

ARSS_Developer's_Guide_2.13.x

ARSS Developer's Guide

Contents

- [ARSS_Developer's_Guide_2.13.x](#)
- [Contents](#)
- [List of Revisions](#)
- [General](#)
 - [Purpose of the document](#)
 - [Requirements for reading](#)
 - [Definitions and acronyms](#)
- [Introduction to ARSS](#)
 - [Remote signatures](#)
 - [Architecture of the Remote Signature service](#)
 - [ARSS](#)
 - [Managing delegated users](#)
 - [Extended authentication methods](#)
- [Servlet WSDLSSL to avoid https redirect on WSDL](#)
- [Remote Digital Signatures in pkcs1](#)
 - [Digital Signature of a fingerprint in accordance with the pkcs1 standard](#)
- [Remote Digital Signatures in the CAdES \(p7m\) standard](#)
 - [Digital signatures](#)
 - [Multiple Signatures](#)
 - [Multiple signature methods](#)
 - [Automatic Signature](#)
 - [Joint Signature \(parallel signature\)](#)
 - [Digital Signature of a fingerprint](#)
- [Remote Digital Signatures in the PAdES \(pdf\) standard](#)
 - [Digital signatures](#)
 - [Multiple Signatures](#)
 - [Automatic Signature](#)
 - [Joint Signature \(parallel signature\)](#)
 - [Signature with a signature field that can be filled in](#)
 - [PDF signatures with DTS timestamp](#)
 - [PDF signatures \(Deprecated\)](#)
- [Remote Digital Signatures in the XAdES \(xml\) standard](#)
 - [Brief recap on XML signatures](#)
 - [Digital signatures](#)
 - [Multiple Signatures](#)
 - [Automatic Signature](#)
- [Remote Digital Signatures in the JAdES](#)
 - [Digital signatures \(SOAP\)](#)
 - [Multiple Signatures \(SOAP\)](#)
 - [Digital signatures \(REST\)](#)
 - [Multiple Signatures \(REST\)](#)
- [Timestamping](#)
 - [Brief recap on timestamping](#)
 - [Timestamp in TSR \(.tsr\) format](#)
 - [Timestamp in TSD \(.tsd\) format](#)
 - [Timestamp in M7M \(.m7m\) format](#)
- [Verifying digital signatures](#)
- [Additional functions](#)
 - [List of certificates associated with a user \(via user credentials\)](#)
 - [List of certificates associated with a user \(via administrator's credentials\)](#)
 - [List of methods to authenticate a user](#)
 - [List of active signature processes](#)
 - [Get ARSS version](#)
 - [Utility for checking connection to ARSS](#)
 - [Utility to authenticate users with an OTP](#)
 - [Utility for sending one of the OTP signature credentials via SMS or ARUBACALL \(caller ID\)](#)
 - [Utility to send one of the OTP signature credentials via SMS or ARUBACALL \(caller ID\) with Credential Proxy active](#)
 - [Utility to get a credential \(OTP PAPERTOKEN\)](#)
 - [File encryption](#)
 - [WS-SECURITY soap header signature](#)
 - [Utility for retrieving users within a domain](#)
 - [Change Password](#)
 - [Update signature](#)
- [Appendices](#)
- [Return Codes](#)

List of Revisions

Version	Changes	Revision date
2.13.2	add return code for specific errors: 0033 for any division by zero 0034 for page index of bound fixed automatic signature cache using of current timezone into pades graphical signature appearance	4-dic-2023
2.13.0	specify a limit for multiple signatures methods review of return codes pdfsignatureV2_multiple add return code for pin blocked status	28-set-2023
2.12.3	bug fix	12-giu-2023
2.12.1	review authMethods return codes	16-mar-2023
2.12.0	Support to wsdlssl servlet Support to JAdES signature	28-dic-2022
2.11.4	Add support to fieldName into appearance of pdfSignatureV2 Performances improvement for automatic signature	11-mag-2022
2.11.3	ext_authtype attribute of Identity object is deprecated add authMethods return codes	3-feb-2022
2.11.0	Add updateSignature method Add new error code 0031 for invalid page number on pdfsignature with appearance	14-ott-2021
2.10.0	Add DSD Usage Mode parameter to manage allowed domains Add signatureProfile to xmlsignature request parameters Reneable transport of type DIRECTORYNAME	11-May-2021
2.9.4	Add XAdES Expert Mode configuration to define how manage XAdES Transforms in xmlsignature method	05-Mar-2020
2.9.0	sendCredentialsAuth method added Remote Proxy Credential incorporated	27-Oct-2020
2.8.0	pdfsignatureDTS method added	14-Aug-2020
2.7.4	Missing change_password method added	7-Jul-2020
2.7.3	List of revisions introduced Incorrect Password return code and Incorrect user deleted, generic Incorrect Credentials code introduced User Temporarily Suspended return code updated for too many incorrect login attempts	
2.7.0	LT formats introduced	

General

Purpose of the document

This document is aimed at software developers and contains the information needed for learning how to use the Aruba PEC **ARSS™** Web Services in as short a time as possible. This document does not replace, but rather supplements, the documents in HTML format (javadoc) that come with the Server component.

Requirements for reading

This document has been written for software developers with good theoretical and practical experience of using Web Services based on the SOAP protocol. It is also assumed that the reader is familiar with the concepts of

- public key encryption

- hashing
- encryption and digital signatures
- X.509 certificates
- certification authorities
- cryptographic envelopes
- pkcs#11.

Definitions and acronyms

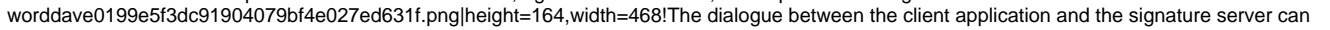
- **API** Application Programming Interface
- **CA** Certification Authority
- **CAPI** CryptoAPI
- **CNIPA** Centro Nazionale per l'Informatica nella Pubblica Amministrazione (National Centre for IT in Public Authorities) (now DigitPA)
- **CSR** Certificate Signing Request
- **DER** Distinguished Encoding Rules
- **DES** Data Encryption Standard
- **DN** Distinguished Name
- **DPCM** Decreto del Presidente del Consiglio dei Ministri (Prime Ministerial Decree)
- **DPR** Decreto del Presidente della Repubblica (Presidential Decree)
- **GUI** Graphical User Interface
- **HMAC** Hash-based MAC
- **HTML** HyperText Markup Language
- **HTTP** HyperText Transfer Protocol
- **I/O** Input/Output
- **LDAP** Light-weight Directory Access Protocol
- **MAC** Message Authentication Code
- **OCSP** Online Certificate Status Protocol
- **PDF** Portable Document Format
- **PEM** Privacy-Enhanced Mail
- **PKCS** Public Key Cryptographic Standard
- **PKI** Public Key Infrastructure
- **RFU** Reserved For Future Use
- **RSA** Rivest Shamir Adelman
- **SHA-1** Secure Hash Algorithm 1
- **SHA-256** Secure Hash Algorithm 256
- **SSL** Secure Sockets Layer
- **TSD** Time-Stamp Data
- **XML** Extensible Markup Language Introduction to ARSS

Introduction to ARSS

Remote signatures

Remote Signatures are an innovative kind of digital signature which, as well as guaranteeing the same level of security and the same legal validity as the traditional digital signature based on a smart card or USB token, offer various specific advantages over the traditional version:

- they do not require the installation of dedicated drivers and hardware, thus virtually reducing the associated hw/sw incompatibility issues and technical support, etc. to zero.
- they are essentially independent of the user's operating environment (Windows, Mac, Linux, ...)
- they allow you to generate digital signatures at any time and at any location.

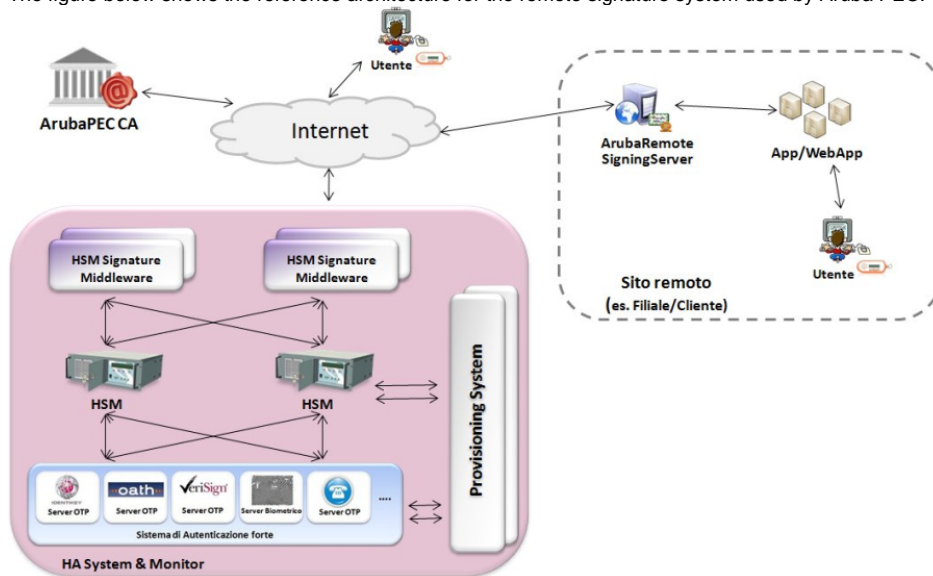
In practical terms, "Remote Signature" refers to a digital signature created with a private key not based on the user's personal device, such as a smartcard, but on a remote device (normally an HSM – Hardware Security Module). The data to be signed (or rather its fingerprint) is sent to the HSM via the Internet and the response comes back to the user, again via the Internet, as simplified in the figure below: !

The dialogue between the client application and the signature server can be summarised as follows:

- the user signs in on the server (more details later in this document);
- the client requests the digital signature by sending the document's digest (hash) to the server;
- the server calculates the signature and sends it back to the client, where it is saved.

This kind of solution can be used – with the same advantages – both to attach signatures to individual documents, interactively, and to bulk sign a series of documents (e.g. a whole folder)

Architecture of the Remote Signature service

The figure below shows the reference architecture for the remote signature system used by Aruba PEC.



architectural components:

The system is made up of the following

- **HSM CoSign** – HSMs within which the keys and digital certificates are generated and kept
- **HSM Signature Middleware** – software component that manages all requests from and to the HSMs
- **Strong Authentication System** – the Strong Authentication System is made up of different components that can be interfaced with the CoSign HSMs and which allow strong authentication for the Owner to release the different signature processes on the same HSM
- **Provisioning System** – software component that manages (orchestrates) the dialogue with the HSMs, with the Strong Authentication System and with the Certification Authority for the successful activation of the remote signature service
- **HA System & Monitor** – software component that runs through the whole remote signature system, made up of different modules that manage the way the System is monitored and implement all the functions needed to guarantee the highest level of availability of the service (fault-tolerance)
- **Aruba Remote Signing Server (ARSS)** – software component that makes it possible for the specific digital signature functionalities of the HSM to be managed remotely. ARSS therefore makes it possible to simply integrate the digital signature functionalities with "remote" applications and systems, i.e., which are supplied by production sites that are different from the one where the remote signature system is based.

ARSS

Aruba Remote Signing Server is the software component that makes it possible to simply integrate applications and systems with Aruba PEC's remote signature service. For applications (in use already or in the future) hosted in different IT systems (production sites) from the one where the Remote Signature system is based, ARSS will arrange to communicate, via a secure communication channel (HTTPS) with mutual authentication, with Aruba PEC's remote signature system, displaying the digital signature functionalities to the applications in question. These functionalities are made available via a simple Web Services interface (SOAP) and include (this list is not exhaustive):

- signatures for documents in CAdES-BES and CAdES-T format (in accordance with the provisions of Deliberation 45/2009 and subsequent Resolution 69/2010)
- signatures for documents in PAdES-Basic, PAdES-BES and PAdES-T format (in accordance with the provisions of Deliberation 45/2009 and subsequent Resolution 69/2010)
- signatures for documents in XAdES-BES and XAdES-T format (in accordance with the provisions of Deliberation 45/2009 and subsequent Resolution 69/2010)
- signatures for all the documents in a specified folder, in the formats mentioned above
- signatures for a hash (fingerprint), when the applications independently organize the creation of the digitally signed document via the APIs and/or services made available by the provider
- document timestamping

The table below gives an overview of the functionalities offered by ARSS:

Message/Method	Functionality
addpkcs7sign	Adds Joint Signature (Parallel Signature)
closesession	Closes a Mass Signature session
getVersion	Returns ARSS version
listCert	Returns the list of certificates associated with a remote user via the user's credentials
listCertAuth	Returns the list of certificates associated with a remote user via the administrator's credentials

authMethods	Returns the list of methods to authenticate a user
listprocess	Returns the list of active signature processes for each user
m7m	Method Removed , Generating Timestamps in m7m format
opensession	Opens a Mass Signature session
Ping	Checks the accessibility of the remote service
pdfsignatureV2	Adds a PAdES (-BES, -T) signature to a pdf file
pdfsignature	Deprecated Method , see pdfsignatureV2
pdfsignatureDTS	Adds a PAdES signature with a DTS timestamp
pkcs7signV2	Adds a CAdES (-BES, -T) signature to a generic file
pkcs7sign	Deprecated Method , see pkcs7signV2
pkcs7signhash	Adds a CAdES (-BES, -T) signature to a fingerprint
Tsd	Generates Timestamps in tsd format
Tsr	Generates Timestamps in tsr format
Verify	Deprecated Method , new version not available
xmlsignature	Adds an XAdES (-BES, -T) signature to an xml file
signhash	Adds a pkcs1 signature to a file's fingerprint
verifyOtp	Authentication via OTP credential
sendCredential	Sends OTP credential via ARUBACALL or SMS
sendCredentialAuth	Sends OTP credential via mobile via ARUBACALL or SMS via just the username
retrieveCredential	Retrieves additional information for extended authentication
pkcs7signhash_multiple	Adds multiple CAdES (-BES, -T) signatures to a fingerprint
pdfsignatureV2_multiple	Adds multiple PAdES (-BES, -T) signatures to a pdf file
xmlsignature_multiple	Adds multiple XAdES (-BES, -T) signatures to an xml file
pkcs7signV2_multiple	Adds multiple AdES (-BES, -T) signatures to a generic file
encryptedEnvelope	Encrypts a file
jwsSignature	Adds single JAdES
jwsSignature_multiple	Adds multiple JAdES

The table below summarises the standards and algorithms supported by ARSS.

Public key encryption algorithms	RSA
Hashing algorithms	SHA-1, SHA256
Digital signature algorithms	SHA1withRSA and SHA256withRSA with PKCS#1 1.5 padding
Cryptographic envelope format	CAdES-BES and CAdES-T (ETSI TS 101 733)
PDF envelope format	PAdES-Basic, PDES-BES, PAdES-T (ETSI TS 102 778)
XML envelope format	XAdES-BES and XAdES-T (ETSI TS 101 903)
Timestamp format (timestamp tokens)	Rfc 3161
Protocol for access to the timestamping service	Rfc 3161
Format of the envelopes with timestamps	Rfc 5544 and m7m

The ARSS component has successfully passed interoperability tests with all DigitPA accredited certification bodies relating to the production of signatures in CAdES, PAdES and XAdES format.

