignature(cadesTLevelSignature, dataToSign);
} catch (Exception e) {
          e.printStackTrace();
}

/*
 *  Generación y Validación de co-firma CAdES T-Level explícita sin política de
firma y algoritmo SHA-1
 */
try {
          cadesTLevelCoSignature = signer.coSign(cadesTLevelSignature, dataToSign,
es.gob.afirma.signature.SignatureConstants.SIGN_ALGORITHM_SHA1WITHRSA, privateKey,
null, true, es.gob.afirma.signature.ISignatureFormatDetector.FORMAT_CADES_B_LEVEL,
null);
          es.gob.afirma.signature.ValidationResult vr =
signer.verifySignature(cadesTLevelCoSignature, dataToSign);
} catch (Exception e) {
          e.printStackTrace();
}

/*
 *  Generación y Validación de contra-firma CAdES T-Level explícita sin política
de firma y algoritmo SHA-512
 */
try {
          cadesTLevelCounterSignature = signer.counterSign(cadesTLevelSignature,
es.gob.afirma.signature.SignatureConstants.SIGN_ALGORITHM_SHA512WITHRSA,
privateKey, null, true,
es.gob.afirma.signature.ISignatureFormatDetector.FORMAT_CADES_B_LEVEL, null);
          es.gob.afirma.signature.ValidationResult vr =
signer.verifySignature(cadesTLevelCounterSignature, dataToSign);
} catch (Exception e) {
          e.printStackTrace();
}

/*
 * Generación y Validación de firma CAdES T-Level explícita con política de firma
y algoritmo SHA-256
 */
try {
          cadesTLevelSignature = signer.sign(dataToSign,
es.gob.afirma.signature.SignatureConstants.SIGN_ALGORITHM_SHA256WITHRSA,
es.gob.afirma.signature.SignatureConstants.SIGN_MODE_EXPLICIT, privateKey, null,
true, es.gob.afirma.signature.ISignatureFormatDetector.FORMAT_CADES_B_LEVEL,
"ASN1_AGE_1.9");
          es.gob.afirma.signature.ValidationResult vr =
signer.verifySignature(cadesTLevelSignature, dataToSign);
} catch (Exception e) {
          e.printStackTrace();
}

/*
 *  Generación y Validación de co-firma CAdES T-Level explícita con política de
firma y algoritmo SHA-1
 */
```

```
try {
          cadesTLevelCoSignature = signer.coSign(cadesTLevelSignature, dataToSign,
es.gob.afirma.signature.SignatureConstants.SIGN_ALGORITHM_SHA1WITHRSA, privateKey,
null, true, es.gob.afirma.signature.ISignatureFormatDetector.FORMAT_CADES_B_LEVEL,
"ASN1_AGE_1.9");
          es.gob.afirma.signature.ValidationResult vr =
signer.verifySignature(cadesTLevelCoSignature, dataToSign);
} catch (Exception e) {
          e.printStackTrace();
}

/*
 *  Generación y Validación de contra-firma CAdES T-Level explícita con política
de firma y algoritmo SHA-512
 */
try {
          cadesTLevelCounterSignature = signer.counterSign(cadesTLevelSignature,
es.gob.afirma.signature.SignatureConstants.SIGN_ALGORITHM_SHA512WITHRSA,
privateKey, null, true,
es.gob.afirma.signature.ISignatureFormatDetector.FORMAT_CADES_B_LEVEL,
"ASN1_AGE_1.9");
          es.gob.afirma.signature.ValidationResult vr =
signer.verifySignature(cadesTLevelCoSignature, dataToSign);
} catch (Exception e) {
          e.printStackTrace();
}
```

Se hará uso de un código como el siguiente si se desea obtener información sobre los datos firmados.

```
CAdESBaselineSigner signer = new CAdESBaselineSigner();
/*
* Obtención de los datos firmados de una firma CAdES Baseline.
*/
try{
          OriginalSignedData osd = signer.getSignedData("array de bytes con la
firma de la que se quiere obtener la información");
} catch(SigningException e) {
          e.printStackTrace();
}
```

A.5.3.3 Formato PAdES

A continuación se muestran ejemplos asociados al formato de firma PAdES mediante el uso de la interfaz Signer de Integr@.

En el siguiente ejemplo se muestra la generación y validación de firmas PAdES-BES y PAdES-EPES:

```
PadesSigner ps = new PadesSigner();

//firma implícita
byte[ ] result = ps.sign("array de bytes con el fichero pdf a firma",
SignatureConstants.SIGN_ALGORITHM_SHA512WITHRSA,
SignatureConstants.SIGN_MODE_IMPLICIT, "clave privada del certificado a utilizar",
null, false, SignatureFormatDetector.FORMAT_PADES_BES, null);
boolean firmaValida = ps.verifySignature(result).isCorrect();
```

```
// firma implícita con política de firma de AGE
Properties extraParams = new Properties();

extraParams.put(SignatureProperties.PADES_CONTACT_PROP, "Ricoh");
extraParams.put(SignatureProperties.PADES_LOCATION_PROP, "Seville");
extraParams.put(SignatureProperties.PADES_REASON_PROP, "Document signed for
demonstrate this authenticity");

result = ps.sign("array de bytes con el fichero pdf a firma",
SignatureConstants.SIGN_ALGORITHM_SHA512WITHRSA,
SignatureConstants.SIGN_MODE_IMPLICIT, "clave privada del certificado a utilizar",
extraParams, false, SignatureFormatDetector.FORMAT_PADES_EPES, null);
firmaValida = ps.verifySignature(result).isCorrect();
```

Si se desea obtener firmas con sello de tiempo puede usarse un código como el que se expone a continuación:

```
PadesSigner padesSigner = new PadesSigner();

// Obtenemos el fichero que se va a sellar
byte[ ] file = UtilsFileSystemCommons.readFile("pdfToSign.pdf", true);

/*
* Generación de firma PAdES-T
*/
byte[ ] padesBES = padesSigner.sign(file,
SignatureConstants.SIGN_ALGORITHM_SHA1WITHRSA,
SignatureConstants.SIGN_MODE_IMPLICIT, "clave privada del certificado a utilizar",
null, true, SignatureFormatDetector.FORMAT_PADES_BES, null);

/*
* Generación de firma PAdES-T con política de firma
*/
byte[ ] padesEPES = padesSigner.sign(file,
SignatureConstants.SIGN_ALGORITHM_SHA256WITHRSA,
SignatureConstants.SIGN_MODE_IMPLICIT, "clave privada del certificado a utilizar",
null, true, SignatureFormatDetector.FORMAT_PADES_EPES, "PDF_AGE_1.9");
```

Se hará uso de un código como el siguiente si se desea actualizar una firma:

```
PadesSigner padesSigner = new PadesSigner();
/*
* Actualización de todos los firmantes de una firma PAdES-BES
*/
byte[ ] upgradedSignature = padesSigner.upgrade("array de bytes con la firma pdf a
actualizar", null);

/*
* Actualización de un firmante de una firma PAdES-BES
*/
List<X509Certificate> listCertificates = new ArrayList<X509Certificate>();
listCertificates.add("certificado a utilizar");

byte[ ] upgradedSignature = padesSigner.upgrade("array de bytes con la firma pdf a
actualizar",, listCertificates);
```

Se hará uso de un código como el siguiente si se desea obtener información sobre los datos firmados.

```
PadesSigner padesSigner = new PadesSigner();
/*
* Obtención de los datos firmados de una firma PAdES.
*/
try{
          OriginalSignedData osd = padesSigner.getSignedData("array de bytes con
la firma de la que se quiere obtener la información");
} catch(SigningException e) {
          e.printStackTrace();
}
```

Se hará uso de un código como el siguiente si se desea generar una co-firma PAdES.

```
PadesSigner padesSigner = new PadesSigner();
/*
* Generación de una co-firma.
*/
Properties extraParams = new Properties();

extraParams.put(SignatureProperties.PADES_CONTACT_PROP, "Ricoh");
extraParams.put(SignatureProperties.PADES_LOCATION_PROP, "Seville");
extraParams.put(SignatureProperties.PADES_REASON_PROP, "Document signed for
demonstrate this authenticity");

try{
result = padesSigner.coSign("array de bytes con el fichero pdf donde se va a
realizar la multi-firma", null, SignatureConstants.SIGN_ALGORITHM_SHA512WITHRSA,"
clave privada del certificado a utilizar" , extraParams, false,
SignatureFormatDetector.FORMAT_PADES_BES, null);
}catch(SigningException e) {
          e.printStackTrace();
}
```

Se hará uso de un código como el siguiente si se desea generar una firma PAdES con rúbrica.

```
PadesSigner padesSigner = new PadesSigner();
/*
* Generación de una firma PAdES con rúbrica.
*/
Properties extraParams = new Properties();

extraParams.put(SignatureProperties.PADES_CONTACT_PROP, "Ricoh");
extraParams.put(SignatureProperties.PADES_LOCATION_PROP, "Seville");
extraParams.put(SignatureProperties.PADES_REASON_PROP, "Document signed for
demonstrate this authenticity");

extraParams.put(SignatureProperties.PADES_IMAGE, "imagen a insertar codificada en
Base64);

//la rúbrica se insertará en la última página del documento.
extraParams.put(SignatureProperties.PADES_IMAGE_PAGE, "-1");
extraParams.put(SignatureProperties.PADES_LOWER_LEFT_X, "20");
extraParams.put(SignatureProperties.PADES_LOWER_LEFT_Y, "40");
extraParams.put(SignatureProperties.PADES_UPPER_RIGHT_X, "250");
extraParams.put(SignatureProperties.PADES_UPPER_RIGHT_Y, "150");
```

```
try{
result = padesSigner.sign((“array de bytes con el fichero pdf donde se va a
realizar la firma”, SignatureConstants.SIGN_ALGORITHM_SHA512WITHRSA,
SignatureConstants.SIGN_MODE_IMPLICIT,”clave privada del certificado a utilizar”,
extraParams, false, SignatureFormatDetector.FORMAT_PADES_B_LEVEL, null);

          es.gob.afirma.signature.validation.ValidationResult vr =
padesSigner.verifySignature(result);
}catch(SigningException e) {
          e.printStackTrace();
}
```

Se hará uso de un código como el siguiente si se desea generar una multi -firma PAdES con rúbrica.