Managing delegated users

From ARSS version 1.9 methods have been introduced to make integration with the ASB (Aruba Security Box) infrastructure possible..One of the most important changes includes the methods to **delegate** automatic signatures, which have resulted in changes to the Auth category implemented by ARSS.The change extends the set of Auth category properties in order to submit the credentials of the user delegated by the Owner of the signature key, during the signing process.

Extended authentication methods

Version 1.9 saw the introduction of strong authentication methods, enabling ARSS to authenticate users with systems other than OTP.The auth subject has been changed to accept complex extended authentication parameters such as binary data as well.In particular, the CNS2 authentication method allows you to send the signature for documents as a pkcs7 blob, signed with a CNS certificate, without the need to specify the user password or an OTP.

Servlet WSDLSSL to avoid https redirect on WSDL

The "<host>/ArubaSignService/wsdls" servlet has been implemented that returns a wsdl with Https link, useful for clients that do not support redirection of w3c schemas https

The standard wsdl is always reachable at <host>/ArubaSignService/ArubaSignService?wsdl

Remote Digital Signatures in pkcs1

Digital Signature of a fingerprint in accordance with the pkcs1 standard

If you intend to sign the fingerprint of a document (or a generic sequence of bytes) based on the pkcs1 standard, you just have to use the signhash() method.This method is defined as follows:

```
public SignHashReturn signhash(SignHashRequest request)
```

This method is normally used for the following:

- Generating "detached" digital signatures, i.e. separate from the signed file

INPUT

- **request** - SignHashRequest containing the following parameters for the signature request
 - **Identity** = Auth object containing the data to authenticate the signatory
 - **TypeHSM** = String containing the type of HSM.Always applied with the 'COSIGN' string
 - **TypeOtpAuth** = String used to indicate the authentication domainsignature If a remote signature user is being used.If a remote signature user is being used with a customized domain, you need to specify this domain (e.g. frXXXX).If an automatic signature user is being used, you need to specify the string shown when installing ARSS.
 - **User** = String containing the signature user's username.
 - **UserPWD** = String containing the signature user's password.
 - **OtpPWD** = String containing the user's valid OTP code for the signature transaction.
 - **ext_authtype** = Enum that allows you to use the extended authentication method. (**Deprecated** see *authMethods* method instead)
 - **ext_auth_blobvalue** = Array of bytes in which to add the authentication credential (encrypted challenge, etc.) for the *extended authentication methods* that allow for the exchange of binary data
 - **ext_auth_value** = String in which to add the authentication credential (OTP or similar) for the *extended authentication methods* that allow for the exchange of data in plain text
 - **delegated_user** = String that contains the username for the delegated user if the automatic signature has been delegated.In this case, the password for the Owner of the Signature key does not need to be specified
 - **delegated_password** = String that contains the delegated user's password
 - **delegated_domain** = Delegated user's domain (used in a similar way to the TypeOtpAuth parameter)
 - **certID** = Unique certificate ID to be used for the signature. The list of IDs can be retrieved using the listCert function explained below. If a specific certificate is not needed, you can specify AS0 to balance the requests, although this method assumes that the certificate itself does not contribute to the calculation of the HASH (not valid for CAdES signatures).
 - **hash** = Array of bytes containing the fingerprint for the document to be signed.
 - **hashtype** = String containing the name of the hash algorithm used to calculate the fingerprint (currently only "SHA256" available).
 - **session_id** = String used to specify the session ID within multiple signature transactions.For a signature for an individual file or a folder, do not specify or set as NULL.
 - **requirecert** = Boolean value that indicates the requirement for the presence of the signatory's certificate in the response.

OUTPUT

SignHashReturn object containing the following parameters:

- **Status** = String containing the outcome of the signature process

- OK Operation completed successfully
- Different from OK In case of error. Further details are included within the SignHashReturn.Description.
- **Description** = String containing a high level description of the error encountered and/or any details relating to the codes returned within the SignReturnV2.return_code property.
- **Return_code** = Enum containing the error code
 - "0001" Generic error in the signature process
 - "0002" Incorrect parameters for the type of transport indicated
 - "0003" Error during the credential verification phase
 - "0004" PIN error (more details in the SignReturnV2.description)
 - "0005" Invalid type of transport
 - "0006" Unauthorized type of transport
 - "0007" Invalid PDF signature profile
 - "0008" Impossible to complete the timestamping operation (e.g. impossible to connect to the service, remaining stamps ended etc.)
 - "0009" Invalid delegate credentials
 - "0010" Invalid user status (e.g. user suspended)
- **signature** = Array of bytes containing the outcome of the signature process.
- **cert** = Array of bytes containing the recipient's certificate if requested see requirecert parameter.
- **certID** = ID for the signatory's certificate.

Remote Digital Signatures in the CAdES (p7m) standard

Digital signatures

The method to be used for generating a digital signature for an individual file is pkcs7signV2().The method is defined as:

Public SignReturnV2 **pkcs7signV2** (SignRequestV2 request, boolean detached, boolean returner)

INPUT

- **request** - SignRequestV2 object containing the following parameters for the signature request
- **Transport** = Enum that indicates the type of input (from here on transport type)
 - BINARYNET If the array of bytes containing the file to sign is being added to the input
 - FILENAME If the path of the file to be signed is being added to the input
 - DIRECTORYNAME If the path of the folder with the files to be signed is being added to the input
 - STREAM If the stream for a file in MTOM format with JAX-WS java technology (for large files) is being added to the input
- **Binaryinput** = Byte array containing the document to be signed. Only used when Transport = BINARYNET
- **SrcName** = String containing the path of the file or folder containing the files to be signed. Only used when Transport = FILENAME or DIRECTORYNAME
- **Stream** = DataHandler to upload large files in Stream format Only used when Transport = STREAM
N.B.: The solution is only available with JAX-WS compliant clients.
N.B.: The resource indicated must be accessible from the ARSS server
- **DstName** = String containing the destination path of the file or folder for the signed files. Only used when Transport = FILENAME or DIRECTORYNAME
N.B.: The path specified must be accessible from the ARSS server
- **requiredmark** = Boolean value used to guide the generation of a CAdES-T envelope: digital signature with contextual addition of the timestamp.
N.B.: If you intend to have a timestamp in CAdES-T format, the ARSS server must first be configured in advance with a valid timestamping account.
 - TRUE If you intend to generate a CAdES-T
 - FALSE If not
- **signingTime** = String used to set the signingtime attribute (OID 1.2.840.113549.1.9.5) to be inserted in the CAdES envelope.
N.B.: ARSS will insert the date on the system of the Server being used.
 - dd/MM/yyyy HH:mm:ss If you intend to specify the signature date explicitly
 - NULL If you intend to refer the attribute value to ARSS.
 - optionally the signing time can include the timezone (eg. dd/MM/yyyy HH:mm:ss zz, 05/10/2023 14:59:00 CEST)
- **Session_id** = String used to specify the session ID within multiple signature transactions.For a signature for an individual file or a folder, do not specify or set as NULL.
- **Identity** = Auth object containing the data to authenticate the signatory
- **TypeHSM** = String containing the type of HSM.Always applied with the 'COSIGN' string
- **TypeOtpAuth** = String used to indicate the authentication
 - domainsignature If a remote signature user is being used. If a remote signature user is being used with a customized domain, you need to specify this domain (e.g. frXXXX). If an automatic signature user is being used, you need to specify the string shown when installing ARSS.
- **User** = String containing the signature user's username.
- **UserPWD** = String containing the signature user's password.
- **OtpPWD** = String containing the user's valid OTP code for the signature transaction.
- **ext_auththtype** = Enum that allows you to use the extended authentication method. (**Deprecated** see *authMethods* instead)
- **ext_auth_blobvalue** = Array of bytes in which to add the authentication credential (encrypted challenge, etc.) for the *extended authentication methods* that allow for the exchange of binary data
- **ext_auth_value** = String in which to add the authentication credential (OTP or similar) for the *extended authentication methods* that allow for the exchange of data in plain text

- **delegated_user** = String that contains the username for the delegated user if the automatic signature has been delegated. In this case, the password for the Owner of the Signature key does not need to be specified
- **delegated_password** = String that contains the delegated user's password
- **delegated_domain** = Delegated user's domain (used in a similar way to the TypeOtpAuth parameter)
- **tsa_identity** = Optional TSAAuth object, containing the data for TSS Server authentication. If not specified, the server will use the default account, if configured.
- **user** = String containing the timestamp user's username
- **password** = String containing the timestamp user's password
- **Notifyemail** = String used to specify the email address to which notification of the completion of the signature operation is sent.
- **Notify_id** = String containing the ID that the user intends to link to the signature session for the purposes of notification of the completion of the operation. Only used when Transport = DIRECTORYNAME for a signature for an individual file, this property can be omitted or set as NULL.
- **CertID** = RFU. Set with AS0
- **profile** = RFU. Set to NULL
- **detached** - Boolean value used to generate a detached CAdES envelope, i.e. without the original document. true = request to generate a CAdES signature not containing the original document (detached) false = request to generate a CAdES signature containing the original document
- **returnder** - Boolean value used to generate a CAdES envelope in DER FORMAT true = request for generation in DER format false = request in BER format (this is the default value)
- **signatureLevel** - Signature level required. (Permitted Values: LT)

OUTPUT

SignReturnV2 object containing the following parameters:

- **Status** = String containing the outcome of the signature process
 - OK Operation completed successfully
 - Different from OK In case of error. Further details are included within the SignReturnV2.description and SignReturnV2.return_code properties.
- **Binaryoutput** = Byte array containing the signed file
- **Stream** = DataHandler to download large files in Stream format Only used when Transport = STREAM
N.B.: The solution is only available with JAX-WS compliant clients.
- **DstPath** = String containing the path specified within the SignRequestV2.DstName parameter during the request phase. If the file specified in SignRequestV2.DstName already exists, the SignReturnV2.DstPath property contains the path to the file actually generated by ARSS.
- **Return_code** = Enum containing the error code
 - "0001" Generic error in the signature process
 - "0002" Incorrect parameters for the type of transport indicated
 - "0003" Error during the credential verification phase
 - "0004" PIN error (more details in the SignReturnV2.description)
 - "0005" Invalid type of transport
 - "0006" Unauthorized type of transport
 - "0007" Invalid PDF signature profile
 - "0008" Impossible to complete the timestamping operation (e.g. impossible to connect to the service, remaining stamps ended etc.)
 - "0009" Invalid delegate credentials
 - "0010" Invalid user status (e.g. user suspended)
- **Description** = String containing a high level description of the error encountered and/or any details relating to the codes returned within the SignReturnV2.return_code property

Multiple Signatures

A Multiple Signature is a type of signature used to sign a group of files while only having to enter your authentication credentials once. From version 1.9 on (see chapter 10), you can also use multiple signature methods that automate and optimize the process described in this chapter. If you have specific requirements, however, it is a good idea to use separate multiple signature methods. To generate a digital signature for a group of files in p7m format, two different approaches are possible:

1. Sign a folder accessible from the ARSS Server in which all the files to be signed are located
2. Repeatedly sign the individual files that make up the whole input

From version 1.9 on you can also use multiple signature methods that automate and optimize the process described in method two.

Both methods allow you to sign files by entering just one OTP, but without reducing the level of security for the signature process. **Signing a folder** In this case, simply use the pkcs7signV2() method described above in the paragraph entitled *Digital Signatures* with the following specifications:

1. Set the SignRequestV2.Transport property to DIRECTORYNAME
2. For the SignRequestV2.SrcName property, use the path for the destination folder that will contain the signed files
3. Set the SignRequestV2.Notify_id and SignRequestV2.Notifyemail properties accordingly to receive a notification message of completion of the signing process for the folder

Repeated signature for individual files

In this case, you need to use a sequence of method calls structured as follows:


```

opensession(session_ID)
pkcs7signV2(file_#1, session_ID)
pkcs7signV2(file_#2, session_ID)
...
pkcs7signV2(file_#n-1, session_ID)
pkcs7signV2(file_#n, session_ID)
closesession(session_ID)

```

Please find below details of how to use each type of method:

```
public String opensession(Auth identity)
```

Method used to open the multiple signature session

INPUT

- **identity** - Auth object containing the following parameters for the signature request
- **TypeHSM** = "COSIGN"
- **TypeOtpAuth** = String used to indicate the authentication domainsignature If a remote signature user is being used. If a remote signature user is being used with a customized domain, you need to specify this domain (e.g. frXXXX). If an automatic signature user is being used, you need to specify the string shown when installing ARSS.
- **User** = String containing the signature user's username.
- **UserPWD** = String containing the signature user's password.
- **OtpPWD** = String containing the user's valid OTP code for the signature transaction.

OUTPUT

String containing the ID for the session that has just been opened if the operation has been successful.

If the operation has failed, one of the following strings will be returned:

```

"KO-0001" Generic error
"KO-0003" Error during the credential verification phase
"KO-0004" PIN error
"KO-0009" Invalid delegation credentials
"KO-0010" Invalid user status (e.g. user suspended)

```

```
public SignReturnV2 pkcs7signV2 (SignRequestV2 request, boolean detached)
```

The pkcs7signV2 method will be launched **n** times when **n** is the number of files to be signed.

INPUT The details of launching this method follow the details given in the paragraph entitled *Digital Signatures* with the following changes, according to the context, for the multiple signatures process:

1. Set the SignRequestV2.Session_id property with the session ID returned in the output of the call before the opensession
2. Set the SignRequestV2.Identity property with the signatory's authentication details

N.B.: In this case, you do not need to set the SignRequestV2.Identity.OtpPWD property as the user is authenticated via the three key elements (User, UserPWD, SessionID)

1. Add the individual file to which you want to add the signature to the input

OUTPUT

The output for each call follows the details given in the corresponding section of the paragraph entitled *Digital Signatures*

```
public String closesession(Auth identity, String sessionid)
```

Method used to close the multiple signature session.