```
PadesSigner padesSigner = new PadesSigner();
/*
* Generación de una firma PAdES con rúbrica.
*/
Properties extraParams = new Properties();

extraParams.put(SignatureProperties.PADES_CONTACT_PROP, "Ricoh");
extraParams.put(SignatureProperties.PADES_LOCATION_PROP, "Seville");
extraParams.put(SignatureProperties.PADES_REASON_PROP, "Document signed for
demonstrate this authenticity");

extraParams.put(SignatureProperties.PADES_IMAGE, “imagen a insertar codificada en
Base64);
extraParams.put(SignatureProperties.PADES_IMAGE_PAGE, "1");
extraParams.put(SignatureProperties.PADES_LOWER_LEFT_X, "20");
extraParams.put(SignatureProperties.PADES_LOWER_LEFT_Y, "40");
extraParams.put(SignatureProperties.PADES_UPPER_RIGHT_X, "250");
extraParams.put(SignatureProperties.PADES_UPPER_RIGHT_Y, "150");

try{
result = padesSigner.coSign(“array de bytes con el fichero pdf donde se va a
realizar la multi-firma, null, SignatureConstants.SIGN_ALGORITHM_SHA512WITHRSA,
“clave privada del certificado a utilizar”, extraParams, false,
SignatureFormatDetector.FORMAT_PADES_B_LEVEL, null);
          es.gob.afirma.signature.validation.ValidationResult vr =
padesSigner.verifySignature(result);
          assertTrue(vr.isCorrect());
}catch(SigningException e) {
          e.printStackTrace();
}
```

A.5.3.4 Formato PAdES Baseline

A continuación de describen diferentes ejemplos asociados a la generación, actualización y validación de firmas PAdES Baseline realizados haciendo uso de la fachada Signer:

```
byte[ ] dataToSign = getDataToSign();
```

```
es.gob.afirma.signature.pades.PAdESBaselineSigner signer = new
es.gob.afirma.signature.pades.PAdESBaselineSigner();
byte[ ] padesBLevelSignature = null;
byte[ ] padesTLevelSignature = null;
es.gob.afirma.signature.validation.PDFValidationResult vr = null;
java.security.KeyStore.PrivateKeyEntry privateKey = getCertificatePrivateKey();
java.util.Properties extraParams = new java.util.Properties();
extraParams.put(es.gob.afirma.signature.SignatureProperties.PADES_CONTACT_PROP,
"Ricoh");
extraParams.put(es.gob.afirma.signature.SignatureProperties.PADES_LOCATION_PROP,
"Seville");
extraParams.put(es.gob.afirma.signature.SignatureProperties.PADES_REASON_PROP,
"Document signed for tests");
extraParams.put(es.gob.afirma.signature.SignatureProperties.PADES_CERTIFICATION_LE
VEL, es.gob.afirma.signature.SignatureConstants.PDF_APPROVAL);

/*
 * Generación y Validación de firma PAdES B-Level explícita sin política de firma
y algoritmo SHA-1
 */
try {
         padesBLevelSignature = signer.sign(dataToSign,
es.gob.afirma.signature.SignatureConstants.SIGN_ALGORITHM_SHA1WITHRSA,
es.gob.afirma.signature.SignatureConstants.SIGN_MODE_EXPLICIT, privateKey,
extraParams, false,
es.gob.afirma.signature.ISignatureFormatDetector.FORMAT_PADES_B_LEVEL, null);
         vr = signer.verifySignature(padesBLevelSignature);
} catch (Exception e) {
         e.printStackTrace();
}

/*
 * Actualización de firma PAdES B-Level sin política de firma a PAdES T-Level
 */
try {
         padesTLevelSignature = signer.upgrade(padesBLevelSignature, null);
         vr = signer.verifySignature(padesTLevelSignature);
} catch (Exception e) {
         e.printStackTrace();
}

/*
 * Generación y Validación de firma PAdES B-Level implícita con política de firma
y algoritmo SHA-256
 */
try {
         padesBLevelSignature = signer.sign(dataToSign,
es.gob.afirma.signature.SignatureConstants.SIGN_ALGORITHM_SHA256WITHRSA,
es.gob.afirma.signature.SignatureConstants.SIGN_MODE_IMPLICIT, privateKey,
extraParams, false,
es.gob.afirma.signature.ISignatureFormatDetector.FORMAT_PADES_B_LEVEL,
"PDF_AGE_1.9");
         vr = signer.verifySignature(padesBLevelSignature);
} catch (Exception e) {
         e.printStackTrace();
}

/*
```

```
 * Actualización de firma PAdES B-Level con política de firma a PAdES T-Level
 */
try {
          padesTLevelSignature = signer.upgrade(padesBLevelSignature, null);
          vr = signer.verifySignature(padesTLevelSignature);
} catch (Exception e) {
          e.printStackTrace();
}

/*
 * Generación y Validación de firma PAdES T-Level explícita sin política de firma
y algoritmo SHA-384
 */
try {
          padesTLevelSignature = signer.sign(dataToSign,
es.gob.afirma.signature.SignatureConstants.SIGN_ALGORITHM_SHA384WITHRSA,
es.gob.afirma.signature.SignatureConstants.SIGN_MODE_EXPLICIT, privateKey,
extraParams, true,
es.gob.afirma.signature.ISignatureFormatDetector.FORMAT_PADES_B_LEVEL, null);
          vr = signer.verifySignature(padesTLevelSignature);
} catch (Exception e) {
          e.printStackTrace();
}

/*
 * Generación y Validación de firma PAdES T-Level explícita con política de firma
y algoritmo SHA-512
 */
try {
          padesTLevelSignature = signer.sign(dataToSign,
es.gob.afirma.signature.SignatureConstants.SIGN_ALGORITHM_SHA512WITHRSA,
es.gob.afirma.signature.SignatureConstants.SIGN_MODE_EXPLICIT, privateKey,
extraParams, true,
es.gob.afirma.signature.ISignatureFormatDetector.FORMAT_PADES_B_LEVEL,
"PDF_AGE_1.9");
          vr = signer.verifySignature(padesTLevelSignature);
} catch (Exception e) {
          e.printStackTrace();
}
```

Se hará uso de un código como el siguiente si se desea obtener información sobre los datos firmados.

```
PAdESBaselineSigner padesBaselineSigner = new PAdESBaselineSigner ();
/*
* Obtención de los datos firmados de una firma PAdES Baseline.
*/
try{
          OriginalSignedData osd = padesBaselineSigner.getSignedData("array de
bytes con la firma de la que se quiere obtener la información");
} catch(SigningException e) {
          e.printStackTrace();
}
```

Se hará uso de un código como el siguiente si se desea generar una co-firma PAdES Baseline.

```
PadesBaselineSigner pbs = new PadesBaselineSigner();
```

```
/*
* Generación de una co-firma.
*/
Properties extraParams = new Properties();

extraParams.put(SignatureProperties.PADES_CONTACT_PROP, "Ricoh");
extraParams.put(SignatureProperties.PADES_LOCATION_PROP, "Seville");
extraParams.put(SignatureProperties.PADES_REASON_PROP, "Document signed for
demonstrate this authenticity");

try{
result = pbs.coSign("array de bytes con el fichero pdf donde se va a realizar la
multi-firma", null, SignatureConstants.SIGN_ALGORITHM_SHA512WITHRSA," clave
privada del certificado a utilizar" , extraParams, false,
SignatureFormatDetector.FORMAT_PADES_BES, null);
}catch(SigningException e) {
         e.printStackTrace();
}
```

Se hará uso de un código como el siguiente si se desea generar una firma PAdES Baseline con rúbrica.

```
PadesBaselineSigner pbs = new PadesBaselineSigner();
/*
* Generación de una firma PAdES con rúbrica.
*/
Properties extraParams = new Properties();

extraParams.put(SignatureProperties.PADES_CONTACT_PROP, "Ricoh");
extraParams.put(SignatureProperties.PADES_LOCATION_PROP, "Seville");
extraParams.put(SignatureProperties.PADES_REASON_PROP, "Document signed for
demonstrate this authenticity");

extraParams.put(SignatureProperties.PADES_IMAGE, "imagen a insertar codificada en
Base64);

//la rúbrica se insertará en la última página del documento.
extraParams.put(SignatureProperties.PADES_IMAGE_PAGE, "-1");
extraParams.put(SignatureProperties.PADES_LOWER_LEFT_X, "20");
extraParams.put(SignatureProperties.PADES_LOWER_LEFT_Y, "40");
extraParams.put(SignatureProperties.PADES_UPPER_RIGHT_X, "250");
extraParams.put(SignatureProperties.PADES_UPPER_RIGHT_Y, "150");

try{
result = pbs.sign(("array de bytes con el fichero pdf donde se va a realizar la
multi-firma", SignatureConstants.SIGN_ALGORITHM_SHA512WITHRSA,
SignatureConstants.SIGN_MODE_IMPLICIT,"clave privada del certificado a utilizar",
extraParams, false, SignatureFormatDetector.FORMAT_PADES_B_LEVEL, null);

         es.gob.afirma.signature.validation.PDFValidationResult vr =
pbs.verifySignature(result);
}catch(SigningException e) {
         e.printStackTrace();
}
```

Se hará uso de un código como el siguiente si se desea generar una multi -firma PAdES Baseline con rúbrica.