INPUT

- **identity** - Auth object containing the following parameters, already specified when the session was opened
- **TypeHSM** = "COSIGN"
- **TypeOtpAuth** = String used to indicate the authentication domainsignature If a remote signature user is being used. If a remote signature user is being used with a customized domain, you need to specify this domain (e.g. frXXXX). If an automatic signature user is being used, you need to specify the string shown when installing ARSS.
- **User** = String containing the signature user's username.
- **UserPWD** = String containing the signature user's password.
- **OtpPWD** = Not necessary, because authentication is based on the sessionID.
- **ext_auth_type** = Enum that allows you to use the extended authentication method. (**Deprecated** see *authMethods* method instead)
- **ext_auth_blobvalue** = Array of bytes in which to add the authentication credential (encrypted challenge, etc.) for the *extended authentication methods* that allow for the exchange of binary data
- **ext_auth_value** = String in which to add the authentication credential (OTP or similar) for the *extended authentication methods* that allow for the exchange of data in plain text

- **delegated_user** = String that contains the username for the delegated user if the automatic signature has been delegated. In this case, the password for the Owner of the Signature key does not need to be specified
- **delegated_password** = String that contains the delegated user's password
- **delegated_domain** = Delegated user's domain (used in a similar way to the TypeOtpAuth parameter)
- **Sessionid** - String containing the ID for the session that you intend to close. Typically contains the same string returned by the initial opensession call corresponding to the multiple signature session

OUTPUT

String containing the outcome of the process:

- "OK" Generic error
- "KO-0001" Generic error

Multiple signature methods

The multiple signature methods are the same in form and substance as the analogous methods used for individual signatures. The only difference is the fact that they accept an incoming **array of requests** and return an **array of responses**. Please find below details of how to use each type of method:

public SignReturnV2_multiple pkcs7signhash_multiple (Auth identity, SignRequestV2[] requestList, boolean countersignature)
public SignReturnV2_multiple pkcs7signV2_multiple(Auth identity, SignRequestV2[] requestList, boolean detached)
public SignReturnV2_multiple pdfsignatureV2_multiple(Auth identity, SignRequestV2[] requestList, PdfSignApparence apparence, PDFProfile pdfprofile)
public SignReturnV2_multiple xmlsignature_multiple(Auth identity, SignRequestV2[] requestList, XmlSignatureParameter parameter)

Warning: the number of file that can be signed in a single multiple method could be different case by case, depending on

- signature type (remote or automatic)
- signature format (cades, pades, xades)
- file sizes
- strategy (best effort signatures or granted signatures)
- actual workload
- network state
- session duration

Usually those methods should be used in an automatic signature scenario.

In remote signature context it's suggested to

1. open a session
2. loop and apply n signatures
3. close the session

This because when many documents (more than 20-30) are sent in a single signature session, the request received by ARSS requires more time to be processed.

In this case the OTP spread validity could not be sufficient and the session could not be open.

INPUT

- **identity** = Auth object containing the data to authenticate the user that will be used to prepare all signature requests contained in the array (see Paragraph 5.1)
- **SignRequestV2[]** = Array of SignRequestV2 objects containing the same parameters for the signature request described in paragraph 5.1
- **detached** = Boolean value used to indicate the generation of a detached CAd envelope (see Paragraph 5.1)

OUTPUT

- **SignReturnV2_multiple** = Complex object made up of the following fields
 - **status** = Overall outcome of the multiple signature operation.
 - **description** = Additional description of the overall status
 - **return_code** = Overall return code
 - **SignReturnV2[]** = Array of SignReturnV2 objects associated with the requests

Automatic Signature

To create an automatic signature, all you need to do is use the pkcs7signV2() method, adding the authentication credentials of an automatic signature user to the input. N.B.: To generate an automatic signature successfully, the user must first have enabled the option via the activation pages (<https://attivazioni.firma-remota.it/asmonitor/>)

public SignReturnV2 pkcs7signV2 (SignRequestV2 request, boolean detached, Boolean returnder)
--

INPUTThe details of launching the pkcs7signV2() method follow the details given in the relevant section of the paragraph entitled *Digital Signatures* with the following changes, according to the context, for the automatic signatures process:

1. Set the SignRequestV2.Identity.TypeOtpAuth property with the string identifying the automatic signature domain.

N.B.: This string is indicated when ARSS is installed

1. Set the SignRequestV2.Identity.OtpPWD property with the string identifying the automatic signature transactions

N.B.: This static string is defined when the ARSS server is installed, and is normally known by the administrator of the IT infrastructure network on which users are working.

OUTPUTThe output returned by the function call follows the details given in the corresponding section of the paragraph entitled *Digital Signatures*

Joint Signature (parallel signature)

To create a joint signature, all you need to do is use the addpkcs7sign() method. The method is defined as:

```
public SignReturnV2 addpkcs7sign (SignRequestV2 request, boolean detached)
```

INPUTThe details of launching the addpkcs7sign() method follow the details given in the instructions for launching the pkcs7signV2() method (see the paragraph entitled *Digital Signatures*) with the following changes, according to the context, for the parallel signatures process:

1. The method only accepts cryptographic envelopes in CADES or PKCS#7 format as input
2. The method can also accept folders containing only files as described in point 1 as input
3. The detached parameter must not be set.

OUTPUTThe output returned by the function call follows the details given in the instructions for the pkcs7signV2() method in the corresponding section of the paragraph entitled *Digital Signatures* with the following changes, according to the context, for the parallel signature process:

1. If files have been input in a format other than CADES or PKCS#7 the error given in the SignReturnV2.Return_code property is "0001" (Generic error in the signature process)

Digital Signature of a fingerprint

If you intend to sign the fingerprint of a document (or a generic sequence of bytes) you simply use the pkcs7signhash() method. This method is defined as follows:

```
public SignReturnV2 pkcs7signhash(SignRequestV2 request, boolean countersignature, boolean excludeSigningTime)
```

This method is normally used for the following:

- Adding a countersignature
- Generating "detached" digital signatures, i.e. separate from the signed file

INPUT

- **countersignature** – Boolean value used to generate a detached CADES envelope from which it is possible to extract the objects needed for the raw construction of a countersignature.
More details about the process for implementing the countersignature can be found in the OUTPUT section of this paragraph
 - false = request to generate a detached CADES signature to construct a standard signature cryptographic envelope
 - true = request to generate a detached CADES signature to construct a countersignature
- **excludeSigningTime** – Boolean value used to exclude the signed attribute (signing-time) of a detached CADES envelope. Useful for constructing a PAdES envelope from the fingerprint
 - false = inclusion of the signed attribute in the detached CADES signature envelope
 - true = exclusion of signed attribute in the detached CADES signature envelope
- **request** - SignRequestV2 object set in accordance with the instructions in the relevant section of the paragraph entitled *Digital Signatures* with the following changes, according to the context for the process for signing a fingerprint.

1. The Transport property must be set to BYNARYNET
2. The SrcName and DstName properties should not be used
3. The Binaryinput property must contain the 32 bytes that represent the SHA-256 fingerprint of the object to be signed (typically the fingerprint of the document that you intend to sign)

OUTPUT

SignReturnV2 object containing the following parameters:

- **Status** = String containing the outcome of the signature process
Further details are included within the SignReturnV2.description and SignReturnV2.return_code properties.
 - OK Operation completed successfully
 - Different from OK In case of error.

- **Binaryoutput** = Byte array containing the "detached" CAdES envelope for the hash specified during the call.
N.B.: In the signedattribute SignerInfo.messageDigest this envelope contains the 32 bytes specified in the Binaryinput array
- **Stream** = DataHandler to download large files in Stream formatOnly used when Transport = STREAM
N.B.: The solution is only available with JAX-WS compliant clients.
- **DstPath** = Not used
- **Return_code** = Enum containing the error code
 - "0001" Generic error in the signature process
 - "0002" Incorrect parameters for the type of transport indicated
 - "0003" Error during the credential verification phase
 - "0004" PIN error (more details in the SignReturnV2.description)
 - "0005" Invalid type of transport
 - "0006" Unauthorized type of transport
 - "0008" Impossible to complete the timestamping operation (e.g. impossible to connect to the service, remaining stamps ended etc.)
 - "0009" Invalid delegate credentials
 - "0010" Invalid user status (e.g. user suspended)
- **Description** = String containing a high level description of the error encountered and/or any details relating to the codes returned within the SignReturnV2.return_code property

Countersignatures

If the countersignature boolean value is set as TRUE, you can use the output returned by the pkcs7signhash() method for the source of the objects needed to construct a countersignature as follows: Mark the CAdES file to which you want to add the countersignature as 'Envelope_A', and the detached CAdES file returned by the pkcs7signhash() command as 'Envelope_B', and below you will find a brief description of the steps to be followed to construct a countersignature

1. Identify the SignerInfo structure for the signature to be countersigned in Envelope_A
2. Extract the 128 bytes corresponding to the value of the SignerInfo.signature field (i.e. excluding the tags and octet lengths)
3. Calculate the SHA256 hash for the 128 bytes obtained in the step above and use the 32 bytes output to set the Binaryinput byte array
4. Follow the pkcs7signhash() method and set the countersignature parameter to true
5. From the resulting Envelope_B, extract the only SignerInfo structure present, and use it as the value for the countersignature unsigned-attribute for Envelope_A
6. From Envelope_B extract the only certificate present in the SignedData.certificates field and add it to the corresponding field for Envelope_A
7. Update the octet lengths for Envelope_A and save the result

Remote Digital Signatures in the PAdES (pdf) standard

Digital signatures

The method to be used for generating a digital signature for an individual file is pdfsignatureV2(). **From version 1.14.0** the method automatically includes the signature fonts within the document, preserving the PDF/A format. **However, the PDF/A format does not support transparent images, so if the Image or ImageBin fields contain a transparent image, the method will produce a PDF that does not match the PDF/A format. The method only accepts PDF files as input and is defined as follows:**

```
public SignReturnV2 pdfsignatureV2(SignRequestV2 request, PdfSignApparence apparence, enum Pdfprofile, String password, DictionarySignedAttributes dict_signed_attributes)
```

INPUT

- **request** - SignRequestV2 object containing the following parameters for the signature request
 - **transport** = Enum that indicates the type of input (from here on transport type)
 - BYNARYNET If the array of bytes containing the file to sign is being added to the input
 - FILENAME If the path of the file to be signed is being added to the input
 - DIRECTORYNAME If the path of the folder with the files to be signed is being added to the input
 - STREAM If the stream for a file in MTOM format with JAX-WS java technology (for large files) is being added to the input
 - **binaryinput** = Byte array containing the document to be signed. Only used when Transport = BINARYNET
 - **srcName** = String containing the path of the file or folder containing the files to be signed. Only used when Transport = FILENAME or DIRECTORYNAME. (N.B.: The resource indicated must be accessible from the ARSS server)
 - **stream** = DataHandler to upload large files in Stream format. Only used when Transport = STREAM. (N.B.: The solution is only available with JAX-WS compliant clients.)
 - **dstName** = String containing the destination path of the file or folder for the signed files. Only used when Transport = FILENAME or DIRECTORYNAME. (N.B.: The path specified must be accessible from the ARSS server)
 - **requiredmark** = Boolean value used to guide the generation of a PAdES-T envelope: digital signature with contextual addition of the timestamp. (N.B.: If you intend to have a timestamp in PAdES-T format, the ARSS server must first be configured with a valid timestamping account.)
 - TRUE If you intend to generate a PAdES-T
 - FALSE If not
 - **signingTime** = String used to set the entry /M for the **Signature Dictionary** object for the PAdES envelope. (N.B.: ARSS will insert the date on the system of the Server being used.)
 - dd/MM/yyyy HH:mm:ss If you intend to specify the signature date explicitly
 - NULL If you intend to refer the attribute value to ARSS.
 - optionally the signing time can include the timezone (eg. dd/MM/yyyy HH:mm:ss zz, 05/10/2023 14:59:00 CEST)

- **session_id** = String used to specify the session ID within multiple signature transactions. For a signature for an individual file or a folder, do not specify or set AS NULL.
- **identity** = Auth object containing the data to authenticate the signatory
- **typeHSM** = String containing the type of HSM. Always applied with the 'COSIGN' string
- **typeOtpAuth** = String used to indicate the authentication domain. If a remote signature user is being used with a customized domain, you need to specify this domain (e.g. frXXXX). If an automatic signature user is being used, you need to specify the string shown when installing ARSS.
 - signature If a remote signature user is being used.
- **user** = String containing the signature user's username.
- **userPWD** = String containing the signature user's password.
- **otpPWD** = String containing the user's valid OTP code for the signature transaction.
- **ext_authtype** = Enum that allows you to use the extended authentication method. (**Deprecated see authMethods** method instead)
- **ext_auth_blobvalue** = Array of bytes in which to add the authentication credential (encrypted challenge, etc.) for the *extended authentication methods* that allow for the exchange of binary data
- **ext_auth_value** = String in which to add the authentication credential (OTP or similar) for the *extended authentication methods* that allow for the exchange of data in plain text
- **delegated_user** = String that contains the username for the delegated user if the automatic signature has been delegated. In this case, the password for the Owner of the Signature key does not need to be specified
- **delegated_password** = String that contains the delegated user's password
- **delegated_domain** = Delegated user's domain (used in a similar way to the TypeOtpAuth parameter)
- **tsa_identity** = Optional TSAAuth object, containing the data for TSS Server authentication. If not specified, the server will use the default account, if configured.
- **user** = String containing the timestamp user's username
- **password** = String containing the timestamp user's password
- **notifyemail** = String used to specify the email address to which notification of the completion of the signature operation is sent.
- **notify_id** = String containing the ID that the user intends to link to the signature session for the purposes of notification of the completion of the operation. Only used when Transport = DIRECTORYNAME. For a signature for an individual file, this property can be omitted or set as NULL.
- **certID** = RFU. Set with AS0
- **profile** = RFU. Set to NULL
- **appearance** – PdfSignAppearance object containing further parameters to generate the PAdES signature:
 - **location** = String used to set the entry **/Location** for the **Signature Dictionary** object. For example, 'Rome'
 - **reason** = String used to set the entry **/Reason** for the **Signature Dictionary** object. For example 'For approval'
 - **page** = Integer that indicates the page number on which the signature needs to be inserted. This field is required and must be > 0 (handled exception otherwise)
 - **leftx** = Integer used to specify the x-coordinate for the bottom-left margin of the signature rectangle. The unit of measurement is expressed in pixels
 - **lefty** = Integer used to specify the y-coordinate for the bottom-left margin of the signature rectangle. The unit of measurement is expressed in pixels
 - **rightx** = Integer used to specify the x-coordinate for the top-right margin of the signature rectangle. The unit of measurement is expressed in pixels
 - **righty** = Integer used to specify the y-coordinate for the top-right margin of the signature rectangle. The unit of measurement is expressed in pixels
 - **image** = String indicating the path of the image to be used as the background in the signature rectangle. The path indicated must point to a graphic resource in PNG/GIF/JPG format accessible from the ARSS server
 - **imageBin** = Specifies the byte array for the image to be used as the background for the signature rectangle. If specified, it takes precedence over the Image parameter. Available from version 1.14.0.
 - **text** = String that allows you to specify free text instead of the standard text shown in the visible part of signatures.
 - **preservePDFA** = Boolean value, if true, the images in the Image or ImageBin fields are converted by deleting any transparency, so that the PDF/A format is preserved.
 - **imageOnly** = Boolean value, if true, only inserts the image.
 - **bScaleFont** = Boolean value, scales the size of the font of the text to fit inside the width of the appearance box.
 - **bShowDateTime** = Boolean value, shows or hides the signature date and time (signing time) in the appearance box.
 - **resizeMode** = Integer that specifies the method used to resize the image. 1 (resize using both sides of the image - stretch), 2 (resize using the longer side and center), 3 (resize using the longer side and right align), 4 (resize using the longer side and left align), 5 do not resize
- **fieldName** = preset field to be used to insert a graphical signature into. If fieldName is present then appearance can be omitted, or its box coordinates can be ignored
- **pdfprofile** = Enum of possible types of PDF supported:
 - BASIC Basic PDF Signature
 - PAdESBES Pades-Bes PDF Signature
 - PAdESLTV Pades-LT PDF Signature
- **password** = owner's password to be specified if the document is password protected.
- **dict_signed_attributes** - DictionarySignedAttributes object that allows you to specify signed properties in the PDF signature dictionary. Available from version 1.14.0
 - **T** = Indicates the value of the /T attribute in the signature dictionary (nominal signature).
- **signatureLevel** – Signature level required. (Permitted Values: LT)

OUTPUT

SignReturnV2 object containing the following parameters:

- **Status** = String containing the outcome of the signature process
 - OK Operation completed successfully
 - Different from OK In case of error. Further details are included within the SignReturnV2.description and SignReturnV2.return_code properties.