```
PadesBaselineSigner pbs = new PadesBaselineSigner();
```

```
/*
* Generación de una firma PAdES con rúbrica.
*/
Properties extraParams = new Properties();

extraParams.put(SignatureProperties.PADES_CONTACT_PROP, "Ricoh");
extraParams.put(SignatureProperties.PADES_LOCATION_PROP, "Seville");
extraParams.put(SignatureProperties.PADES_REASON_PROP, "Document signed for
demonstrate this authenticity");

extraParams.put(SignatureProperties.PADES_IMAGE, "imagen a insertar codificada en
Base64);
extraParams.put(SignatureProperties.PADES_IMAGE_PAGE, "1");
extraParams.put(SignatureProperties.PADES_LOWER_LEFT_X, "20");
extraParams.put(SignatureProperties.PADES_LOWER_LEFT_Y, "40");
extraParams.put(SignatureProperties.PADES_UPPER_RIGHT_X, "250");
extraParams.put(SignatureProperties.PADES_UPPER_RIGHT_Y, "150");

try{
result = pbs.coSign("array de bytes con el fichero pdf donde se va a realizar la
multi-firma, null, SignatureConstants.SIGN_ALGORITHM_SHA512WITHRSA, "clave privada
del certificado a utilizar", extraParams, false,
SignatureFormatDetector.FORMAT_PADES_B_LEVEL, null);
          es.gob.afirma.signature.validation.PDFValidationResultvr =
pbs.verifySignature(result);
          assertTrue(vr.isCorrect());
}catch(SigningException e) {
          e.printStackTrace();
}
```

A.6 Ejemplos de Uso de los Servicios para el tipo de integración 5

En este anexo se incluyen algunos ejemplos de uso relacionados con la generación, validación y actualización de firmas XAdES (BES y EPES) realizadas por Integr@, así como la obtención de información de los datos originalmente firmados de firmas ASiC-S.

Hay que señalar que todas las operaciones y por lo tanto **los ejemplos expuestos para el resto de tipos de intgración pueden realizarse también con este tipo de integración**.

A.6.1 Configuración general.

Para este tipo de integración es necesario hacer uso de las librerías de Integr@ y librerías de terceros enumeradas en el apartado 7.6.

Por otro lado, será necesario disponer en una ruta accesible desde la aplicación en la que se usa Integr@ de los ficheros de configuración necesarios para este caso concreto y que pueden encontrarse en el entregable. Puenden verse los ficheros de propiedades necesarios en el apartado 8.4.

Para el caso que nos ocupa damos por hecho que se encuentra desplegada la plataforma de TS@ con una aplicación "pruebasTest" dada de alta y correctamente configurada. De igual forma, si en el fichero **integra.properties** cuya configuración se detalla a continuación se opta por realizar validaciones de estado de revocación de certificados (CERTIFICATE_VALIDATION_LEVEL = 2), será necesario tener desplegada la plataforma de @Firma con una aplicación "afirmaTest" creada y correctamente configurada con el acceso mediante ocsp.

A continuación, se procederá a configurar el lenguaje de los mensajes de Integr@ editando el archivo de propiedades **Language.properties**:

```
LANGUAGE = es_ES
```

Con esta configuración la plataforma emitirá los mensajes en español.

Se procederá a configurar el archivo **integra.properties**:

```
CERTIFICATE_VALIDATION_LEVEL = 1
TSA_APP_ID = pruebasTest
TSA_COMMUNICATION_TYPE = DSS
TSA_TIMESTAMP_TYPE = XML
```

Con esta configuración se extrae que, tanto el certificado firmante a usar en la generación de la firma XAdES, como el certificado firmante del sello de tiempo, se validarán respecto al periodo de validez y caducidad; y que Integr@ se comunicará con el Servicio de Generación de Sello de Tiempo con protocolo DSS de TS@ para obtener un sello de tiempo XML. Si se configura el parámetro `CERTIFICATE_VALIDATION_LEVEL` con valor 2 será necesario añadir al archivo **ingegra.properties** con los datos de acceso al servicio ocsp de validación de certificados de @Firma (con la misma configuración que se detalla en el ejemplo A.2.1).

Se procederá también a configurar el fichero **integra.properties** para establecer las propiedades de la política de firma a utilizar en las firmas XML en la generación de firmas XAdES-EPES:

Hay que señalar que si se desean realizar firmas CAdES y PAdES EPES habría que combinar el contenido de este fichero con el expuesto en el apartado A.5.1,

```
#Listado de URIs para algoritmos de hash en firmas XML.
XML_HASH-SHA1 = http://www.w3.org/2000/09/xmldsig#sha1
XML_HASH-SHA256 = http://www.w3.org/2001/04/xmlenc#sha256
XML_HASH-SHA512 = http://www.w3.org/2001/04/xmlenc#sha512
#Listado de URIs para algoritmos de firma en firmas XML.
XML_SIGN_HASH-SHA1WithRSA = http://www.w3.org/2000/09/xmldsig#rsa-sha1
XML_SIGN_HASH-SHA256WithRSA = http://www.w3.org/2001/04/xmldsig-more#rsa-sha256
XML_SIGN_HASH-SHA512WithRSA = http://www.w3.org/2001/04/xmldsig-more#rsa-sha512
#Identificador de la política de firma a usar en la generación de firmas XML.
XML_POLICY_ID = XML_AGE_1.9_URL
#Identificador de la política de firma.
XML_AGE_1.9_URL-IDENTIFIER_XML =
http://administracionelectronica.gob.es/es/ctt/politicafirma/politica_firma_AGE_v1
_9.pdf
#Algoritmo de resumen a usar para calcular la huella digital del documento legible
de política de firma.
```

```
XML_AGE_1.9_URL-HASH_ALGORITHM = SHA-1
#Valor del hash de la política de firma.
XML_AGE_1.9_URL-HASH_VALUE = 7SxX3erFuH31TvAw9LZ70N7p1vA=
#Algoritmos de resumen válidos.
XML_AGE_1.9_URL-ALLOWED_HASH_ALGORITHM = XML_HASH-SHA1,XML_HASH-SHA256,XML_HASH-
SHA512
#Algoritmos de firma válidos.
XML_AGE_1.9_URL-ALLOWED_SIGN_ALGORITHM = XML_SIGN_HASH-SHA1WithRSA,XML_SIGN_HASH-
SHA256WithRSA,XML_SIGN_HASH-SHA512WithRSA
#Descripción de la política de firma
XML_AGE_1.9_URL-DESCRIPTION = Política de Firma Electrónica y de Certificados de
la Administración General del Estado 1.9 para firmas XAdES (URL)
#Elementos firmados obligatorios.
XML_AGE_1.9_URL-MANDATORY_SIGNED_ELEMENTS =
SigningTime,SigningCertificate,SignaturePolicyIdentifier,DataObjectFormat
#Elementos firmados opcionales.
XML_AGE_1.9_URL-OPTIONAL_SIGNED_ELEMENTS =
SignatureProductionPlace,SignerRole,CommitmentTypeIndication,AllDataObjectsTimeSta
mp,IndividualDataObjectsTimeStamp
#Elementos no firmados opcionales.
XML_AGE_1.9_URL-OPTIONAL_UNSIGNED_ELEMENTS = CounterSignature
#Elementos hijos obligatorios.
XML_AGE_1.9_URL-[SignerRole]-REQUIRED_CHILD = ClaimedRoles|CertifiedRoles
#Valor obligatorio de elemento.
XML_AGE_1.9_URL-[ClaimedRoles]-REQUIRED_VALUE =
supplier|emisor|customer|receptor|third party|tercero
#Modos de firma permitidos.
XML_AGE_1.9_URL-ALLOWED_SIGNING_MODES = Detached,Enveloped,Enveloping
```

Con esta configuración se aplicarían las restricciones asociadas a la Política de Firma Electrónica y de Certificados de la Administración General del Estado 1.9 en lo que se refiere a firmas XAdES.

Si se va a hacer uso de la fachada de Integr@, como en el caso del ejemplo A.5.2, será necesario editar el fichero **integra.properties** añadiendo valores como los siguientes.

```
FACADE_SIGNATURE_ALGORITHM = SHA256withRSA
FACADE_SIGNATURE_TYPE = XAdES
```

Si se van a realizar peticiones de sello de tiempo a TS@ mediante RFC 3161 será necesario añadir al archivo de propiedades **integra.properties**, los parámetros generales de acceso a TS@. Para el ejemplo que nos ocupa la configuración sería la siguiente.

```
com.trustedstorePath = D:/workspace-baseJdk1.8/Integra-parent/Integra-
ws/src/test/resources/truststoreWS.jks

com.trustedstorePassword = 12345

com.serviceWSDLPath = file:/D:/workspace-baseJdk1.8/Integra-parent/Integra-
ws/src/test/resources/TimeStampWS.wsdl
```

También será necesario crear y configurar el fichero **tsapruebasTest.properties** con la siguiente configuración para el ejemplo que nos ocupa.

```
callTimeout =20000
authorizationMethod =UserNameToken
UserNameToken.userName = user
UserNameToken.userPassword = 12345
request.symmetricKey.use =false
renewTimeStampWS.validationLevel = 0
```

Finalmente sera necesario mapear los ficheros de configuración dinámicos en **mappingFiles.properties**.

```
tsapruebasTest = tsapruebasTest.properties
integra = integra.properties
```

A.6.2 Fachada de Integr@: Generación, validación y actualización de firma, co-firma y contrafirma.

En el ejemplo que se expone a continuación se realizan tres bloques de firmas, BES, EPES y BES con sello de tiempo, de forma que en cada uno de los cuales, se realizan firmas, co-firmas, contra-firmas, actualizaciones y validaciones.

```
byte[ ] dataToSign =
UtilsFileSystemCommons.getArrayByteFileBase64Encoded("ficheroAfirmar.xml", true);
PrivateKeyEntry privateKey = "clave privada del certificado a utilizar";

/*
 * AdES-BES
*/
byte[ ] signature = IntegraFacade.generateSignature(dataToSign, privateKey, false,
false);

es.gob.afirma.signature.validation.ValidationResult vr =
(es.gob.afirma.signature.validation.ValidationResult)IntegraFacade.verifySignature
(signature, dataToSign);

byte[ ] coSignature = IntegraFacade.generateCoSignature(signature, dataToSign,
privateKey, false, false);
vr =
(es.gob.afirma.signature.validation.ValidationResult)IntegraFacade.verifySignature
(coSignature, dataToSign);

byte[ ] counterSignature = IntegraFacade.generateCounterSignature(signature,
privateKey, false, false);
vr =
(es.gob.afirma.signature.validation.ValidationResult)IntegraFacade.verifySignature
(counterSignature, dataToSign);

byte[ ] upgradedSignature = IntegraFacade.upgradeSignature(signature, null);
vr =
(es.gob.afirma.signature.validation.ValidationResult)IntegraFacade.verifySignature
(upgradedSignature, dataToSign);
```

```
/*
* AdES-EPES
*/
byte[ ] signature = IntegraFacade.generateSignature(dataToSign, privateKey, true,
false);

es.gob.afirma.signature.validation.ValidationResult vr =
(es.gob.afirma.signature.validation.ValidationResult)IntegraFacade.verifySignature
(signature, dataToSign);

byte[ ] coSignature = IntegraFacade.generateCoSignature(signature, dataToSign,
privateKey, true, false);
vr =
(es.gob.afirma.signature.validation.ValidationResult)IntegraFacade.verifySignature
(coSignature, dataToSign);

byte[ ] counterSignature = IntegraFacade.generateCounterSignature(signature,
privateKey, true, false);
vr =
(es.gob.afirma.signature.validation.ValidationResult)IntegraFacade.verifySignature
(counterSignature, dataToSign);

byte[ ] upgradedSignature = IntegraFacade.upgradeSignature(signature, null);
vr =
(es.gob.afirma.signature.validation.ValidationResult)IntegraFacade.verifySignature
(upgradedSignature, dataToSign);

/*
* AdES-BES con sello de tiempo
*/
byte[ ] signature = IntegraFacade.generateSignature(dataToSign, privateKey, false,
true);

es.gob.afirma.signature.validation.ValidationResult vr =
(es.gob.afirma.signature.validation.ValidationResult)IntegraFacade.verifySignature
(signature, dataToSign);

byte[ ] coSignature = IntegraFacade.generateCoSignature(signature, dataToSign,
privateKey, false, true);
vr =
(es.gob.afirma.signature.validation.ValidationResult)IntegraFacade.verifySignature
(coSignature, dataToSign);

byte[ ] counterSignature = IntegraFacade.generateCounterSignature(signature,
privateKey, false, true);
vr =
(es.gob.afirma.signature.validation.ValidationResult)IntegraFacade.verifySignature
(counterSignature, dataToSign);

byte[ ] upgradedSignature = IntegraFacade.upgradeSignature(signature, null);
```

A.6.3 Uso de la interfaz Signer.

A.6.3.1 Formato XAdES

A continuación se muestran ejemplos asociados al formato de firma XAdES mediante el uso de la interfaz Signer de Integr@.

En el siguiente ejemplo se muestra la generación y validación de firmas XAdES-BES y XAdES-EPES:

```
//Firmas binarias enveloping
XadesSigner xadesSign = new XadesSigner();
byte[ ] dataToSign = "documento de texto a firmar";
byte[ ] signature = xadesSign.sign(dataToSign,
SignatureConstants.SIGN_ALGORITHM_SHA384WITHRSA,
SignatureConstants.SIGN_FORMAT_XADES_ENVELOPING, getCertificatePrivateKey(),
getDataFormatParams(), false, SignatureFormatDetector.FORMAT_XADES_BES, null);
// validamos firma
boolean firmaValida = xadesSign.verifySignature(signature).isCorrect();

//Firmas XML enveloping
XadesSigner xadesSign = new XadesSigner();
byte[ ] dataToSign = "documento xml a firmar";

byte[ ] signature = xadesSign.sign(dataToSign,
SignatureConstants.SIGN_ALGORITHM_SHA384WITHRSA,
SignatureConstants.SIGN_FORMAT_XADES_ENVELOPING, getCertificatePrivateKey(), null,
false, SignatureFormatDetector.FORMAT_XADES_BES, null);
// validamos firma
boolean firmaValida = xadesSign.verifySignature(signature).isCorrect();

//Firmas binarias enveloped
byte[ ] signature = new XadesSigner().sign("documento de texto a firmar",
SignatureConstants.SIGN_ALGORITHM_SHA1WITHRSA,
SignatureConstants.SIGN_FORMAT_XADES_ENVELOPED, getCertificatePrivateKey(), null,
false, SignatureFormatDetector.FORMAT_XADES_BES, null);
// validamos firma
boolean firmaValida = xadesSign.verifySignature(signature).isCorrect();

//Firmas XML enveloped
XadesSigner xadesSign = new XadesSigner();

byte[ ] signature = xadesSign.sign("documento xml a firmar",
SignatureConstants.SIGN_ALGORITHM_SHA1WITHRSA,
SignatureConstants.SIGN_FORMAT_XADES_ENVELOPED, getCertificatePrivateKey(),
getDataFormatParams(), false, SignatureFormatDetector.FORMAT_XADES_BES, null);
// validamos firma
boolean firmaValida = xadesSign.verifySignature(signature).isCorrect();

//Firmas binarias detached
XadesSigner xadesSign = new XadesSigner();

byte[ ] signature = xadesSign.sign("documento de texto a firmar",
SignatureConstants.SIGN_ALGORITHM_SHA1WITHRSA,
SignatureConstants.SIGN_FORMAT_XADES_DETACHED, getCertificatePrivateKey(),
getDataFormatParams(), false, SignatureFormatDetector.FORMAT_XADES_BES, null);
// validamos firma
boolean firmaValida = xadesSign.verifySignature(signature).isCorrect();

//Firmas XML detached
XadesSigner xadesSign = new XadesSigner();

byte[ ] signature = xadesSign.sign("documento xml a firmar",
SignatureConstants.SIGN_ALGORITHM_SHA1WITHRSA,
```

```
SignatureConstants.SIGN_FORMAT_XADES_DETACHED, getCertificatePrivateKey(), null,
false, SignatureFormatDetector.FORMAT_XADES_BES, null);
// validamos firma
boolean firmaValida = xadesSign.verifySignature(signature).isCorrect();


//Firmas externally detached
XadesSigner xadesSign = new XadesSigner();

// Creamos listado de propiedades adicionales que incluirá el objeto
// manifest con todas las referencias externas.
Properties extraParams = new Properties();
ReferenceData rd = new ReferenceData("http://www.w3.org/2000/09/xmldsig#sha1",
"zyjp8GJOX69990Kkqw8ioPXGExk=");
String xPath = "self::text()[ancestor-or-self::node()=/Class/e[1]]";
TransformData transform = rd.new
TransformData("http://www.w3.org/2000/09/xmldsig#base64", null);
TransformData transform2 = rd.new TransformData("http://www.w3.org/TR/1999/REC-
xpath-19991116", Collections.singletonList(xPath));
List<TransformData> transformList = new ArrayList<TransformData>(2);
transformList.add(transform);
transformList.add(transform2);
rd.setTransforms(transformList);
rd.setId("idAttribute");
rd.setType("typeAttribute");
rd.setUri("uriAttribute");
List<ReferenceData> rdlist = Collections.singletonList(rd);
extraParams.put(SignatureConstants.MF_REFERENCES_PROPERTYNAME, rdlist);

byte[ ] signature = xadesSign.sign(null,
SignatureConstants.SIGN_ALGORITHM_SHA1WITHRSA,
SignatureConstants.SIGN_FORMAT_XADES_EXTERNALLY_DETACHED,
getCertificatePrivateKey(), extraParams, false,
SignatureFormatDetector.FORMAT_XADES_BES, null);
// validamos firma
boolean firmaValida = xadesSign.verifySignature(signature).isCorrect();

//Firmas XML EPES
XadesSigner xadesSign = new XadesSigner();
byte[ ] dataToSign = "documento xml a firmar";

Properties optionalParams = getDataFormatParams();
optionalParams.putAll(getPolicyParams());

byte[ ] signature = xadesSign.sign(dataToSign,
SignatureConstants.SIGN_ALGORITHM_SHA1WITHRSA,
SignatureConstants.SIGN_FORMAT_XADES_ENVELOPING, getCertificatePrivateKey(),
optionalParams, false, SignatureFormatDetector.FORMAT_XADES_EPES,
"XML_AGE_1.9_URN");
// validamos firma
boolean firmaValida = xadesSign.verifySignature(signature).isCorrect();


private Properties getPolicyParams() {
          Properties policyParams = new Properties();
          policyParams.put(SignatureProperties.XADES_CLAIMED_ROLE_PROP, "emisor");
          return policyParams;
}
```

```
private Properties getDataFormatParams() {
          Properties dataFormatProp = new Properties();
          dataFormatProp.put(SignatureProperties.XADES_DATA_FORMAT_DESCRIPTION_PRO
P, "Texto plano");
          dataFormatProp.put(SignatureProperties.XADES_DATA_FORMAT_ENCODING_PROP,
"utf-8");
          dataFormatProp.put(SignatureProperties.XADES_DATA_FORMAT_MIME_PROP,
"text/plain");
          return dataFormatProp;
}
```

Si se desea obtener firmas con sello de tiempo puede usarse el mismo código del ejemplo anterior haciendo las llamadas al método ***sign*** de ***XadesSigner*** indicando "true" en el parámetro habilitado a tal efecto.

Se hará uso de un código como el siguiente si se desea generar una co-firma:

```
XadesSigner xadesSign = new XadesSigner();

byte[ ] eSignature = "firma a co-firmar";
byte[ ] data = "documento xml original firmado";

// cofirma documento xml
byte[ ] coSignature = xadesSign.coSign(eSignature, data,
SignatureConstants.SIGN_ALGORITHM_SHA256WITHRSA, "clave privada del certificado a
utilizar", null, false, SignatureFormatDetector.FORMAT_XADES_BES, null);
```

Se hará uso de un código como el siguiente si se desea generar una contra-firma:

```
XadesSigner xadesSign = new XadesSigner();
byte[ ] eSignature = "firma a contra-firmar""

Properties dataFormatProp = new Properties();
dataFormatProp.put(SignatureProperties.XADES_DATA_FORMAT_DESCRIPTION_PROP, "Texto
plano");
dataFormatProp.put(SignatureProperties.XADES_DATA_FORMAT_ENCODING_PROP, "utf-8");
dataFormatProp.put(SignatureProperties.XADES_DATA_FORMAT_MIME_PROP, "text/plain");

// contrafirma documento xml
byte[ ] counterSign = xadesSign.counterSign(eSignature,
SignatureConstants.SIGN_ALGORITHM_SHA256WITHRSA, "clave privada del certificado a
utilizar", dataFormatProp, false, SignatureFormatDetector.FORMAT_XADES_BES, null);
```

Se hará uso de un código como el siguiente si se desea actualizar una firma:

```
/*
* Actualizar todos los firmantes de una firma
byte[ ] xades = UtilsFileSystemCommons.readFile("signatures/XML/XAdES-T.xml",
true);
byte[ ] xadesTUpgrade = xadesSigner.upgrade(xades, null);

/*
* Actualizar un firmante que no existe de una firma XAdES
*/
Properties dataFormatProp = new Properties();
```

```
dataFormatProp.put(SignatureProperties.XADES_DATA_FORMAT_DESCRIPTION_PROP, "Texto
plano");
dataFormatProp.put(SignatureProperties.XADES_DATA_FORMAT_ENCODING_PROP, "utf-8");
dataFormatProp.put(SignatureProperties.XADES_DATA_FORMAT_MIME_PROP, "text/plain");

byte[ ] xades = xadesSigner.sign("documento xml a firmar",
SignatureConstants.SIGN_ALGORITHM_SHA384WITHRSA,
SignatureConstants.SIGN_FORMAT_XADES_ENVELOPING, "clave privada de certificado a
utilizar", dataFormatProp, false, SignatureFormatDetector.FORMAT_XADES_BES, null);
byte[ ] certificateBytes = UtilsFileSystemCommons.readFile("serversigner.cer",
true);
X509Certificate certificateServerSigner2 =
UtilsCertificate.generateCertificate(certificateBytes);
List<X509Certificate> listSigners = new ArrayList<X509Certificate>();
listSigners.add(certificateServerSigner2);
byte[ ] upgradedSignature = xadesSigner.upgrade(xades, listSigners);
```

A.6.3.2 Formato XAdES Baseline

A continuación de describen diferentes ejemplos asociados a la generación, actualización y validación de firmas XAdES Baseline realizados haciendo uso de la fachada Signer:

```
byte[ ] dataToSign = getDataToSign();

es.gob.afirma.signature.xades.XAdESBaselineSigner signer = new
es.gob.afirma.signature.xades.XAdESBaselineSigner();
byte[ ] xadesBLevelSignature = null;
byte[ ] xadesBLevelCoSignature = null;
byte[ ] xadesBLevelCounterSignature = null;
byte[ ] xadesTLevelSignature = null;
byte[ ] xadesTLevelCoSignature = null;
byte[ ] xadesTLevelCounterSignature = null;
es.gob.afirma.signature.validation.ValidationResult vr = null;
java.security.KeyStore.PrivateKeyEntry privateKey = getCertificatePrivateKey();
java.util.Properties extraParams = new java.util.Properties();
extraParams.put(es.gob.afirma.signature.SignatureProperties.XADES_DATA_FORMAT_DESC
RIPTION_PROP, "Description Test");
extraParams.put(es.gob.afirma.signature.SignatureProperties.XADES_DATA_FORMAT_ENCO
DING_PROP, "UTF-8");
extraParams.put(es.gob.afirma.signature.SignatureProperties.XADES_DATA_FORMAT_MIME
_PROP, "application-xml");

/*
 * Generación y Validación de firma XAdES B-Level detached sin política de firma y
algoritmo SHA-1
 */
try {
        xadesBLevelSignature = signer.sign(dataToSign,
es.gob.afirma.signature.SignatureConstants.SIGN_ALGORITHM_SHA1WITHRSA,
es.gob.afirma.signature.SignatureConstants.SIGN_FORMAT_XADES_DETACHED, privateKey,
extraParams, false,
es.gob.afirma.signature.ISignatureFormatDetector.FORMAT_XADES_B_LEVEL, null);
        vr = signer.verifySignature(xadesBLevelSignature);
} catch (Exception e) {
        e.printStackTrace();
}
```

```
/*
 *  Generación y Validación de co-firma XAdES B-Level detached sin política de
firma y algoritmo SHA-256
 */
try {
          xadesBLevelCoSignature = signer.coSign(xadesBLevelSignature, dataToSign,
es.gob.afirma.signature.SignatureConstants.SIGN_ALGORITHM_SHA256WITHRSA,
privateKey, extraParams, false,
es.gob.afirma.signature.ISignatureFormatDetector.FORMAT_XADES_B_LEVEL, null);
          vr = signer.verifySignature(xadesBLevelCoSignature);
} catch (Exception e) {
          e.printStackTrace();
}

/*
 *  Generación y Validación de contra-firma XAdES B-Level detached sin política de
firma y algoritmo SHA-384
 */
try {
          xadesBLevelCounterSignature = signer.counterSign(xadesBLevelSignature,
es.gob.afirma.signature.SignatureConstants.SIGN_ALGORITHM_SHA384WITHRSA,
privateKey, extraParams, false,
es.gob.afirma.signature.ISignatureFormatDetector.FORMAT_XADES_B_LEVEL, null);
          vr = signer.verifySignature(xadesBLevelCounterSignature);
} catch (Exception e) {
          e.printStackTrace();
}

/*
 * Generación y Validación de firma XAdES B-Level enveloped sin política de firma
y algoritmo SHA-1
 */
try {
          xadesBLevelSignature = signer.sign(dataToSign,
es.gob.afirma.signature.SignatureConstants.SIGN_ALGORITHM_SHA1WITHRSA,
es.gob.afirma.signature.SignatureConstants.SIGN_FORMAT_XADES_ENVELOPED,
privateKey, extraParams, false,
es.gob.afirma.signature.ISignatureFormatDetector.FORMAT_XADES_B_LEVEL, null);
          vr = signer.verifySignature(xadesBLevelSignature);
} catch (Exception e) {
          e.printStackTrace();
}

/*
 *  Generación y Validación de co-firma XAdES B-Level enveloped sin política de
firma y algoritmo SHA-256
 */
try {
          xadesBLevelCoSignature = signer.coSign(xadesBLevelSignature, dataToSign,
es.gob.afirma.signature.SignatureConstants.SIGN_ALGORITHM_SHA256WITHRSA,
privateKey, extraParams, false,
es.gob.afirma.signature.ISignatureFormatDetector.FORMAT_XADES_B_LEVEL, null);
          vr = signer.verifySignature(xadesBLevelCoSignature);
} catch (Exception e) {
          e.printStackTrace();
}

/*
```

```
 *  Generación y Validación de contra-firma XAdES B-Level enveloped sin política
de firma y algoritmo SHA-512
 */
try {
          xadesBLevelCounterSignature = signer.counterSign(xadesBLevelSignature,
es.gob.afirma.signature.SignatureConstants.SIGN_ALGORITHM_SHA512WITHRSA,
privateKey, extraParams, false,
es.gob.afirma.signature.ISignatureFormatDetector.FORMAT_XADES_B_LEVEL, null);
          vr = signer.verifySignature(xadesBLevelCounterSignature);
} catch (Exception e) {
          e.printStackTrace();
}

/*
 * Generación y Validación de firma XAdES B-Level enveloping sin política de firma
y algoritmo SHA-1
 */
try {
          xadesBLevelSignature = signer.sign(dataToSign,
es.gob.afirma.signature.SignatureConstants.SIGN_ALGORITHM_SHA1WITHRSA,
es.gob.afirma.signature.SignatureConstants.SIGN_FORMAT_XADES_ENVELOPING,
privateKey, extraParams, false,
es.gob.afirma.signature.ISignatureFormatDetector.FORMAT_XADES_B_LEVEL, null);
          vr = signer.verifySignature(xadesBLevelSignature);
} catch (Exception e) {
          e.printStackTrace();
}

/*
 *  Generación y Validación de co-firma XAdES B-Level enveloping sin política de
firma y algoritmo SHA-256
 */
try {
          xadesBLevelCoSignature = signer.coSign(xadesBLevelSignature, dataToSign,
es.gob.afirma.signature.SignatureConstants.SIGN_ALGORITHM_SHA256WITHRSA,
privateKey, extraParams, false,
es.gob.afirma.signature.ISignatureFormatDetector.FORMAT_XADES_B_LEVEL, null);
          vr = signer.verifySignature(xadesBLevelCoSignature);
} catch (Exception e) {
          e.printStackTrace();
}

/*
 *  Generación y Validación de contra-firma XAdES B-Level enveloping sin política
de firma y algoritmo SHA-512
 */
try {
          xadesBLevelCounterSignature = signer.counterSign(xadesBLevelSignature,
es.gob.afirma.signature.SignatureConstants.SIGN_ALGORITHM_SHA512WITHRSA,
privateKey, extraParams, false,
es.gob.afirma.signature.ISignatureFormatDetector.FORMAT_XADES_B_LEVEL, null);
          vr = signer.verifySignature(xadesBLevelCounterSignature);
} catch (Exception e) {
          e.printStackTrace();
}

/*
 * Generación y Validación de firma XAdES B-Level detached con política de firma y
algoritmo SHA-1
```

```
 */
try {
          xadesBLevelSignature = signer.sign(dataToSign,
es.gob.afirma.signature.SignatureConstants.SIGN_ALGORITHM_SHA1WITHRSA,
es.gob.afirma.signature.SignatureConstants.SIGN_FORMAT_XADES_DETACHED, privateKey,
extraParams, false,
es.gob.afirma.signature.ISignatureFormatDetector.FORMAT_XADES_B_LEVEL,
"XML_AGE_1.9_URL");
          vr = signer.verifySignature(xadesBLevelSignature);
} catch (Exception e) {
          e.printStackTrace();
}

/*
 *  Generación y Validación de co-firma XAdES B-Level detached con política de
firma y algoritmo SHA-256
 */
try {
          xadesBLevelCoSignature = signer.coSign(xadesBLevelSignature, dataToSign,
es.gob.afirma.signature.SignatureConstants.SIGN_ALGORITHM_SHA256WITHRSA,
privateKey, extraParams, false,
es.gob.afirma.signature.ISignatureFormatDetector.FORMAT_XADES_B_LEVEL,
"XML_AGE_1.9_URL");
          vr = signer.verifySignature(xadesBLevelCoSignature);
} catch (Exception e) {
          e.printStackTrace();
}

/*
 *  Generación y Validación de contra-firma XAdES B-Level detached con política de
firma y algoritmo SHA-512
 */
try {
          xadesBLevelCounterSignature = signer.counterSign(xadesBLevelSignature,
es.gob.afirma.signature.SignatureConstants.SIGN_ALGORITHM_SHA512WITHRSA,
privateKey, extraParams, false,
es.gob.afirma.signature.ISignatureFormatDetector.FORMAT_XADES_B_LEVEL,
"XML_AGE_1.9_URL");
          vr = signer.verifySignature(xadesBLevelCounterSignature);
} catch (Exception e) {
          e.printStackTrace();
}

/*
 * Generación y Validación de firma XAdES B-Level enveloped con política de firma
y algoritmo SHA-1
 */
try {
          xadesBLevelSignature = signer.sign(dataToSign,
es.gob.afirma.signature.SignatureConstants.SIGN_ALGORITHM_SHA1WITHRSA,
es.gob.afirma.signature.SignatureConstants.SIGN_FORMAT_XADES_ENVELOPED,
privateKey, extraParams, false,
es.gob.afirma.signature.ISignatureFormatDetector.FORMAT_XADES_B_LEVEL,
"XML_AGE_1.9_URL");
          vr = signer.verifySignature(xadesBLevelSignature);
} catch (Exception e) {
          e.printStackTrace();
}
```

```
/*
 *  Generación y Validación de co-firma XAdES B-Level enveloped con política de
firma y algoritmo SHA-256
 */
try {
          xadesBLevelCoSignature = signer.coSign(xadesBLevelSignature, dataToSign,
es.gob.afirma.signature.SignatureConstants.SIGN_ALGORITHM_SHA256WITHRSA,
privateKey, extraParams, false,
es.gob.afirma.signature.ISignatureFormatDetector.FORMAT_XADES_B_LEVEL,
"XML_AGE_1.9_URL");
          vr = signer.verifySignature(xadesBLevelCoSignature);
} catch (Exception e) {
          e.printStackTrace();
}