- **Binaryoutput** = Byte array containing the signed file
- **Stream** = DataHandler to download large files in Stream format. Only used when Transport = STREAM. (N.B.: The solution is only available with JAX-WS compliant clients.)
- **DstPath** = String containing the path specified within the SignRequestV2.DstName parameter during the request phase. If the file specified in SignRequestV2.DstName already exists, the SignReturnV2.DstPath property contains the path to the file actually generated by ARSS.
- **Return_code** = Enum containing the error code
 - "0001" Generic error in the signature process
 - "0002" Incorrect parameters for the type of transport indicated
 - "0003" Error during the credential verification phase
 - "0004" PIN error (more details in the SignReturnV2.description)
 - "0005" Invalid type of transport
 - "0006" Unauthorized type of transport
 - "0007" Invalid PDF signature profile
 - "0008" Impossible to complete the timestamping operation (e.g. impossible to connect to the service, remaining stamps ended etc.)
 - "0009" Invalid delegate credentials
 - "0010" Invalid user status (e.g. user suspended)
 - "0031" Invalid page index, must be greater than 0
- **Description** = String containing a high level description of the error encountered and/or any details relating to the codes returned within the SignReturnV2.return_code property

Multiple Signatures

To generate a digital signature for a group of files in PDF format, the same considerations apply as those already outlined in the paragraph entitled *Multiple Signatures* in the chapter entitled **Remote Digital Signatures in pkcs1 Digital Signature** of a fingerprint in accordance with the pkcs1 standard. If you intend to sign the fingerprint of a document (or a generic sequence of bytes) based on the pkcs1 standard, you simply need to use the `signhash()` method. This method is defined as follows:

```
public SignHashReturn signhash(SignHashRequest request)
```

This method is normally used for the following:

- Generating "detached" digital signatures, i.e. separate from the signed file

INPUT

- **request** - SignHashRequest object containing the following parameters for the signature request:
 - **certID** = Unique certificate ID to be used for the signature. The list of IDs can be retrieved using the listCert function explained below. If a specific certificate is not needed, you can specify AS0 to balance the requests, although this method assumes that the certificate itself does not contribute to the calculation of the HASH (not valid for CADES signatures).
 - **hash** = Array of bytes containing the fingerprint for the document to be signed.
 - **hashtype** = String containing the name of the hash algorithm used to calculate the fingerprint (currently only "SHA256" available).
 - **identity** = Auth object containing the data to authenticate the signatory
 - **TypeHSM** = String containing the type of HSM. Always applied with the 'COSIGN' string
 - **TypeOtpAuth** = String used to indicate the authentication domain. If an automatic signature user is being used, you need to specify the string shown when installing ARSS.
 - signature If a remote signature user is being used.
 - **User** = String containing the signature user's username.
 - **UserPWD** = String containing the signature user's password.
 - **OtpPWD** = String containing the user's valid OTP code for the signature transaction.
 - **ext_authtype** = Enum that allows you to use the extended authentication method. (**Deprecated** see *authMethods* method instead)
 - **ext_auth_blobvalue** = Array of bytes in which to add the authentication credential (encrypted challenge, etc.) for the *extended authentication methods* that allow for the exchange of binary data
 - **ext_auth_value** = String in which to add the authentication credential (OTP or similar) for the *extended authentication methods* that allow for the exchange of data in plain text
 - **delegated_user** = String that contains the username for the delegated user if the automatic signature has been delegated. In this case, the password for the Owner of the Signature key does not need to be specified
 - **delegated_password** = String that contains the delegated user's password
 - **delegated_domain** = Delegated user's domain (used in a similar way to the TypeOtpAuth parameter)
 - **session_id** = String used to specify the session ID within multiple signature transactions. For a signature for an individual file or a folder, do not specify or set AS NULL.
 - **requirecert** = Boolean value that indicates the requirement for the presence of the signatory's certificate in the response.

OUTPUT

SignHashReturn object containing the following parameters:

- **Status** = String containing the outcome of the signature process
 - OK Operation completed successfully
 - Different from OK In case of error. Further details are included within the SignHashReturn.Description.
- **Description** = String containing a high level description of the error encountered and/or any details relating to the codes returned within the SignReturnV2.return_code property.
- **Return_code** = Enum containing the error code
 - "0001" Generic error in the signature process

- "0002" Incorrect parameters for the type of transport indicated
- "0003" Error during the credential verification phase
- "0004" PIN error (more details in the SignReturnV2.description)
- "0005" Invalid type of transport
- "0006" Unauthorized type of transport
- "0007" Invalid PDF signature profile
- "0008" Impossible to complete the timestamping operation (e.g. impossible to connect to the service, remaining stamps ended etc.)
- "0009" Invalid delegate credentials
- "0010" Invalid user status (e.g. user suspended)
- "0031" Invalid page index, must be greater than 0
- **signature** = Array of bytes containing the outcome of the signature process.
- **cert** = Array of bytes containing the recipient's certificate if requested see requirecert parameter.
- **certID** = ID for the signatory's certificate.

Remote Digital Signatures in the CAdES (p7m) standard The only differences compared to the description for CAdES signatures are as follows:

1. The method to be used for launching the signature function must be pdfsignatureV2() and not pkcs7signV2()
2. When the transport type is DIRECTORYNAME only files with features that fit in with the parameters specified within the appearance object are processed.
3. The files that are not processed will be summarized in the notification email that will be sent at the end of the process to the address specified in Notifyemail

Automatic Signature

To create an automatic signature in PDF format, simply use the pdfsignatureV2() method, adding the authentication credentials of an automatic signature user to the input.

INPUT

The details of launching the pdfsignatureV2() method follow the details given in the relevant section of the paragraph entitled *Digital Signatures* with the following changes, according to the context, for the automatic signatures process:

1. Set the SignRequestV2.Identity.TypeOtpAuth property with the string identifying the automatic signature domain.

N.B.: This string is indicated when ARSS is installed

1. Set the SignRequestV2.Identity.OtpPWD property with the string identifying the automatic signature transactions

N.B.: This static string is defined when the ARSS server is installed, and is normally known by the administrator of the IT infrastructure network on which users are working.

OUTPUT

The output returned by the function call follows the details given in the corresponding section of the paragraph entitled *Digital Signatures*

Joint Signature (parallel signature)

The PAdES standard does not allow for the use of so-called joint signatures, in other words signatures with different signatories to the same content. In PDF format, each signature is actually inserted in append mode using the so-called "Incremental Update" system typical of the PDF format, and therefore covers the whole content of the input PDF file, including any existing signatures. Below you will find a diagram illustrating this mechanism. ! worddav92bbde2422f2d9bb8f9d2b444d6c8b98.png|height=388,width=468! Consequently, the only signature operation that can be completed for a PAdES file is the addition of a signature in "Enveloped" mode, simply by using the pdfsignatureV2() method as described in the paragraph entitled *Digital Signatures* in the Chapter entitled **Remote Digital Signatures in the PAdES (pdf) standard**. The only difference is the fact that a file in PAdES format must be added to the input.

Signature with a signature field that can be filled in

The following call is available in the SignService to add multiple PDF signatures to the same document and use pre-completed signature fields.

```
public SignResponse pdfSignature(PdfSignRequest request)
```

INPUT

- **Binaryinput** = Byte array containing the document to be signed. Only used when Transport = BINARYNET.
N.B. = The input file must be a well-formed xml file
- **DstName** = String containing the destination path of the file or folder for the signed files. Only used when Transport = FILENAME or DIRECTORYNAME
- **Notify_id** = String containing the ID that the user intends to link to the signature session for the purposes of notification of the completion of the operation. Only used when Transport = DIRECTORYNAME
- **SrcName** = String containing the path of the file or folder containing the files to be signed. Only used when Transport = FILENAME or DIRECTORYNAME

- **Stream** = DataHandler to download large files in Stream format
N.B. Only used when Transport = STREAM
N.B. The solution is only available with JAX-WS compliant clients.
- **Transport** = Enum that indicates the type of input (from here on transport type)
 - BYNARYNET If the array of bytes containing the file to sign is being added to the input
 - FILENAME If the path of the file to be signed is being added to the input
 - DIRECTORYNAME If the path of the folder with the files to be signed is being added to the input
 - STREAM If the stream for a file in MTOM format with JAX-WS java technology (for large files) is being added to the input
- **Identity** = Auth object containing the data to authenticate the signatory
- **delegated_domain** = Delegated user's domain (used in a similar way to the TypeOtpAuth parameter)
- **delegated_password** = String that contains the delegated user's password
- **delegated_user** = String that contains the username for the delegated user if the automatic signature has been delegated. In this case, the password for the Owner of the Signature key does not need to be specified
- **ext_auth_blobvalue** = Array of bytes in which to add the authentication credential (encrypted challenge, etc.) for the *extended authentication methods* that allow for the exchange of binary data
- **ext_auth_value** = String in which to add the authentication credential (OTP or similar) for the *extended authentication methods* that allow for the exchange of data in plain text
- **ext_authtype** = Enum that allows you to use the extended authentication method. (**Deprecated** see *authMethods* method instead)
- **OtpPWD** = String containing the user's valid OTP code for the signature transaction.
- **TypeHSM** = String containing the type of HSM. Always applied with the 'COSIGN' string
- **TypeOtpAuth** = String used to indicate the signature authentication domain. If a remote signature user is being used. If a remote signature user is being used with a customised domain, it is necessary to specify this domain (e.g. frXXXX). If an automatic signature user is being used, you need to specify the string shown when installing ARSS.
- **user** = String containing the timestamp user's username
- **UserPWD** = String containing the signature user's password. For a signature for an individual file, this property can be omitted or set as NULL.
- **CertID** = RFU. Set with AS0
- **requiredmark** = Boolean value used to guide the generation of a PAdES-T envelope: digital signature with contextual addition of the timestamp.
N.B.: If you intend to have a timestamp in PAdES-T format, the ARSS server must first be configured with a valid timestamping account.
 - TRUE If you intend to generate a PAdES-T
 - FALSE If not
- **signingTime** = String used to set the entry /M for the **Signature Dictionary** object for the PAdES envelope.
 - dd/MM/yyyy HH:mm:ss If you intend to specify the signature date
 - NULL If you intend to refer the attribute value to ARSS.
 - optionally the signing time can include the timezone (eg. dd/MM/yyyy HH:mm:ss zz, 05/10/2023 14:59:00 CEST)
- **sessionID** = String used to specify the session ID within multiple signature transactions. For a signature for an individual file or a folder, do not specify or set AS NULL.
- **tsaIdentity** = Optional TSAAuth object, containing the data for TSS Server authentication. If not specified, the server will use the default account, if configured.
 - **user** = String containing the timestamp user's username
 - **password** = String containing the timestamp user's password
 - **tsaurl** = Contains the TSA address
- **appearance** = PdfSignAppearance object containing further parameters to generate the PAdES signature
- **Image** = String indicating the path of the image to be used as the background in the signature rectangle. The path indicated must point to a graphic resource in PNG/GIF/JPG format accessible from the ARSS server
- **ImageBin** = Specifies the byte array for the image to be used as the background for the signature rectangle. If specified, it takes precedence over the Image parameter. Available from version 1.14.0.
- **ImageOnly** = Boolean value, if true, only inserts the image.
- **Leftx** = Integer used to specify the x-coordinate for the bottom-left margin of the signature rectangle. The unit of measurement is expressed in pixels
- **Lefty** = Integer used to specify the y-coordinate for the bottom-left margin of the signature rectangle. The unit of measurement is expressed in pixels
- **Location** = String used to set the /Location entry for the Signature Dictionary object. For example, 'Rome'
- **Page** = Integer that indicates the page number on which the signature needs to be inserted. This field is required
- **Reason** = String used to set the /Reason entry for the Signature Dictionary object. For example 'For approval'
- **Rightx** = Integer used to specify the x-coordinate for the top-right margin of the signature rectangle. The unit of measurement is expressed in pixels
- **Righty** = Integer used to specify the y-coordinate for the top-right margin of the signature rectangle. The unit of measurement is expressed in pixels
- **Text** = String that allows you to specify free text instead of the standard text shown in the visible part of signatures.
- **bScaleFont** = Boolean value, scales the size of the font of the text to fit inside the width of the appearance box.
- **bShowDateTime** = Boolean value, shows or hides the signature date and time (signing time) in the appearance box.
- **resizeMode** = Integer that specifies the mode used to resize the image. 1 (resize using both sides of the image - stretch), 2 (resize using the longer side and centre), 3 (resize using the longer side and right align), 4 (resize using the longer side and left align), 5 do not resize
- **preservePDFA** = Boolean value, allows you to preserve the PDF/A
- **fieldName** = String indicating the name of the pre-filled signature field on which to create the PDF appearance.
- **Pdfprofile** = Enum of possible types of PDF supported:
 - BASIC Basic PDF Signature
 - PAdESBES Pades-Bes PDF Signature
 - PAdESLTV Pades-LT PDF Signature
- **dictSignedAttributes** - DictionarySignedAttributes object that allows you to specify signed properties in the PDF signature dictionary. Available from version 1.14.0
- **T** = Indicates the value of the /T attribute in the signature dictionary (nominal signature).

PDF signatures with DTS timestamp

This method allows you to sign a PDF by adding a DTS timestamp, which is a particular type of timestamp that is displayed as a signature. This can be done using the pdfsignatureDTS call.