/*
 *  Generación y Validación de contra-firma XAdES B-Level enveloped con política
de firma y algoritmo SHA-512
 */
try {
          xadesBLevelCounterSignature = signer.counterSign(xadesBLevelSignature,
es.gob.afirma.signature.SignatureConstants.SIGN_ALGORITHM_SHA512WITHRSA,
privateKey, extraParams, false,
es.gob.afirma.signature.ISignatureFormatDetector.FORMAT_XADES_B_LEVEL,
"XML_AGE_1.9_URL");
          vr = signer.verifySignature(xadesBLevelCounterSignature);
} catch (Exception e) {
          e.printStackTrace();
}

/*
 * Generación y Validación de firma XAdES T-Level detached sin política de firma y
algoritmo SHA-1
 */
try {
          xadesTLevelSignature = signer.sign(dataToSign,
es.gob.afirma.signature.SignatureConstants.SIGN_ALGORITHM_SHA1WITHRSA,
es.gob.afirma.signature.SignatureConstants.SIGN_FORMAT_XADES_DETACHED, privateKey,
extraParams, true,
es.gob.afirma.signature.ISignatureFormatDetector.FORMAT_XADES_B_LEVEL, null);
          vr = signer.verifySignature(xadesTLevelSignature);
} catch (Exception e) {
          e.printStackTrace();
}

/*
 *  Generación y Validación de co-firma XAdES T-Level detached sin política de
firma y algoritmo SHA-256
 */
try {
          xadesTLevelCoSignature = signer.coSign(xadesTLevelSignature, dataToSign,
es.gob.afirma.signature.SignatureConstants.SIGN_ALGORITHM_SHA256WITHRSA,
privateKey, extraParams, true,
es.gob.afirma.signature.ISignatureFormatDetector.FORMAT_XADES_B_LEVEL, null);
          vr = signer.verifySignature(xadesTLevelCoSignature);
} catch (Exception e) {
          e.printStackTrace();
}
```

```
/*
 *  Generación y Validación de contra-firma XAdES T-Level detached sin política de
firma y algoritmo SHA-384
 */
try {
          xadesTLevelCounterSignature = signer.counterSign(xadesTLevelSignature,
es.gob.afirma.signature.SignatureConstants.SIGN_ALGORITHM_SHA384WITHRSA,
privateKey, extraParams, true,
es.gob.afirma.signature.ISignatureFormatDetector.FORMAT_XADES_B_LEVEL, null);
          vr = signer.verifySignature(xadesTLevelCounterSignature);
} catch (Exception e) {
          e.printStackTrace();
}

/*
 * Generación y Validación de firma XAdES T-Level enveloped sin política de firma
y algoritmo SHA-1
 */
try {
          xadesTLevelSignature = signer.sign(dataToSign,
es.gob.afirma.signature.SignatureConstants.SIGN_ALGORITHM_SHA1WITHRSA,
es.gob.afirma.signature.SignatureConstants.SIGN_FORMAT_XADES_ENVELOPED,
privateKey, extraParams, true,
es.gob.afirma.signature.ISignatureFormatDetector.FORMAT_XADES_B_LEVEL, null);
          vr = signer.verifySignature(xadesTLevelSignature);
} catch (Exception e) {
          e.printStackTrace();
}

/*
 *  Generación y Validación de co-firma XAdES T-Level enveloped sin política de
firma y algoritmo SHA-256
 */
try {
          xadesTLevelCoSignature = signer.coSign(xadesTLevelSignature, dataToSign,
es.gob.afirma.signature.SignatureConstants.SIGN_ALGORITHM_SHA256WITHRSA,
privateKey, extraParams, true,
es.gob.afirma.signature.ISignatureFormatDetector.FORMAT_XADES_B_LEVEL, null);
          vr = signer.verifySignature(xadesTLevelCoSignature);
} catch (Exception e) {
          e.printStackTrace();
}

/*
 *  Generación y Validación de contra-firma XAdES T-Level enveloped sin política
de firma y algoritmo SHA-512
 */
try {
          xadesTLevelCounterSignature = signer.counterSign(xadesTLevelSignature,
es.gob.afirma.signature.SignatureConstants.SIGN_ALGORITHM_SHA512WITHRSA,
privateKey, extraParams, true,
es.gob.afirma.signature.ISignatureFormatDetector.FORMAT_XADES_B_LEVEL, null);
          vr = signer.verifySignature(xadesTLevelCounterSignature);
} catch (Exception e) {
          e.printStackTrace();
}

/*
```

```
 * Generación y Validación de firma XAdES T-Level enveloping sin política de firma
y algoritmo SHA-1
 */
try {
          xadesTLevelSignature = signer.sign(dataToSign,
es.gob.afirma.signature.SignatureConstants.SIGN_ALGORITHM_SHA1WITHRSA,
es.gob.afirma.signature.SignatureConstants.SIGN_FORMAT_XADES_ENVELOPING,
privateKey, extraParams, true,
es.gob.afirma.signature.ISignatureFormatDetector.FORMAT_XADES_B_LEVEL, null);
          vr = signer.verifySignature(xadesTLevelSignature);
} catch (Exception e) {
          e.printStackTrace();
}

/*
 *  Generación y Validación de co-firma XAdES T-Level enveloping sin política de
firma y algoritmo SHA-256
 */
try {
          xadesTLevelCoSignature = signer.coSign(xadesTLevelSignature, dataToSign,
es.gob.afirma.signature.SignatureConstants.SIGN_ALGORITHM_SHA256WITHRSA,
privateKey, extraParams, true,
es.gob.afirma.signature.ISignatureFormatDetector.FORMAT_XADES_B_LEVEL, null);
          vr = signer.verifySignature(xadesTLevelCoSignature);
} catch (Exception e) {
          e.printStackTrace();
}

/*
 *  Generación y Validación de contra-firma XAdES T-Level enveloping sin política
de firma y algoritmo SHA-512
 */
try {
          xadesTLevelCounterSignature = signer.counterSign(xadesTLevelSignature,
es.gob.afirma.signature.SignatureConstants.SIGN_ALGORITHM_SHA512WITHRSA,
privateKey, extraParams, true,
es.gob.afirma.signature.ISignatureFormatDetector.FORMAT_XADES_B_LEVEL, null);
          vr = signer.verifySignature(xadesTLevelCounterSignature);
} catch (Exception e) {
          e.printStackTrace();
}

/*
 * Generación de firma XAdES B-Level detached con política de firma y algoritmo
SHA-1 y actualización a XAdES T-Level
 */
try {
          xadesBLevelSignature = signer.sign(dataToSign,
es.gob.afirma.signature.SignatureConstants.SIGN_ALGORITHM_SHA1WITHRSA,
es.gob.afirma.signature.SignatureConstants.SIGN_FORMAT_XADES_DETACHED, privateKey,
extraParams, false,
es.gob.afirma.signature.ISignatureFormatDetector.FORMAT_XADES_B_LEVEL,
"XML_AGE_1.9_URL");
          xadesTLevelSignature = signer.upgrade(xadesBLevelSignature, null);
          vr = signer.verifySignature(xadesTLevelSignature);
} catch (Exception e) {
          e.printStackTrace();
}
```

```
/*
 *  Generación de co-firma XAdES B-Level detached con política de firma y
algoritmo SHA-256 y actualización a XAdES T-Level
 */
try {
          xadesBLevelCoSignature = signer.coSign(xadesBLevelSignature, dataToSign,
es.gob.afirma.signature.SignatureConstants.SIGN_ALGORITHM_SHA256WITHRSA,
privateKey, extraParams, false,
es.gob.afirma.signature.ISignatureFormatDetector.FORMAT_XADES_B_LEVEL,
"XML_AGE_1.9_URL");
          xadesTLevelCoSignature = signer.upgrade(xadesBLevelCoSignature, null);
          vr = signer.verifySignature(xadesTLevelCoSignature);
} catch (Exception e) {
          e.printStackTrace();
}

/*
 *  Generación de contra-firma XAdES B-Level detached con política de firma y
algoritmo SHA-512 y actualización a XAdES T-Level
 */
try {
          xadesBLevelCounterSignature = signer.counterSign(xadesBLevelSignature,
es.gob.afirma.signature.SignatureConstants.SIGN_ALGORITHM_SHA512WITHRSA,
privateKey, extraParams, false,
es.gob.afirma.signature.ISignatureFormatDetector.FORMAT_XADES_B_LEVEL,
"XML_AGE_1.9_URL");
          xadesTLevelCounterSignature =
signer.upgrade(xadesBLevelCounterSignature, null);
          vr = signer.verifySignature(xadesTLevelCounterSignature);
} catch (Exception e) {
          e.printStackTrace();
}

/*
 * Generación de firma XAdES B-Level enveloped con política de firma y algoritmo
SHA-1 y actualización a XAdES T-Level
 */
try {
          xadesBLevelSignature = signer.sign(dataToSign,
es.gob.afirma.signature.SignatureConstants.SIGN_ALGORITHM_SHA1WITHRSA,
es.gob.afirma.signature.SignatureConstants.SIGN_FORMAT_XADES_ENVELOPED,
privateKey, extraParams, false,
es.gob.afirma.signature.ISignatureFormatDetector.FORMAT_XADES_B_LEVEL,
"XML_AGE_1.9_URL");
          xadesTLevelSignature = signer.upgrade(xadesBLevelSignature, null);
          vr = signer.verifySignature(xadesTLevelSignature);
} catch (Exception e) {
          e.printStackTrace();
}

/*
 *  Generación de co-firma XAdES B-Level enveloped con política de firma y
algoritmo SHA-256 y actualización a XAdES T-Level
 */
try {
          xadesBLevelCoSignature = signer.coSign(xadesBLevelSignature, dataToSign,
es.gob.afirma.signature.SignatureConstants.SIGN_ALGORITHM_SHA256WITHRSA,
privateKey, extraParams, false,
```

```
es.gob.afirma.signature.ISignatureFormatDetector.FORMAT_XADES_B_LEVEL,
"XML_AGE_1.9_URL");
          xadesTLevelCoSignature = signer.upgrade(xadesBLevelCoSignature, null);
          vr = signer.verifySignature(xadesTLevelCoSignature);
} catch (Exception e) {
          e.printStackTrace();
}