The method is defined as follows:

```
public MarkReturn pdfsignatureDTS(MarkRequest request, HttpServletRequest srvrequest, String password)
```

INPUT

- **request** - SignRequestV2 object containing the following parameters for the signature request
 - **MarkRequest**- MarkRequest object containing the following parameters for the timestamp request
 - **Transport** = Enum that indicates the type of input (from here on transport type)BYNARYNET If the array of bytes containing the file to sign is being added to the inputFILENAME If the path of the file to be timestamped is being added to the inputDIRECTORYNAME If the path of the folder with the files to be timestamped is being added to the inputSTREAM If the stream for a file in MTOM format with JAX-WS java technology (for large files) is being added to the input
 - **Binaryinput** = Byte array containing the document to be timestamped. Only used when Transport = BINARYNET
 - **SrcName** = String containing the path of the file or folder containing the files to be timestamped. Only used when Transport = FILENAME or DIRECTORYNAME N.B.: The resource indicated must be accessible from the ARSS server
 - **Stream** = DataHandler to upload large files in Stream formatOnly used when Transport = STREAMN.B.: The solution is only available with JAX-WS compliant clients.
 - **DstName** = String containing the destination path of the file or folder for the timestamped files. Only used when Transport = FILENAME or DIRECTORYNAME N.B.: The path specified must be accessible from the ARSS server
 - **Notifymail** = String used to specify the email address to which notification of completion of the timestamping operation is sent.
 - **Notify_id** = String containing the ID that the user intends to link to the timestamp session for the purposes of notification of completion of the operation. Only used when Transport = DIRECTORYNAME For a timestamp for an individual file, this property can be omitted or set as NULL.
 - **User** = String containing the timestamp user's username
 - **Password** = String containing the timestamp user's password
 - **binaryinput** = Byte array containing the document to be signed. Only used when Transport = BINARYNET
 - **user** = String containing the timestamp user's username
 - **notify_id** = String containing the ID that the user intends to link to the signature session for the purposes of notification of the completion of the operation. Only used when Transport = DIRECTORYNAME. For a signature for an individual file, this property can be omitted or set as NULL.
 - **notifymail** = String used to specify the email address to which notification of the completion of the signature operation is sent.
 - **stream** = DataHandler to upload large files in Stream format. Only used when Transport = STREAM. (N.B.: The solution is only available with JAX-WS compliant clients.)
 - **dstName** = String containing the destination path of the file or folder for the signed files. Only used when Transport = FILENAME or DIRECTORYNAME. (N.B.: The path specified must be accessible from the ARSS server)
 - **transport** = Enum that indicates the type of input (from here on transport type)
 - BINARYNET If the array of bytes containing the file to sign is being added to the input
 - FILENAME If the path of the file to be signed is being added to the input
 - DIRECTORYNAME If the path of the folder with the files to be signed is being added to the input
 - STREAM If the stream for a file in MTOM format with JAX-WS java technology (for large files) is being added to the input
 - **srcName** = String containing the path of the file or folder containing the files to be signed. Only used when Transport = FILENAME or DIRECTORYNAME. (N.B.: The resource indicated must be accessible from the ARSS server)
 - **url** = String containing the url of the reference tsa
 - **password** = String containing the timestamp user's password

OUTPUTSignReturnV2 object containing the following parameters:

- **Status** = String containing the outcome of the signature process
 - OK Operation completed successfully
 - Different from OK In case of error. Further details are included within the SignReturnV2.description and SignReturnV2.return_code properties.
- **Binaryoutput** = Byte array containing the signed file
- **Stream** = DataHandler to download large files in Stream format. Only used when Transport = STREAM. (N.B.: The solution is only available with JAX-WS compliant clients.)
- **DstPath** = String containing the path specified within the SignRequestV2.DstName parameter during the request phase. If the file specified in SignRequestV2.DstName already exists, the SignReturnV2.DstPath property contains the path to the file actually generated by ARSS.
- **Return_code** = Enum containing the error code
 - "0001" Generic error in the signature process
 - "0002" Incorrect parameters for the type of transport indicated
 - "0003" Error during the credential verification phase
 - "0004" PIN error (more details in the SignReturnV2.description)
 - "0005" Invalid type of transport
 - "0006" Unauthorized type of transport
 - "0007" Invalid PDF signature profile

- "0008" Impossible to complete the timestamping operation (e.g. impossible to connect to the service, remaining stamps ended etc.)
- "0009" Invalid delegate credentials
- "0010" Invalid user status (e.g. user suspended)
- **Description** = String containing a high level description of the error encountered and/or any details relating to the codes returned within the SignReturnV2.return_code property

PDF signatures (Deprecated)

This method allows you to sign a PDF multiple times. This can be done using the pdfsignature call.

The method is defined as follows:

```
public SignResponse pdfsignature(HttpServletRequest srvrequest, PdfSignRequest request)
```

INPUT

- **request** - SignRequestV2 object containing the following parameters for the signature request
 - **password** = The PDF password
 - **isPdfA** = Whether the PDF should be saved as PDF-A or not (**ignored**)
 - **binaryInput** = Byte array that contains the document to sign or the hash for hash signature
 - **dstName** = Destination file path or destination directory path
 - **notifyId** = ID associated to the signatures session for notifications
 - **notifyMail** = Email for notifications
 - **srcName** = Path to file or directory path that contains the files to be signed
 - **stream** = Stream used to load large size file
 - **inputTransport** = Type of input
 - BYNARYNET If the array of bytes containing the file to sign is being added to the input
 - FILENAME If the path of the file to be signed is being added to the input
 - DIRECTORYNAME If the path of the folder with the files to be signed is being added to the input
 - STREAM If the stream for a file in MTOM format with JAX-WS java technology (for large files) is being added to the input
 - **outputTransport** = Type of output
 - BYNARYNET If the array of bytes containing the file to sign is being added to the input
 - FILENAME If the path of the file to be signed is being added to the input
 - DIRECTORYNAME If the path of the folder with the files to be signed is being added to the input
 - STREAM If the stream for a file in MTOM format with JAX-WS java technology (for large files) is being added to the input
 - **signaturesToDo** = The signatures to do, each with some specific parameters
 - **identity** = Object that contains signer credentials for authentication
 - **certId** = Certificate id, for future use. For now set to "AS0"
 - **requiredMark** = If true then a temporal mark is required
 - **signingTime** = Signature date (time format is "dd/MM/yyyy HH:mm:ss"). If null, the current date is used. Optionally the signing time can include the timezone (eg. dd/MM/yyyy HH:mm:ss zz, 05/10/2023 14:59:00 CEST)
 - **sessionId** = Session id used to create multiple signatures without specify PIN for every signature
 - **tsaIdentity** = Object that contains authentication data for TSA Server
 - **appearance** = The PDF appearance in case of visible signature
 - **fieldName** = The AcroForm field name
 - **pdfProfile** = The PDF profile
 - **dictSignedAttributes** = The dictionary signed attributes

OUTPUT SignResponse object containing the following parameters:

- **binaryOutput** = Byte array of signed file
- **dstPath** = File path specified in request used by server
- **Stream** = Stream used to download large size file
- **status** = String containing the outcome of the signature process
 - OK Operation completed successfully
 - Different from OK In case of error.
- **code** = String containing the error code
- **message** = String containing the error message

Remote Digital Signatures in the XAdES (xml) standard

Brief recap on XML signatures

XML is an incredibly commonly used format for exchanging data efficiently and flexibly between applications interacting online. The "XML-Signature Syntax and Processing" standard (<http://www.w3.org/TR/xmldsig-core/>) defines the specifications that allow you to "sign" XML documents, or parts of them, guaranteeing their authenticity and integrity. This means that applications can generate and verify digital signatures in a similar way to the process for CAdES and PAdES envelopes. It is important to highlight the fact that a signed XML document is still an XML document.

There are five steps in the XML signature process:

1. identify the document to be signed;
2. convert it into a "canonical" format;
3. apply any transformations (XPath, XSLT);
4. create a summary (digest) of the results of the transformation;
5. sign the digest.

XML-Signature, is the structure available for the XML standard to support digital signatures. This structure, which supports digital signatures in an XML file, is called **<Signature>** and is made up of three main elements: **<SignedInfo>** **<SignatureValue>** **<KeyInfo>** The first element, **<SignedInfo>**, contains information on how the signature was generated:

- the algorithm used to obtain the canonical XML format
- the algorithm used to produce the signature
- the information about the signed resource.

The above information about the signed resource is specified within the **<Reference>** element and relates to the URI for the XML document, the algorithm used to calculate its digest and the value of the digest itself. The **<SignatureValue>** element contains the value of the digital signature obtained according to the algorithm indicated in the previous element. The last element, **<KeyInfo>**, contains information about the signatory's public key and is used to decrypt the XML digital signature when the document's authenticity and integrity are verified. It is important to remember that the standard allows for different types of XML Signature:

- Enveloping (<http://www.w3.org/TR/xmldsig-core/#def-SignatureEnveloping>)
- Enveloped (<http://www.w3.org/TR/xmldsig-core/#def-SignatureEnveloped>)
- Detached (<http://www.w3.org/TR/xmldsig-core/#def-SignatureDetached>)

Enveloping Signature is the term used when the signature, i.e. the **<Signature>** element, contains the signed object. **Enveloped Signature**, on the other hand, is the term used when the signed object contains the **<Signature>** element. Lastly, a **Detached Signature** is when the **<Signature>** element is not a "parent" or a "descendant" of the signed object. This last description can be broken down further into:

- **Detached Internal Signature** if the **<Signature>** element is contained within the same XML document that contains the signed object and both (signature and signed object) are so-called "sibling" elements
- **Detached External Signature** if the **<Signature>** element is contained within a different XML document from the one that contains the signed object.

Digital signatures

The method to be used for generating a digital signature for an individual file in XAdES format is `xmlsignature()`.

The method is defined as:

```
public SignReturnV2 xmlsignature(SignRequestV2 request, XmlSignatureParameter parameter)
```

INPUT

- **request** - SignRequestV2 object containing the following parameters for the signature request
 - **Transport** = Enum that indicates the type of input (from here on transport type)
 - BYNARYNET If the array of bytes containing the file to sign is being added to the input
 - FILENAME If the path of the file to be signed is being added to the input
 - DIRECTORYNAME If the path of the folder with the files to be signed is being added to the input
 - STREAM If the stream for a file in MTOM format with JAX-WS java technology (for large files) is being added to the input
 - **Binaryinput** = Byte array containing the document to be signed. Only used when Transport = BINARYNET
N.B. = The input file must be a well-formed xml file
 - **SrcName** = String containing the path of the file or folder containing the files to be signed. Only used when Transport = FILENAME or DIRECTORYNAME
N.B. = The input file must be a well-formed xml file
N.B.: The resource indicated must be accessible from the ARSS server
 - **Stream** = DataHandler to upload large files in Stream format Only used when Transport = STREAMN.B.: The solution is only available with JAX-WS compliant clients.
 - **DstName** = String containing the destination path of the file or folder for the signed files. Only used when Transport = FILENAME or DIRECTORYNAME
N.B.: The path specified must be accessible from the ARSS server
 - **requiredmark** = RFU, do not use
 - **signingTime** = String used to set the `<xsd:SigningTime>` element to be inserted in the XADES envelope.
At the moment the only possible value for this parameter is NULL, to tell the ARSS server that the element must be populated by the insertion of the date on the system of the Server being used.
Optionally the signing time can include the timezone (eg. dd/MM/yyyy HH:mm:ss zz, 05/10/2023 14:59:00 CEST)
 - **Session_id** = String used to specify the session ID within multiple signature transactions. For a signature for an individual file or a folder, do not specify or set as NULL.
 - **Identity** = Auth object containing the data to authenticate the signatory
 - **TypeHSM** = String containing the type of HSM. Always applied with the 'COSIGN' string
 - **TypeOtpAuth** = String used to indicate the authentication domainsignature If a remote signature user is being used. If a remote signature user is being used with a customized domain, you need to specify this domain (e.g. frXXXX). If an automatic signature user is being used, you need to specify the string shown when installing ARSS. **User** = String containing the signature user's username.
 - **UserPWD** = String containing the signature user's password.

- **OtpPWD** = String containing the user's valid OTP code for the signature transaction.
- **ext_auth_type** = Enum that allows you to use the extended authentication method. (**Deprecated** see *authMethods* method instead)
- **ext_auth_blobvalue** = Array of bytes in which to add the authentication credential (encrypted challenge, etc.) for the *extended authentication methods* that allow for the exchange of binary data
- **ext_auth_value** = String in which to add the authentication credential (OTP or similar) for the *extended authentication methods* that allow for the exchange of data in plain text
- **delegated_user** = String that contains the username for the delegated user if the automatic signature has been delegated. In this case, the password for the Owner of the Signature key does not need to be specified
- **delegated_password** = String that contains the delegated user's password
- **delegated_domain** = Delegated user's domain (used in a similar way to the TypeOtpAuth parameter)
- **tsa_identity** = Optional TSAAuth object, containing the data for TSS Server authentication. If not specified, the server will use the default account, if configured.
- **user** = String containing the timestamp user's username
- **password** = String containing the timestamp user's password
- **Notifyemail** = String used to specify the email address to which notification of the completion of the signature operation is sent.
- **Notify_id** = String containing the ID that the user intends to link to the signature session for the purposes of notification of the completion of the operation. Only used when Transport = DIRECTORYNAMEFor a signature for an individual file, this property can be omitted or set as NULL.
- **CertID** = RFU. Set with ASO
- **profile** = RFU. Set to NULL
- **parameter** – XmlSignatureParameter object with the following parameters:
- **Type** = Enum used to indicate the type of XML signature to generate:
 - XMLENVELOPED If you intend to generate an **Enveloped** signature
 - XMLENVELOPING If you intend to generate an **Enveloping** signature
 - XMLDETACHED_INTERNAL If you intend to generate a **Detached Internal** signature
- **signatureProfile** = Enum used to indicate the profile of XML signature to generate:
 - ETSI_EN_319_132_1_v1_1_1 If you intend to generate a signature with EN3191321v1.1.1 profile
 - ETSI_TS_103171_v2_1_1 If you intend to generate a signature with TS103171v2.1.1 profile
 - XMLDETACHED_INTERNAL If you intend to generate a **Detached Internal** signature
- **Transforms** = Array of Transform objects that indicates the ordered list of transformations to apply to the document before calculating the hash. Each Transform element is made up of the following fields:
- **Type** = Enum indicating the type of transformation
 - CANONICAL_WITH_COMMENT
 - CANONICAL_OMIT_COMMENT
 - BASE64
 - XPATH2_INTERSECT
 - XPATH2_SUBTRACT
 - XPATH2_UNION
 - XSLT
- **Value** = Any value to be inserted, depending on the type of transformation specified in TypeFor example, Xpath expressions, etc.
- **canonicalizedType** = Enum indicating the type of canonicalization. Possible values:
 - ALGO_ID_C14N11_OMIT_COMMENTS
 - ALGO_ID_C14N11_WITH_COMMENTS
- **signatureLevel** – Signature level required. (Permitted Values: LT)

OUTPUT

SignReturnV2 object containing the following parameters:

- **Status** = String containing the outcome of the signature process
 - OK Operation completed successfully
 - Different from OK In case of error. Further details are included within the SignReturnV2.description and SignReturnV2.return_code properties.
- **Binaryoutput** = Byte array containing the signed file
- **Stream** = DataHandler to download large files in Stream formatOnly used when Transport = STREAM
N.B.: The solution is only available with JAX-WS compliant clients.
- **DstPath** = String containing the path specified within the SignRequestV2.DstName parameter during the request phase. If the file specified in SignRequestV2.DstName already exists, the SignReturnV2.DstPath property contains the path to the file actually generated by ARSS.
- **Return_code** = Enum containing the error code
 - "0001" Generic error in the signature process
 - "0002" Incorrect parameters for the type of transport indicated
 - "0003" Error during the credential verification phase
 - "0004" PIN error (more details in the SignReturnV2.description)
 - "0005" Invalid type of transport
 - "0006" Unauthorized type of transport
 - "0008" Impossible to complete the timestamping operation (e.g. impossible to connect to the service, remaining stamps ended etc.)
 - "0009" Invalid delegate credentials
 - "0010" Invalid user status (e.g. user suspended)
- **Description** = String containing a high level description of the error encountered and/or any details relating to the codes returned within the SignReturnV2.return_code property

Multiple Signatures

To generate a digital signature for a group of files in XML format, the same considerations apply as those outlined in the paragraph entitled *Multiple Signatures* in the chapter entitled **Remote Digital Signatures in pkcs1 Digital Signature** of a fingerprint in accordance with the pkcs1 standardIf you

intend to sign the fingerprint of a document (or a generic sequence of bytes) based on the pkcs1 standard, you simply need to use the `signhash()` method. This method is defined as follows:

```
public SignHashReturn signhash(SignHashRequest request)
```

This method is normally used for the following:

- Generating "detached" digital signatures, i.e. separate from the signed file

INPUT

- [request](#) - SignHashRequest containing the following parameters for the signature request
- [Identity](#) = Auth object containing the data to authenticate the signatory
- [TypeHSM](#) = String containing the type of HSM. Always applied with the 'COSIGN' string
- [TypeOtpAuth](#) = String used to indicate the authentication domain signature. If a remote signature user is being used. If a remote signature user is being used with a customized domain, you need to specify this domain (e.g. frXXXX). If an automatic signature user is being used, you need to specify the string shown when installing ARSS.
- [User](#) = String containing the signature user's username.
- [UserPWD](#) = String containing the signature user's password.
- [OtpPWD](#) = String containing the user's valid OTP code for the signature transaction.
- [ext_authtype](#) = Enum that allows you to use the extended authentication method. (**Deprecated** see [authMethods](#) method instead)
- [ext_auth_blobvalue](#) = Array of bytes in which to add the authentication credential (encrypted challenge, etc.) for the *extended authentication methods* that allow for the exchange of binary data
- [ext_auth_value](#) = String in which to add the authentication credential (OTP or similar) for the *extended authentication methods* that allow for the exchange of data in plain text
- [delegated_user](#) = String that contains the username for the delegated user if the automatic signature has been delegated. In this case, the password for the Owner of the Signature key does not need to be specified
- [delegated_password](#) = String that contains the delegated user's password
- [delegated_domain](#) = Delegated user's domain (used in a similar way to the TypeOtpAuth parameter)
- [certID](#) = Unique certificate ID to be used for the signature. The list of IDs can be retrieved using the [listCert](#) function explained below. If a specific certificate is not needed, you can specify AS0 to balance the requests, although this method assumes that the certificate itself does not contribute to the calculation of the HASH (not valid for CAdES signatures).
- [hash](#) = Array of bytes containing the fingerprint for the document to be signed.
- [hashtype](#) = String containing the name of the hash algorithm used to calculate the fingerprint (currently only "SHA256" available).
- [session_id](#) = String used to specify the session ID within multiple signature transactions. For a signature for an individual file or a folder, do not specify or set as NULL.
- [requirecert](#) = Boolean value that indicates the requirement for the presence of the signatory's certificate in the response.

OUTPUT

SignHashReturn object containing the following parameters:

- [Status](#) = String containing the outcome of the signature process
 - OK Operation completed successfully
 - Different from OK In case of error. Further details are included within the SignHashReturn.Description.
- [Description](#) = String containing a high level description of the error encountered and/or any details relating to the codes returned within the SignReturnV2.return_code property.
- [Return_code](#) = Enum containing the error code
 - "0001" Generic error in the signature process
 - "0002" Incorrect parameters for the type of transport indicated
 - "0003" Error during the credential verification phase
 - "0004" PIN error (more details in the SignReturnV2.description)
 - "0005" Invalid type of transport
 - "0006" Unauthorized type of transport
 - "0007" Invalid PDF signature profile
 - "0008" Impossible to complete the timestamping operation (e.g. impossible to connect to the service, remaining stamps ended etc.)
 - "0009" Invalid delegate credentials
 - "0010" Invalid user status (e.g. user suspended)
- [signature](#) = Array of bytes containing the outcome of the signature process.
- [cert](#) = Array of bytes containing the recipient's certificate if requested see [requirecert](#) parameter.
- [certID](#) = ID for the signatory's certificate. Remote Digital Signatures in the CAdES (p7m) standard. The only differences compared to the description for CAdES signatures are as follows:
 1. The method to be used for launching the signature function must be [xmldsignature\(\)](#) and not [pkcs7signV2\(\)](#)
 2. When the transport type is DIRECTORYNAME only XML files with features that fit in with the parameters specified within the [parameter](#) object are processed.
 3. The files that are not processed will be summarized in the notification email that will be sent at the end of the process to the address specified in [Notifymail](#)

Automatic Signature

To create an automatic signature in XML format, simply use the `xmldsignature()` method, adding the authentication credentials of an automatic signature user to the input.

INPUT

The details for launching the pdfsignatureV2() method follow the details given in the relevant section of the paragraph entitled *Digital Signatures* with the following changes, according to the context, for the automatic signatures process:

1. Set the SignRequestV2.Identity.TypeOtpAuth property with the string identifying the automatic signature domain.

N.B.: This string is indicated when ARSS is installed

1. Set the SignRequestV2.Identity.OtpPWD property with the string identifying the automatic signature transactions

N.B.: This static string is defined when the ARSS server is installed, and is normally known by the administrator of the IT infrastructure network on which users are working.

OUTPUT

The output returned by the function call follows the details given in the corresponding section of the paragraph entitled *Digital Signatures* in this Chapter

Remote Digital Signatures in the JAdES

Digital signatures (SOAP)

INPUT

jwsSignature

- **request** - SignRequestV2 object containing the following parameters for the signature request
- **BinaryInput** = Byte Array that contains the document to signature OR the hash for hash signature (only for Transport = BINARYNET)
- **CertId** = Certificate Id (for future use). For now set to 'AS0'
- **Identity** = Object that contains Signer Credentials for authentication.
- **TypeOtpAuth** = String used to indicate the authentication domainsignature
- **User** = String containing the signature user's username
- **UserPwd** = String containing the signature user's password.
- **OtpPwd** = String containing the user's valid OTP code for the signature transaction.
- **Transport**= Enum Class that indicates the type of input : BYNARYNET, FILENAME, DIRECTORYNAME, STREAM
- **Parameter** :
 - **Jwt**
 - **Audience**
 - **JwtSeriliatizationType** : Enum Class that indicates the type jws serialization : COMPACT, JSON_FLATTENED, JSON

OUTPUT

jwsSignatureResponse

- **Status** = String containing the outcome of the signature process
 - OK Operation completed successfully
 - Different from OK In case of error.
- **Binaryoutput** = Byte array containing the signed file
- **Return_code** = Enum containing the error code
 - "0003" Invalid User Credentials
 - "0004" Otp Credential Invalid Or Blocked
- **Description** = String containing a high level description of the error.

Multiple Signatures (SOAP)

INPUT

jwsSignature_multiple

- **request** - SignRequestV2 object containing the following parameters for the signature request
- **BinaryInput** = Byte Array that contains the document to signature OR the hash for hash signature (only for Transport = BINARYNET)
- **CertId** = Certificate Id (for future use). For now set to 'AS0'
- **Identity** = Object that contains Signer Credentials for authentication.
- **TypeOtpAuth** = String used to indicate the authentication domainsignature
- **User** = String containing the signature user's username
- **UserPwd** = String containing the signature user's password.
- **OtpPwd** = String containing the user's valid OTP code for the signature transaction.
- **Transport**= Enum Class that indicates the type of input : BYNARYNET, FILENAME, DIRECTORYNAME, STREAM
- **Parameter** :
 - **Jwt**
 - **Audience**

- **JwtSeriliatizationType** : Enum Class that indicates the type jws serialization : COMPACT, JSON_FLATTENED, JSON

OUTPUT

jwsSignatureResponse

- **Status** = String containing the outcome of the signature process
 - OK Operation completed successfully
 - Different from OK In case of error.
- **Binaryoutput** = Byte array containing the signed file
- **Return_code** = Enum containing the error code
 - "0003" Invalid User Credentials
 - "0004" Otp Credential Invalid Or Blocked
- **Description** = String containing a high level description of the error.

Digital signatures (REST)

INPUT

- **request** - SignRequestV2 object containing the following parameters for the signature request
- **BinaryInput** = Byte Array that contains the document to signature OR the hash for hash signature (only for Transport = BINARYNET)
- **CertId** = Certificate Id (for future use). For now set to 'AS0'
- **Identity** = Object that contains Signer Credentials for authentication.
- **TypeOtpAuth** = String used to indicate the authentication domainsignature
- **User** = String containing the signature user's username
- **UserPwd** = String containing the signature user's password.
- **TypeHsm** = = "COSIGN"
- **OtpPwd** = String containing the user's valid OTP code for the signature transaction.
- **Transport**= Enum Class that indicates the type of input : BYNARYNET, FILENAME, DIRECTORYNAME, STREAM
- **Parameter** :
 - **Jwt**
 - **Audience**
 - **JwtSeriliatizationType** : Enum Class that indicates the type jws serialization : COMPACT, JSON_FLATTENED, JSON

OUTPUT

- **Status** = String containing the outcome of the signature process
- **Binaryoutput** = Byte array containing the signed file
- **Return_code** = Enum containing the error code
- **Description** = String containing a high level description of the error.
- **Stream** = DataHandler to download large files in Stream formatOnly used when Transport = STREAM
- **DstPath** = String containing the path specified within the SignRequestV2.DstName parameter during the request phase.

Multiple Signatures (REST)

INPUT

- **request** - SignRequestV2 object containing the following parameters for the signature request
- **requestList** = contains the BinaryInput and transport for all signatories
- **BinaryInput** = Byte Array that contains the document to signature OR the hash for hash signature (only for Transport = BINARYNET)
- **Identity** = Object that contains Signer Credentials for authentication.
- **TypeOtpAuth** = String used to indicate the authentication domainsignature
- **User** = String containing the signature user's username
- **UserPwd** = String containing the signature user's password.
- **TypeHsm** = = "COSIGN"
- **OtpPwd** = String containing the user's valid OTP code for the signature transaction.
- **Transport**= Enum Class that indicates the type of input : BYNARYNET, FILENAME, DIRECTORYNAME, STREAM

OUTPUT

- **Status** = String containing the outcome of the signature process
- **Binaryoutput** = Byte array containing the signed file
- **Return_code** = Enum containing the error code
- **Description** = String containing a high level description of the error.
- **Stream** = DataHandler to download large files in Stream formatOnly used when Transport = STREAM
- **DstPath** = String containing the path specified within the SignRequestV2.DstName parameter during the request phase.

Timestamping

The ARSS server allows you to access digital timestamping services that comply with the rfc 3161 standard. In particular, with ARSS you can request timestamping for any data.

Brief recap on timestamping

A digital timestamping (DTS) service plays an important role in making sure that transactions are not rejected. In particular, the purpose of a DTS service is to demonstrate that a particular piece of data, D (message, buffer, document, etc.) existed no later than a certain point, T, in time. This mechanism can be used, for example, to verify that a digital signature was generated before the corresponding certificate was revoked; in this case, the revoked certificate can nevertheless be used to verify digital signatures created before the date/time that the certificate was revoked. This is an important process in the area of PKIs (public key infrastructure). Another important use of timestamping lies in applications involved in "filing" documents (e.g. balance sheets, patents, tax returns, etc.) electronically, when the moment at which they are sent or received is of critical importance. The basic mechanism behind digital timestamping is very simple. Essentially, the party making the request (client) calculates the hash, H, of the original piece of data, D, then sends the H value to a timestamping server (managed by a trusted third party that provides this service, known as a Timestamping Authority or TSA). The server connects the H value to the current time, T (date and time), obtained from a reliable source, and then signs everything with its own private key. The result, called a timestamp token, is sent back to the client. The figure below illustrates the simple interaction on which a timestamping service is based: !worddav1759059da9a20667f5c3a73557ac9671.png!height=152,width=468!The Rfc 3161 standard defines the communication protocol between client and server, in other words the data structures exchanged and the transport methods. The timestamping support offered by ARSS is in fact based on this standard, and allows simple integration by "masking" all the low-level details of the Rfc 3161 protocol for the timestamping request via individual SOAP calls. In order to generate timestamps successfully via ARSS, you will first need to have activated a timestamp user and have the associated login credentials to hand.

Timestamp in TSR (.tsr) format

The method to use for generating a timestamp in tsr format not connected to the timestamped file is `tsr()`. The method is defined as:

```
public MarkReturn tsr (MarkRequest request)
```

INPUT

- **request** - MarkRequest object containing the following parameters for the timestamp request
 - **Transport** = Enum that indicates the type of input (from here on transport type)
 - BYNARYNET If the array of bytes containing the file to sign is being added to the input
 - FILENAME If the path of the file to be timestamped is being added to the input
 - DIRECTORYNAME If the path of the folder with the files to be timestamped is being added to the input
 - STREAM If the stream for a file in MTOM format with JAX-WS java technology (for large files) is being added to the input
 - **Binaryinput** = Byte array containing the document to be timestamped. Only used when Transport = BINARYNET
 - **SrcName** = String containing the path of the file or folder containing the files to be timestamped. Only used when Transport = FILENAME or DIRECTORYNAME N.B.: The resource indicated must be accessible from the ARSS server
 - **Stream** = DataHandler to upload large files in Stream format Only used when Transport = STREAM N.B.: The solution is only available with JAX-WS compliant clients.
 - **DstName** = String containing the destination path of the file or folder for the timestamped files. Only used when Transport = FILENAME or DIRECTORYNAME N.B.: The path specified must be accessible from the ARSS server
 - **Notifyemail** = String used to specify the email address to which notification of completion of the timestamping operation is sent.
 - **Notify_id** = String containing the ID that the user intends to link to the timestamp session for the purposes of notification of completion of the operation. Only used when Transport = DIRECTORYNAME For a timestamp for an individual file, this property can be omitted or set as NULL.
 - **User** = String containing the timestamp user's username
 - **Password** = String containing the timestamp user's password
 - **Url** = Url for the timestamp service

OUTPUT

MarkReturn object containing the following parameters:

- **Status** = String containing the outcome of the timestamping operation OK Operation completed successfully Different from OK In case of error. Further details are included within the MarkReturn.description and MarkReturn.return_code properties.
- **Binaryoutput** = Byte array containing the timestamp in tsr format
- **Stream** = DataHandler to download large files in Stream format Only used when Transport = STREAM N.B.: The solution is only available with JAX-WS compliant clients.
- **DstPath** = String containing the path specified within the MarkRequest.DstName parameter during the request phase. If the file specified in MarkRequest.DstName already exists, the MarkReturn.DstPath property will point to the file actually generated by ARSS.
- **Return_code** = Enum containing the error code
 - "0001" Generic error in the timestamp process
 - "0002" Incorrect parameters for the type of transport indicated
 - "0003" Error during the credential verification phase
 - "0005" Invalid type of transport
 - "0006" Unauthorized type of transport
 - "0007" Invalid PDF signature profile
 - "0008" Impossible to complete the timestamping operation (e.g. impossible to connect to the service, remaining stamps ended etc.)
 - "0009" Invalid delegate credentials
 - "0010" Invalid user status (e.g. user suspended)
- **Description** = String containing a high level description of the error encountered and/or any details relating to the codes returned within the MarkReturn.return_code property

Timestamp in TSD (.tsd) format

The method to be used to generate a timestamp connected to the timestamped file in tsd format is `tsd()`. With this method, ARSS allows you to comprehensively manage "timestamped envelopes" in accordance with the published specification RFC 5544 ("TimeStampedData"). The TimeStampedData format allows you to "bind" any file (not necessarily signed) with a timestamp. This means you can avoid using proprietary formats that are not interoperable. The method is defined as:

```
public MarkReturn tsd (MarkRequest request)
```

INPUT

- **request** - MarkRequest object containing the following parameters for the timestamp request
 - **Transport** = Enum indicating the type of transport
 - BYNARYNET If the array of bytes containing the file to sign is being added to the input
 - FILENAME If the path of the file to be timestamped is being added to the input
 - DIRECTORYNAME If the path of the folder with the files to be timestamped is being added to the input
 - **Binaryinput** = Byte array containing the document to be timestamped. Only used when Transport = BINARYNET
 - **SrcName** = String containing the path of the file or folder containing the files to be timestamped. Only used when Transport = FILENAME or DIRECTORYNAME
N.B.: The resource indicated must be accessible from the ARSS server
 - **DstName** = String containing the destination path of the file or folder for the timestamped files. Only used when Transport = FILENAME or DIRECTORYNAME
N.B.: The path specified must be accessible from the ARSS server
 - **Stream** = DataHandler to upload large files in Stream format Only used when Transport = STREAM
N.B.: The solution is only available with JAX-WS compliant clients.
 - **Notifyemail** = String used to specify the email address to which notification of completion of the timestamping operation is sent.
 - **Notify_id** = String containing the ID that the user intends to link to the timestamp session for the purposes of notification of completion of the operation. Only used when Transport = DIRECTORYNAME for a timestamp for an individual file, this property can be omitted or set as NULL.
 - **User** = String containing the timestamp user's username
 - **Password** = String containing the timestamp user's password
 - **Url** = Url for the timestamp service

OUTPUT

MarkReturn object containing the following parameters:

- **Status** = String containing the outcome of the timestamping operation
 - OK Operation completed successfully
 - Different from OK In case of error. Further details are included within the MarkReturn.description and MarkReturn.return_code properties.
- **Binaryoutput** = Byte array containing the timestamp in tsd format
- **Stream** = DataHandler to download large files in Stream format Only used when Transport = STREAM
N.B.: The solution is only available with JAX-WS compliant clients.
- **DstPath** = String containing the path specified within the MarkRequest.DstName parameter during the request phase. If the file specified in MarkRequest.DstName already exists, the MarkReturn.DstPath property will point to the file actually generated by ARSS.
- **Return_code** = Enum containing the error code
 - "0001" Generic error in the timestamp process
 - "0002" Incorrect parameters for the type of transport indicated
 - "0003" Error during the credential verification phase
 - "0005" Invalid type of transport
 - "0006" Unauthorized type of transport
- **Description** = String containing a high level description of the error encountered and/or any details relating to the codes returned within the MarkReturn.return_code property

Timestamp in M7M (.m7m) format

METHOD REMOVED

Verifying digital signatures

From Version ARSS 1.13.1 the methods used for verification have been removed. The benchmark product for verification operations is now VOL. Contact your sales representative for more details.

Additional functions

List of certificates associated with a user (via user credentials)

The method to be used to get the list of certificates associated with a user via their credentials is `listCert()`. The method is defined as this.

```
public UserCertList listCert(Auth identity)
```

INPUT

- **identity** - Auth object containing the data for authenticating the user
- **TypeHSM** = String containing the type of HSM. Always applied with the 'COSIGN' string
- **TypeOtpAuth** = String used to indicate the authentication domain signature. If a remote signature user is being used. If a remote signature user is being used with a customized domain, you need to specify this domain (e.g. frXXXX). If an automatic signature user is being used, you need to specify the string shown when installing ARSS.
- **User** = String containing the signature user's username.
- **UserPWD** = String containing the signature user's password.
- **OtpPWD** = NULL

OUTPUT

UserCertList object containing the following parameters:

- **status** = String containing the outcome of the search carried out
 - OK Search carried out successfully
 - Different from OK In case of error. Further details are included within the userCertList.description property.
- **Description** = String containing a high level description of the error encountered during the search for certificates associated with the user
- **App1** = List object <Cert> containing the base64 for certificates configured on the first HSM
- **App2** = List object <Cert> containing the base64 for certificates configured on the second HSM

List of certificates associated with a user (via administrator's credentials)

The method to be used to get the list of certificates associated with a user via the administrator's credentials is listCertAuth(). The method is defined as this.

```
public UserCertListAuth listCertAuth(ApplicationAuth identity, String domain, String user)
```

INPUT

- **identity** - ApplicationAuth object containing the data to authenticate the administrator
- **applicationuser** = Administrator username
- **applicationpassword** = Administrator password
- **domain** - Domain to which the user for whom you want to search certificates belongs
- **user** - Username for the user for whom you want to search certificates

OUTPUT

UserCertListAuth object containing the following parameters:

- **status** = String containing the outcome of the search carried out
 - OK Search carried out successfully
 - Different from OK In case of error. Further details are included within the userCertList.description property.
- **Description** = String containing a high level description of the error encountered during the search for certificates associated with the user
- **certs** = List<Cert> object containing the base64 for certificates configured on the first HSM

List of methods to authenticate a user

The method to be used for generating the list of methods to authenticate a user is authMethods(). The method is defined as this.

```
public AuthMethodsReturn authMethods(Auth identity)
```

INPUT

- **identity** - Auth object containing the data for authenticating the user
- **TypeHSM** = String containing the type of HSM. Always applied with the 'COSIGN' string
- **TypeOtpAuth** = String used to indicate the authentication domain signature. If information for a remote signature user is being requested. If information for an automatic signature user is being requested instead. The string indicated when ARSS is installed must be specified.
- **User** = String containing the signature user's username.
- **UserPWD** = String containing the signature user's password.
- **OtpPWD** = NULL

OUTPUT

AuthMethodsReturn object containing the following parameters:

- **status** = String containing the outcome of the search carried out
 - OK Search carried out successfully
 - Different from OK In case of error. Further details are included within the userCertList.description property.

- **Description** = String containing a high level description of the error encountered during the search for certificates associated with the user
- **return_code** = String containing the return code, set to "0000" if successful and another value if there has been an error.
- **methods** = List of strings that identify the methods to authenticate a given user
 - CNS2 Indicates authentication using the CNS card
 - ARUBACALL Indicates authentication using an OTP sent via the ARUBACALL service
 - SMS Indicates authentication using an OTP sent via the ARUBASMS service
 - OTP Indicates authentication using an OTP sent via the ARUBAOTP application

List of active signature processes

If the folder signature has been launched, the listprocess() method can be used to generate a list of the operations still underway. The method is defined as this.

```
public String[] listprocess(Auth identity)
```

INPUT

- **identity** - Auth object containing the data for authenticating the user
- **TypeHSM** = String containing the type of HSM. Always applied with the 'COSIGN' string
- **TypeOtpAuth** = String used to indicate the authentication domain signature. If a remote signature user is being used. If a remote signature user is being used with a customized domain, you need to specify this domain (e.g. frXXXX). If an automatic signature user is being used, you need to specify the string shown when installing ARSS.
- **User** = String containing the signature user's username.
- **UserPWD** = String containing the signature user's password.
- **OtpPWD** = NULL

OUTPUT

Array of strings containing the list of signature processes underway. Each element in the array contains the string specified in the SignRequestV2. Session_id field when the particular bulk folder signature process is launched. These processes, that have no ID associated with them in advance, will be mapped within the array with random values generated by the ARSS server.

Get ARSS version

The method to be used for finding out which version of the ARSS server is being used is getVersion(). The method is defined as this.

```
public String getVersion()
```

INPUT

The method has no parameters.

OUTPUT

String describing the ARSS version in use.

Utility for checking connection to ARSS

The method to be used for ensuring that the ARSS server is being reached and is working is ping(). The method is defined as this.

```
public String ping()
```

INPUT

The method has no parameters.

OUTPUT

'Pong' string if the ARSS server is accessible and working.

Utility to authenticate users with an OTP

The method to be used to authenticate a remote/automatic signature user is verifyOtp(). The method is defined as this.

```
public ArssReturn verifyOtp(Auth identity)
```

INPUT

- **identity** - Auth object containing the data for authenticating the user

- **TypeHSM** = String containing the type of HSM. Always applied with the 'COSIGN' string
- **TypeOtpAuth** = String used to indicate the authentication domain signature. If a remote signature user is being used. If a remote signature user is being used with a customized domain, you need to specify this domain (e.g. frXXXX). If an automatic signature user is being used, you need to specify the string shown when installing ARSS.
- **User** = String containing the signature user's username.
- **UserPWD** = String containing the signature user's password.
- **OtpPWD** = OTP to be verified.

OUTPUT

ArssReturn object containing the following parameters:

- **Status** = String containing the outcome of the timestamping operation
 - OK Operation completed successfully
 - Different from OK In case of error. Further details are included within the ArssReturn.description and ArssReturn.return_code properties.
- **Description** = String containing a high level description of the error encountered during authentication via OTP.
- **Return_code** = Enum containing the error code
 - "0001" Generic error in the authentication process
 - "0002" Incorrect parameters
 - "0003" Error during the credential verification phase
 - "0004" PIN error

Utility for sending one of the OTP signature credentials via SMS or ARUBACALL (caller ID)

The method to be used for sending the OTP to the user is sendCredential(). The method is defined as this.

```
public ArssReturn sendCredential(Auth identity, CredentialsType type)
```

INPUT

- **identity** - Auth object containing the data for authenticating the user
- **TypeHSM** = String containing the type of HSM. Always applied with the 'COSIGN' string
- **TypeOtpAuth** = String used to indicate the authentication domain signature. If a remote signature user is being used. If a remote signature user is being used with a customized domain, you need to specify this domain (e.g. frXXXX). If an automatic signature user is being used, you need to specify the string shown when installing ARSS.
- **User** = String containing the signature user's username.
- **UserPWD** = String containing the signature user's password.
- **OtpPWD** = NULL
- **type** - CredentialsType object containing the data to identify the user authentication type. For all intents and purposes, this is an ENUM with two possible values
 - "ARUBACALL"
 - "SMS"
 - "PAPERTOKEN" **Deprecated, DO NOT use**

OUTPUT

ArssReturn object containing the following parameters:

- **Status** = String containing the outcome of the timestamping operation
 - OK Operation completed successfully
 - Different from OK In case of error. Further details are included within the ArssReturn.description and ArssReturn.return_code properties.
- **Description** = String containing a high level description of the error encountered when sending the OTP credential
- **Return_code** = Enum containing the error code
 - "0002" Incorrect parameters
 - "0003" Error during the credential verification phase

Utility to send one of the OTP signature credentials via SMS or ARUBACALL (caller ID) with Credential Proxy active

The method to be used to send the credentials to the user when the Credential Proxy is active is sendCredentialAuth(). The method is defined as this.

```
public ArssReturn sendCredentialAuth(ApplicationAuth identity, CredentialsType type, String domain, String user)
```

If the proxy credential is active (globally, on the user and domain) the entire identity of the user is not required but only his/her username and domain: the password will be obtained internally from this data.

INPUT

- **identity** = ApplicationAuth object containing the application credentials
 - **applicationuser** = application username
 - **applicationpassword** = application password

- **type** = CredentialsType object containing the data to identify the user authentication type. For all intents and purposes, this is an ENUM with two possible values "ARUBACALL" "SMS""PAPERTOKEN" - **Deprecated, DO NOT use**
- **domain** = domain of the user to which the credential is to be sent
- **username** = username of the user to which the credential is to be sent

OUTPUT

ArssReturn object containing the following parameters:

- **Status** = String containing the outcome of the timestamping operation OK Operation completed successfully Different from OK In case of error. Further details are included within the ArssReturn.description and ArssReturn.return_code properties.
- **Description** = String containing a high level description of the error encountered when sending the OTP credential
- **Return_code** = Enum containing the error code "0002" Incorrect parameters "0003" Error during the credential verification phase "1000" Credentials proxy not globally enabled "1001" Credentials not present "1002" Credential Proxy disabled at domain level "1003" Credential Proxy deactivated at domain level "1004" Credential Proxy not enabled at user level

Utility to get a credential (OTP PAPERTOKEN)

WARNING: The PAPERTOKEN authentication method has been deprecated! DO NOT use this method! The method to be used for retrieving additional OTP information is retrieveCredential() The method is defined as this.

```
public ArssReturn retrieveCredential(Auth identity, CredentialsType type)
```

INPUT

- **identity** - Auth object containing the data for authenticating the user
- **TypeHSM** = String containing the type of HSM. Always applied with the 'COSIGN' string
- **TypeOtpAuth** = String used to indicate the authentication domain signature If a remote signature user is being used. If a remote signature user is being used with a customized domain, you need to specify this domain (e.g. frXXXX). If an automatic signature user is being used, you need to specify the string shown when installing ARSS.
- **User** = String containing the signature user's username.
- **UserPWD** = String containing the signature user's password.
- **OtpPWD** = NULL
- **type** - CredentialsType object containing the data to identify the type of authentication the user would like to use to recover additional information about the OTP. For now, only PAPERTOKEN is supported.

OUTPUT

RetrieveCredentialReturn object containing the following parameters:

- **Status** = String containing the outcome of the timestamping operation
 - OK Operation completed successfully
 - Different from OK In case of error. Further details are included within the ArssReturn.description and ArssReturn.return_code properties.
- **Description** = String containing a high level description of the error encountered when sending the OTP credential
- **Return_code** = Enum containing the error code
 - "0002" Incorrect parameters
 - "0003" Error during the credential verification phase
- **blob** = If allowed by the authentication method it contains the byte array needed to recover the credential
- **textvalue** = If allowed by the authentication method it contains the text needed to recover the credential. For example, for PAPERTOKEN it contains the list of coordinates.

File encryption

The method to be used to encrypt files.

```
public EncryptedEnvelopReturn encryptedEnvelope(EncryptedEnvelopReq request)
```

INPUT

- **request** - EncryptedEnvelopReq object
- **user** = user of the encryption service (must be statically configured in the ARSS options)
- **password** = password for the encryption service (must be statically configured in the ARSS options)
- **Transport** = Enum indicating the type of transport
 - BINARYNET If the array of bytes containing the file to sign is being added to the input
 - FILENAME If the path of the file to be timestamped is being added to the input
 - DIRECTORYNAME If the path of the folder with the files to be timestamped is being added to the input
- **Binaryinput** = Byte array containing the document to be encrypted. Only used when Transport = BINARYNET
- **SrcName** = String containing the path of the file or the folder with the files to be encrypted. Only used when Transport = FILENAME or DIRECTORYNAME.N.B.: The input file must be a signed file.N.B.: The resource indicated must be accessible from the ARSS server
- **DstName** = String containing the destination path of the file or the folder of encrypted files. Only used when Transport = FILENAME or DIRECTORYNAME.N.B.: The path specified must be accessible from the ARSS server
- **Notifyemail** = String used to specify the email address to which notification of completion of the timestamping operation is sent.

- **Notify_id** = String containing the ID that the user intends to link to the timestamp session for the purposes of notification of completion of the operation. Only used when Transport = DIRECTORYNAME. For a timestamp for an individual file, this property can be omitted or set as NULL.
- **Recipients** = List of array of bytes containing the certificates to be used for encryption.
- **Algorithm** = Enum indicating the type of algorithm to be used for encryption
N.B.: Not all algorithms can be authorized depending on the JDK configuration.
 - DES_EDE3_CBC - Triple DES
 - CBCRC2_CBC - RC2
 - CBCIDEA_CBC - IDEA
 - CBCCAST5_CBC - CAST5
 - CBCAES128_CBC - AES 128
 - CBCAES192_CBC - AES 192
 - CBCAES256_CBC - AES 256
 - CBCCAMELLIA128_CBC - CAMELLIA 128
 - CBCCAMELLIA192_CBC - CAMELLIA 192
 - CBCCAMELLIA256_CBC - CAMELLIA 256 CBC

OUTPUT

EncryptedEnvelopReturn object containing the following parameters:

- **Status** = String containing the outcome of the timestamping operation
 - OK Operation completed successfully
 - Different from OK In case of error. Further details are included within the ArssReturn.description and ArssReturn.return_code properties.
- **Description** = String containing a high level description of the error encountered when sending the OTP credential
- **Return_code** = Enum containing the error code
 - "0002" Incorrect parameters
 - "0011" Invalid certificate format
 - "0012" Invalid key usage for the certificate
 - "0013" Invalid algorithm.
- **binaryoutput** = If allowed by the transport method it contains the encrypted byte array in CMS format.
- **dstPath** = If allowed by the transport method it contains the path for the encrypted file.
- **stream** = If allowed by the transport method it contains the stream containing the encrypted file in CMS format.

WS-SECURITY soap header signature

The method to be used is *soapsignature*.

```
public SignReturnV2 soapsignature(SignRequestV2 request)
```

INPUT

- **request** - SignRequestV2 object containing the following parameters for the signature request
 - **Transport** = Enum that indicates the type of input (from here on transport type)
 - BYNARYNET If the array of bytes containing the file to sign is being added to the input
 - FILENAME If the path of the file to be signed is being added to the input
 - DIRECTORYNAME If the path of the folder with the files to be signed is being added to the input
 - STREAM If the stream for a file in MTOM format with JAX-WS java technology (for large files) is being added to the input
 - **Binaryinput** = Byte array containing the document to be signed. Only used when Transport = BINARYNET. (N.B.: The input file must be a well-formed xml file)
 - **SrcName** = String containing the path of the file or folder containing the files to be signed. Only used when Transport = FILENAME or DIRECTORYNAME. (N.B.: The input file must be a well-formed xml file, N.B.: The resource indicated must be accessible from the ARSS server)
 - **Stream** = DataHandler to upload large files in Stream format. Only used when Transport = STREAM. (N.B.: The solution is only available with JAX-WS compliant clients.)
 - **DstName** = String containing the destination path of the file or folder for the signed files. Only used when Transport = FILENAME or DIRECTORYNAME. (N.B.: The path specified must be accessible from the ARSS server)
 - **requiredMark** = RFU, do not use
 - **signingTime** = String used to set the <xsd:SigningTime> element to be inserted in the XADES envelope.
At the moment the only possible value for this parameter is NULL, to tell the ARSS server that the element must be populated by the insertion of the date on the system of the Server being used.
Optionally the signing time can include the timezone (eg. dd/MM/yyyy HH:mm:ss zz, 05/10/2023 14:59:00 CEST)
 - **Session_id** = String used to specify the session ID within multiple signature transactions. For a signature for an individual file or a folder, do not specify or set AS NULL.
 - **Identity** = Auth object containing the data to authenticate the signatory
 - **TypeHSM** = String containing the type of HSM. Always applied with the 'COSIGN' string
 - **TypeOtpAuth** = String used to indicate the authentication domain
 - Signature If a remote signature user is being used.
 - If a remote signature user is being used with a customised domain, it is necessary to specify this domain (e.g. frXXXX).
 - If an automatic signature user is being used, you need to specify the string shown when installing ARSS.
 - **User** = String containing the signature user's username.
 - **UserPWD** = String containing the signature user's password.
 - **OtpPWD** = String containing the user's valid OTP code for the signature transaction.

- **ext_authtype** = Enum that allows you to use the extended authentication method. (see *authMethods* method instead)
- **ext_auth_blobvalue** = Array of bytes in which to add the authentication credential (encrypted challenge, etc.) for the *extended authentication methods* that allow for the exchange of binary data
- **ext_auth_value** = String in which to add the authentication credential (OTP or similar) for the *extended authentication methods* that allow for the exchange of data in plain text
- **delegated_user** = String that contains the username for the delegated user if the automatic signature has been delegated. In this case, the password for the Owner of the Signature key does not need to be specified
- **delegated_password** = String that contains the delegated user's password
- **delegated_domain** = Delegated user's domain (used in a similar way to the TypeOtpAuth parameter)
- **tss_identity** = Optional TSAAuth object, containing the data for TSS Server authentication. If not specified, the server will use the default account, if configured.
- **user** = String containing the timestamp user's username
- **password** = String containing the timestamp user's password
- **notifyemail** = String used to specify the email address to which notification of the completion of the signature operation is sent.
- **notify_id** = String containing the ID that the user intends to link to the signature session for the purposes of notification of the completion of the operation. Only used when Transport = DIRECTORYNAME. For a signature for an individual file, this property can be omitted or set as NULL.
- **certID** = RFU. Set with AS0
- **profile** = RFU. Set to NULL

OUTPUT

SignReturnV2 object containing the following parameters:

- **status** = String containing the outcome of the signature process
 - OK Operation completed successfully
 - Different from OK In case of error. Further details are included within the SignReturnV2.description and SignReturnV2.return_code properties.
- **binaryoutput** = Byte array containing the signed file
- **stream** = DataHandler to download large files in Stream format. Only used when Transport = STREAM. (N.B.: The solution is only available with JAX-WS compliant clients.)
- **dstPath** = String containing the path specified within the SignRequestV2.DstName parameter during the request phase. If the file specified in SignRequestV2.DstName already exists, the SignReturnV2.DstPath property contains the path to the file actually generated by ARSS.
- **return_code** = Enum containing the error code
 - "0001" Generic error in the signature process
 - "0002" Incorrect parameters for the type of transport indicated
 - "0003" Error during the credential verification phase
 - "0004" PIN error (more details in the SignReturnV2.description)
 - "0005" Invalid type of transport
 - "0006" Unauthorized type of transport
 - "0008" Impossible to complete the timestamping operation (e.g. impossible to connect to the service, remaining timestamps ended etc.)
 - "0009" Invalid delegate credentials
 - "0010" Invalid user status (e.g. user suspended)
- **description** = String containing a high level description of the error encountered and/or any details relating to the codes returned within the SignReturnV2.return_code property

Utility for retrieving users within a domain

The utility method to be used to retrieve users within a domain is

```
public CredentialListReturn credentials_query(CredentialListQuery credential_query)
```

This is an exact search

INPUT

- **credentials_query** - CredentialListQuery object containing the following parameters
 - **constraints** = QueryConstraint list containing the following parameters
 - **field** = String containing the name of the attribute searched (only value currently allowed CF)
 - **value** = String containing the value of the searched attribute
 - **appidentity** = ApplicationAuth object containing the application credentials
 - **domain** = String containing the domain

OUTPUT

CredentialListReturn object containing the following parameters:

- **credentials** = CredentialInfo list containing the following parameters
 - **userid** = String containing the user's ID
 - **certs** = Array of Certificate objects available to the user
 - **status** = String containing the user's status, one of:
 - **V** = valid
 - **S** = suspended
 - **C** = deleted

Change Password

Method to be used when changing the password, for remote signature users you must also specify an OTP.

```
public GWReturn changePassword( Auth identity, String newPassword )
```

INPUT

- **change_password** - change_password object containing the following parameters
 - **identity** = Auth object containing the data to authenticate the signatory
 - **newPassword** = New password

OUTPUT

GWReturn object containing the following parameters:

- Object containing the following return properties
 - **description** = Outcome Description.
 - **return_code** = Return code see table. 0000 positive outcome.
 - **status** = General status of the operation.

Update signature

Method to be user for upgrading a digital signature's level from BES/T to T/LT

```
public SignReturnV2 updateSignature(UpdateSignatureRequest request)
```

INPUT

- **request** - UpdateSignatureRequest object containing the following parameters for the upgrade request
 - **transport** - Enum that indicates the type of input (from here on transport type)
 - **BYNARYNET** - If the array of bytes containing the signed file is being added to the input
 - **FILENAME** - If the path of the signed file is being added to the input
 - **DIRECTORYNAME** - If the path of the folder with the signed files is being added to the input
 - **STREAM** - If the stream for a file in MTOM format with JAX-WS java technology (for large files) is being added to the input
 - **binaryinput** - Byte array containing the signed file. Only used when Transport = BINARYNET
 - **originalDataBinaryinput** - Byte array containing the original file. Only used if Binaryinput is a CADES detached signature. Only used when Transport = BINARYNET
 - **stream** - DataHandler to upload large files in Stream format. Only used when Transport = STREAM
 - N.B.: The solution is only available with JAX-WS compliant clients.**
 - N.B.: The resource indicated must be accessible from the ARSS server**
 - **originalDataStream** - DataHandler to upload large files in Stream format. Only used if Stream is a CADES detached signature. Only used when Transport = STREAM
 - N.B.: The solution is only available with JAX-WS compliant clients.**
 - N.B.: The resource indicated must be accessible from the ARSS server**
 - **srcName** - String containing the path of the file or folder containing the signed files. Only used when Transport = FILENAME or DIRECTORYNAME
 - **originalDataSrcName** - String containing the path of the original file. Only used if Stream is a CADES detached signature. Only used when Transport = FILENAME
 - **requestdstName** - String containing the destination path of the file or folder for the upgraded files. Only used when Transport = FILENAME or DIRECTORYNAME
 - N.B.: The path specified must be accessible from the ARSS server**
 - **signatureType** - Enum that indicates the signature type
 - **CADES**
 - **PADES**
 - **XADES**
 - **signatureLevel** - Enum that indicate the level of the signature after the upgrade
 - **T**
 - **LT**
 - **signaturePath** - String containing the path of the signature to be upgraded. Each signature is a zero-based index separated by ':'. Example 1:0 = first signature of under the second signature
 - N.B.** CADES and XAdES signatures admit countersignatures. For PAdES the signature path is a single zero based index
 - **returnder** - Boolean value used to generate a CADES envelope in DER FORMAT true = request for generation in DER format false = request in BER format (this is the default value)
 - **tsa_identity** - Optional TSAAuth object, containing the data for TSS Server authentication. If not specified, the server will use the default account, if configured.

OUTPUT

SignReturnV2 Object containing the following parameters:

- **Status** = String containing the outcome of the signature process
 - OK Operation completed successfully

- Different from OK In case of error. Further details are included within the SignReturnV2.description and SignReturnV2.return_code properties.
- **Binaryoutput** = Byte array containing the signed file
- **Stream** = DataHandler to download large files in Stream format. Only used when Transport = STREAM. (N.B.: The solution is only available with JAX-WS compliant clients.)
- **DstPath** = String containing the path specified within the SignRequestV2.DstName parameter during the request phase. If the file specified in SignRequestV2.DstName already exists, the SignReturnV2.DstPath property contains the path to the file actually generated by ARSS.
- **Return_code** = Enum containing the error code
 - "0001" Generic error in the signature process
 - "0002" Incorrect parameters for the type of transport indicated
 - "0003" Error during the credential verification phase
 - "0004" PIN error (more details in the SignReturnV2.description)
 - "0005" Invalid type of transport
 - "0006" Unauthorized type of transport
 - "0007" Invalid PDF signature profile
 - "0008" Impossible to complete the timestamping operation (e.g. impossible to connect to the service, remaining stamps ended etc.)
 - "0009" Invalid delegate credentials
 - "0010" Invalid user status (e.g. user suspended)
- **Description** = String containing a high level description of the error encountered and/or any details relating to the codes returned within the SignReturnV2.return_code property

Appendices

- **ARSS_javadoc.zip:**

ARSS javadoc

- **ArubaSignService.xml**

WSDL describing the format for SOAP messages used by ARSS

- **ArubaSignService.xsd**

Diagram describing the objects used by the protocol implemented by ARSS.

Return Codes

Below is a full list of ARSS return codes.

- "0001" Generic error
- "0002" Invalid parameter. %s || **NB:** (%s is a variable parameter)
- "0003" Error during the credential verification phase
- "0004" PIN error (more details in the SignReturnV2.description)
- "0005" Invalid type of transport
- "0006" Unauthorized type of transport
- "0007" Invalid PDF Profile
- "0008" Unable To required Mark: %s || **NB:** (%s is a variable parameter)
- "0009" Invalid delegate credentials
- "0010" Invalid Certificate Status
- "0011" Invalid certificate format
- "0012" Invalid key usage for the certificate
- "0013" Invalid algorithm
- "0014" Invalid session
- "0015" Error occurred: %s || **NB:** (%s is a variable parameter)
- "0016" Operation not supportd yet
- "0019" Signature Timeout Reached
- "0020" Max request Exceeded
- "0021" Invalid license
- "0022" Operational Status Blocked
- "0023" Credential Expired
- "0024" Open Session Denied
- "0025" Password Mismatch
- "0026" Invalid Password Lentgh
- "0027" User temporary suspended
- "0028" XAdES Transformations are disabled
- "0029" Wrong XAdES Transformation
- "0030" Domain Not Allowed
- "0031" Invalid page index, must be greater than 0
- "0032" User pin is blocked (max wrong attemps reached)
- "0033" NaN is not a finite number (division by zero)
- "0034" Page index out of bounds

Credential Proxy module

- "1000" Credential proxy not enabled
- "1001" Credential not found
- "1002" Credential Proxy not enabled for domain
- "1003" Credential Proxy not active for domain
- "1004" Credential Proxy generic error