/*
 *  Generación de contra-firma XAdES B-Level enveloped con política de firma y
algoritmo SHA-512 y actualización a XAdES T-Level
 */
try {
          xadesBLevelCounterSignature = signer.counterSign(xadesBLevelSignature,
es.gob.afirma.signature.SignatureConstants.SIGN_ALGORITHM_SHA512WITHRSA,
privateKey, extraParams, false,
es.gob.afirma.signature.ISignatureFormatDetector.FORMAT_XADES_B_LEVEL,
"XML_AGE_1.9_URL");
          xadesTLevelCounterSignature =
signer.upgrade(xadesBLevelCounterSignature, null);
          vr = signer.verifySignature(xadesTLevelCounterSignature);
} catch (Exception e) {
          e.printStackTrace();
}

// Creamos listado de propiedades adicionales que incluirá el objeto manifest con
todas las referencias externas.
es.gob.afirma.signature.xades.ReferenceData rd = new
es.gob.afirma.signature.xades.ReferenceData("http://www.w3.org/2000/09/xmldsig#sha
1", "zyjp8GJOX69990Kkqw8ioPXGExk=");
String xPath = "self::text()[ancestor-or-self::node()=/Class/e[1]]";
es.gob.afirma.signature.xades.ReferenceData.TransformData transform = new
es.gob.afirma.signature.xades.ReferenceData.TransformData("http://www.w3.org/2000/
09/xmldsig#base64", null);
es.gob.afirma.signature.xades.ReferenceData.TransformData transform2 = new
es.gob.afirma.signature.xades.ReferenceData.TransformData("http://www.w3.org/TR/19
99/REC-xpath-19991116", java.util.Collections.singletonList(xPath));
java.util.List<es.gob.afirma.signature.xades.ReferenceData.TransformData>
transformList = new
java.util.ArrayList<es.gob.afirma.signature.xades.ReferenceData.TransformData>(2);
transformList.add(transform);
transformList.add(transform2);
rd.setTransforms(transformList);
rd.setId("idAttribute");
rd.setType("typeAttribute");
rd.setUri("uriAttribute");
java.util.List<es.gob.afirma.signature.xades.ReferenceData> rdlist =
java.util.Collections.singletonList(rd);
extraParams.put(es.gob.afirma.signature.SignatureConstants.MF_REFERENCES_PROPERTYN
AME, rdlist);

/*
 * Generación y Validación de firma XAdES B-Level externally detached sin política
de firma y algoritmo SHA-1
 */
try {
          xadesBLevelSignature = signer.sign(dataToSign,
es.gob.afirma.signature.SignatureConstants.SIGN_ALGORITHM_SHA1WITHRSA,
es.gob.afirma.signature.SignatureConstants.SIGN_FORMAT_XADES_EXTERNALLY_DETACHED,
```

```
privateKey, extraParams, false,
es.gob.afirma.signature.ISignatureFormatDetector.FORMAT_XADES_B_LEVEL, null);
          vr = signer.verifySignature(xadesBLevelSignature);
} catch (Exception e) {
          e.printStackTrace();
}

/*
 *  Generación y Validación de co-firma XAdES B-Level externally detached sin
política de firma y algoritmo SHA-256
 */
try {
          xadesBLevelCoSignature = signer.coSign(xadesBLevelSignature, dataToSign,
es.gob.afirma.signature.SignatureConstants.SIGN_ALGORITHM_SHA256WITHRSA,
privateKey, extraParams, false,
es.gob.afirma.signature.ISignatureFormatDetector.FORMAT_XADES_B_LEVEL, null);
          vr = signer.verifySignature(xadesBLevelCoSignature);
} catch (Exception e) {
          e.printStackTrace();
}

/*
 *  Generación y Validación de contra-firma XAdES B-Level externally detached sin
política de firma y algoritmo SHA-512
 */
try {
          xadesBLevelCounterSignature = signer.counterSign(xadesBLevelSignature,
es.gob.afirma.signature.SignatureConstants.SIGN_ALGORITHM_SHA512WITHRSA,
privateKey, extraParams, false,
es.gob.afirma.signature.ISignatureFormatDetector.FORMAT_XADES_B_LEVEL, null);
          vr = signer.verifySignature(xadesBLevelCounterSignature);
} catch (Exception e) {
          e.printStackTrace();
}

/*
 * Generación y Validación de firma XAdES T-Level externally detached sin política
de firma y algoritmo SHA-1
 */
try {
          xadesTLevelSignature = signer.sign(dataToSign,
es.gob.afirma.signature.SignatureConstants.SIGN_ALGORITHM_SHA1WITHRSA,
es.gob.afirma.signature.SignatureConstants.SIGN_FORMAT_XADES_EXTERNALLY_DETACHED,
privateKey, extraParams, true,
es.gob.afirma.signature.ISignatureFormatDetector.FORMAT_XADES_B_LEVEL, null);
          vr = signer.verifySignature(xadesTLevelSignature);
} catch (Exception e) {
          e.printStackTrace();
}

/*
 *  Generación y Validación de co-firma XAdES T-Level externally detached sin
política de firma y algoritmo SHA-256
 */
try {
          xadesTLevelCoSignature = signer.coSign(xadesTLevelSignature, dataToSign,
es.gob.afirma.signature.SignatureConstants.SIGN_ALGORITHM_SHA256WITHRSA,
privateKey, extraParams, true,
es.gob.afirma.signature.ISignatureFormatDetector.FORMAT_XADES_B_LEVEL, null);
```

```
            vr = signer.verifySignature(xadesTLevelCoSignature);
} catch (Exception e) {
            e.printStackTrace();
}

/*
 *  Generación y Validación de contra-firma XAdES T-Level externally detached sin
política de firma y algoritmo SHA-512
 */
try {
            xadesTLevelCounterSignature = signer.counterSign(xadesTLevelSignature,
es.gob.afirma.signature.SignatureConstants.SIGN_ALGORITHM_SHA512WITHRSA,
privateKey, extraParams, true,
es.gob.afirma.signature.ISignatureFormatDetector.FORMAT_XADES_B_LEVEL, null);
            vr = signer.verifySignature(xadesTLevelCounterSignature);
} catch (Exception e) {
            e.printStackTrace();
}
```

A.6.3.3 Formato ASiC-S Baseline

A continuación de describen diferentes ejemplos asociados a la generación, actualización y validación de firmas ASiC-S Baseline, así como la obtención de información acerca de los datos firmados de firmas ASiC-S Baseline realizados haciendo uso de la fachada Signer:

```
es.gob.afirma.signature.asic.ASiCSBaselineSigner asicSBaselineSignatureManager =
new es.gob.afirma.signature.asic.ASiCSBaselineSigner();
byte[] asicSSignatureWithCAdESBLevelSignature =
getASiCSSignatureWithCAdESBLevelSignatureInside();
byte[] asicSSignatureWithXAdESBLevelSignature =
getASiCSSignatureWithXAdESBLevelSignatureInside();
byte[ ] upgradedASiCSignature = null;
es.gob.afirma.signature.validation.ValidationResult vr = null;

/*
 * Actualización de todos los firmantes que posee la firma CAdES B-Level contenida
en una firma ASiC-S, y validación posterior de la firma ASiC-S
 */
try{
            upgradedASiCSignature=
asicSBaselineSignatureManager.upgrade(asicSSignatureWithCAdESBLevelSignature,
null);
            vr =
asicSBaselineSignatureManager.verifySignature(upgradedASiCSignature);
}
catch(Exception e){
            e.printStackTrace();
}

/*
 * Actualización de todos los firmantes que posee la firma XAdES B-Level contenida
en una firma ASiC-S, y validación posterior de la firma ASiC-S
 */
try{
```

```
        upgradedASiCSignature=
asicSBaselineSignatureManager.upgrade(asicSSignatureWithXAdESBLevelSignature,
null);
        vr =
asicSBaselineSignatureManager.verifySignature(upgradedASiCSignature);
}
catch(Exception e){
        e.printStackTrace();
}
```

Se hará uso de un código como el siguiente si se desea obtener información sobre los datos firmados.

```
es.gob.afirma.signature.asic.ASiCSBaselineSigner asicSBaselineSignatureManager =
new es.gob.afirma.signature.asic.ASiCSBaselineSigner();
/*
* Obtención de los datos firmados de una firma ASiC-S Baseline.
*/
try{
        OriginalSignedData osd =
asicSBaselineSignatureManager.getSignedData("array de bytes con la firma de la que
se quiere obtener la información");
} catch(SigningException e) {
        e.printStackTrace();
}
```

Se hará uso de un código como el siguiente si se desea generar una firma ASiC-S Baseline conteniendo una firma XAdES Baseline.

```
ASiCSBaselineSigner signer = new ASiCBaselineSigner();

byte[ ] dataToSign = "documento a firmar";

//generación de firma ASiC-S Baseline con firma XAdES Baseline detached, sin
política de firma y sin sello de tiempo.

byte[ ] signature = signer.sign(dataToSign, SignatureConstants.
SIGN_ALGORITHM_SHA256WITHRSA, SignatureConstants.SIGN_FORMAT_XADES_DETACHED,
getCertificatePrivateKey(),getDataFormatParams(), false,
ISignatureFormatDetector.FORMAT_XADES_B_LEVEL, null);

// validamos firma
boolean firmaValida = signer.verifySignature(signature).isCorrect();
```

Se hará uso de un código como el siguiente si se desea generar una firma ASiC-S Baseline conteniendo una firma CAdES Baseline.

```
ASiCSBaselineSigner signer = new ASiCBaselineSigner();

byte[ ] dataToSign = "documento a firmar";

//generación de firma ASiC-S Baseline con firma CAdES Baseline sin política de
firma
asicsCadesBaseline = signer.sign(dataToSign,
SignatureConstants.SIGN_ALGORITHM_SHA512WITHRSA, SignatureConstants.
```

```
SIGN_MODE_EXPLICIT, getCertificatePrivateKey(), null, false,
ISignatureFormatDetector.FORMAT_CADES_B_LEVEL, null);

//validamos la firma
Boolean firmaValida = signer.verifySignature(asicsCadesBaseline).isCorrect();
```

A.7 Ejemplos de Uso de los Servicios para el tipo de integración 6

En este anexo se incluye algunos ejemplos de uso relacionado con la funcionalidad de cifrado y descifrado que ofrece Integr@

A.7.1 Configuración general.

Para este tipo de integración es necesario hacer uso de las librerías de Integr@ y librerías de terceros enumeradas en el apartado 7.7.

A continuación, se procederá a configurar el lenguaje de los mensajes de Integr@ editando el archivo de propiedades **Language.properties**:

```
LANGUAGE = es_ES
```

Con esta configuración la plataforma emitirá los mensajes en español.

A.7.2 Cifrado simétrico de datos.

Se hará uso de un código como el siguiente para realizar el cifrado de datos utilizando un algoritmo simétrico, para el ejemplo, el algoritmo seleccionado es **AES**.

```
String text = "Prueba de cifrado simétrico";
//Se instancia la clase CipherIntegra, indicando el algoritmo a utilizar y la
clave necesaria para el encriptado.
CipherIntegra ci = new CipherIntegra(AlgorithmCipherEnum.AES, key);

//se llama al método encryp pasándole como parámetro el mensaje a encriptar.
String cipherText = ci.encrypt(text);
```

A.7.3 Descifrado simétrico de datos.

Se hará uso de un código como el siguiente para realizar el descifrado de datos utilizando un algoritmo simétrico, para el ejemplo, el algoritmo seleccionado es **AES**.

```
String cipherText = " +ptk2U6CeAeVkQgLQV10CstcfbraIeZ5Y9Pzygy8eMk=";
//Se instancia la clase CipherIntegra, indicando el algoritmo a utilizar y la
clave correspondiente para el descifrado.
CipherIntegra ci = new CipherIntegra(AlgorithmCipherEnum.AES, key);

//se llama al método decryp pasándole como parámetro el mensaje a encriptar.
```

```
String originalText = ci.decrypt(cipherText);
```

A.7.4 Cifrado asimétrico de datos.

Se hará uso de un código como el siguiente para realizar el cifrado de datos utilizando un algoritmo asimétrico, para el ejemplo, el algoritmo seleccionado es **RSA-PKCS#1**.

```
String text = "Prueba de cifrado asimétrico";
//Se instancia la clase CipherIntegra, indicando el algoritmo a utilizar y la
clave pública necesaria para el encriptado.
CipherIntegra ci = new CipherIntegra(AlgorithmCipherEnum.RSA_PKCS1, publicKey);

//se llama al método encryp pasándole como parámetro el mensaje a encriptar.
String cipherText = ci.encrypt(text);
```

A.7.5 Descifrado asimétrico de datos.

Se hará uso de un código como el siguiente para realizar el descifrado de datos utilizando un algoritmo asimétrico, para el ejemplo, el algoritmo seleccionado es **RSA-PKCS#1**.

```
String cipherText = "Sj+XyxHSU8BN1N4+txRqspTBJId8Sbcw94mTmrFKB2aFBpsQ0Ut4…";

//Se instancia la clase CipherIntegra, indicando el algoritmo a utilizar y la
clave privada correspondiente para el descifrado.
CipherIntegra ci = new CipherIntegra(AlgorithmCipherEnum.AES, privateKey);

//se llama al método decryp pasándole como parámetro el mensaje a desencriptar.
String originalText = ci.decrypt(cipherText);
```

A.8 Ejemplos de Uso del API para el tipo de integración 7

En este anexo se incluyen algunos ejemplos de uso relacionados con invocación del API de generación de informes de firma. En definitiva, se trata de construir y completar los distintos tipos de datos que forman parte de los parámetros de entrada y por último invocar el método de generación del informe.

A.8.1 Obtención de parámetros de entrada

A.8.1.1 Datos de validación de la firma

```
//Depende de la configuración de la plantilla XSLT.
//Obtenemos los datos desde la parte más "interna" hacia el "exterior":
//CertificateInfo → IndividualSignature → ValidationResult

//Obtenemos la información del firmante para una firma individual
//Podrán añadirse 0…n pares clave/valor de información sobre firmantes.
LinkedHashMap<String, String> certInfo = new LinkedHashMap<String, String>();
certInfo.put("primerApellidoResponsable", "Rodríguez");
certInfo.put("segundoApellidoResponsable", "Gómez");
certInfo.put("nombreResponsable", "José Javier");
certInfo.put("NIFResponsable", "11111111T");

//Obtenemos el objeto que representa la información de la firma individual
//Podrán incluirse 0…n ocurrencias de esta información.
List<IndividualSignature> listSig = new ArrayList<IndividualSignature>();
DateTimeFormatter formatter = DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm");
String isResultMinor = "";
String isResultMajor = "urn:afirma:dss:1.0:profile:XSS:resultmajor:ValidSignature";
String isResultMessage = "La firma es válida";
LocalDateTime timestamp = LocalDateTime.parse("2020-09-15", formatter);
IndividualSignature indivSig = new
IndividualSignature(isResultMajor,isResultMinor,isResultMessage,timestamp,certInfo);
listSig.add(IndividualSignature);

//Obtenemos el objeto que representa la información del resultado de validación.
//Este dato es opcional, luego su valor podría ser nulo.
String resultMinor = "";
String resultMajor = "urn:afirma:dss:1.0:profile:XSS:resultmajor:ValidSignature";
String resultMessage = "La firma es válida";
ValidationData validData = new ValidationData(resultMajor, resultMinor, resultMessage, listSig);
//Formato de la la fecha de cada firma individual
validData.setTimeStampFormat("yyyy-MM-dd HH:mm");
```

A.8.1.2 Datos de inclusón del document original

```
//El documento podrá incluirse de dos maneras excluyentes: embebido o
//concatenado.
//En este ejemplo se presentan ambas maneras.

DocInclusionData docIncData = new DocInclusionData();

//Ejemplo documento embebido
//Depende de la configuración de la plantilla XSLT.
docIncData.setDocInclusionMode(0);
docIncData.setDocConcatRule(null);

//////////////////////////////////////////////////////
```

```
//Ejemplo documento concatenado
docIncData.setDocInclusionMode(1);
docIncData.setDocConcatRule("REP+DOC");
```

A.8.1.3 Códigos de barra

```
//Listado de códigos de barra a incluir en el informe.
//Depende de la configuración de la plantilla XSLT.
ArrayList<Barcode> listBarcodes = new ArrayList<Barcode>();
String type = "QRCode";
String data = "Mensaje codificado en el código de barras…";
Barcode barcode = new Barcode(type, data);
listBarcodes.add(barcode);
```

A.8.1.4 Adjuntos al informe generado

```
//Listado de adjuntos al informe
ArrayList<FileAttachment> attachments = new ArrayList<FileAttachment>();
String path = "[ruta_absoluta_del_fichero_adjunto]";
String name = "nombre_adjunto.ext";
String desc = "descripción del adjunto";
File fileanx = new File(path);
byte[] byteanx = Files.readAllBytes(fileanx.toPath());
fileatt = new FileAttachment(name, desc);
fileatt.setContent(byteanx);
attachements.add(fileatt);
```

A.8.1.5 Parámetros externos/adicionales

```
//Depende de la configuración de la plantilla XSLT.
LinkedHashMap<String, String> addParams = new LinkedHashMap<String, String>();
addParams.put("nombreVariable1", "valorVariable1");
addParams.put("nombreVariable2", "valorVariable2");
……….
……….
```

A.8.2 Invocación del API

```
try {
```

```
        IReportManager repman = new PdfReportManager();

        //Se invoca el método de generación del informe de firma con los parámetros
        //obtenidos anteriormente.
        byte[] pdf = repman.createReport(validData, docIncData, xsltbyte, docoribyte, listBarcodes,
                                                    attachments, addparams);

        //Tratamiento del byte[] resultante para obtener el informe de firma de la
        //manera deseada.

} catch (SignatureReportException e) {
            LOGGER.error("Ha ocurrido un error durante la generación del informe", e);
}
```

A.9 Ejemplos de Uso del API para el tipo de integración 8

En este anexo se incluyen algunos ejemplos de uso para la invocación del API de validación de certificados mediante TSL.

A.9.1 Obtención de la TSL

A.9.1.1 Obtención de la TSL desde un fichero en local.

```
//Ruta absoluta donde se encuentra el fichero XML con la TSL a utilizar
String pathTsl = "D:/ Pruebas_Validar_TSL/TSL/TSL-IT.xml";

//Se instancia la clase TSLValidation
ITslValidation tslValidation = new TslValidation();

//se obtiene el objeto que representa la TSL
ITSLObject tsl = tslValidation.getTSLObject(pathTsl);
```

A.9.1.2 Obtención de la TSL desde su URI de publicación.

```
//URI de publicación de la TSL de Italia.
String uriTSL = https://eidas.agid.gov.it/TL/TSL-IT.xml;
//Timeout de conexión en milisegundos
int connectionTimeout =10000;
//Timeout de lectura en milisegundos
Int readyTimeout=10000;


//Se instancia la clase TSLValidation
ITslValidation tslValidation = new TslValidation();

//se obtiene el objeto que representa la TSL
ITSLObject tsl = tslValidation.downloadTLSbyHTTP(uriTSL, connectionTimeout, readyTimeout);
```

A.9.2 Validación de certificado mediante TSL

```
//Parámetros para obtener la TSL

//URI de publicación de la TSL de Italia.
String uriTSL = https://eidas.agid.gov.it/TL/TSL-IT.xml;
//Timeout de conexión en milisegundos
int connectionTimeout =10000;
//Timeout de lectura en milisegundos
int readyTimeout=10000;

//Parámetros para la validación de certificado con la TSL.

//Fecha frente a la que se realiza el proceso de validación de certificado
String  validationDateString = “2019-10-01T14:41:05.811+01:00”;
Date validationDate = UtilsDate.transformDate(validationDateString, "yyyy-MM-
dd'T'HH:mm:ss.SSSXXX");
//Ruta absoluta donde se localiza el certificado a validar.
String x509CertLocation = “D:/ Pruebas_Validar_TSL/TSL/certIt.crt”;

//Parámetro que indica si se desea obtener información de mapeos del certificado
//(true) o no (false).
integra.tsl.getInfo=true

//Parámetro que indica si se desea comprobar el estado de revocación del
//certificado (true) o no (false).
integra.tsl.checkRevStatus=true
//Se instancia la clase TSLValidation
ITslValidation tslValidation = new TslValidation();

//se obtiene la TSL con alguno de los dos métodos explicados en los dos apartados
anteriores, por ejemplo el de descarga.
ITSLObject tsl = tslValidation.downloadTLSbyHTTP(uriTSL, connectionTimeout, readyTimeout);


//Se obtiene el certificado como un array de byte, de la siguiente forma
File x509certFile = new File(x509CertLocation);
if (x509certFile.exists()) {
	try {
		x509CertByteArray = IOUtils.toByteArray(new FileInputStream(x509certFile));
	} catch (IOException e) {
		System.err.println("No se ha obtenido el certificado a validar." );
	}
}


//se llama al método de validar pasándole los parámetros necesarios
DetectCertInTslInfoAndValidationResponse response =
tslValidation.validateCertificateTsl(x509CertByteArray, tslObject, validationDate, getInfo,
checkRevocationStatus);
```