# SWS Integration Guide


- Introduction
- Soap Interface
  - Methods for automatic and remote signature
    - Method signPades
    - Method signCades
    - Method signXades
  - Methods only for remote signature
    - Method getOtpList
    - Method sendOtpBySMS
    - Method openSession
    - Method getRemainingTimeForSession
    - Method closeSession
  - Method for apply timestamp
    - Method timestamp
    - Method getAvailableTimestamps (since SWS v2.5.44)
- How Sign the file
  - Credentials Object
    - For automatic and remote signature
    - Only for remote signature
      - How works method getOTPList?
      - Sign with OTP SMS
      - Sign with OTP GENERATOR (App)
      - Sign with sessionKey
      - How obtain the sessionKey?
      - How check if the session has expired or valid
      - Destroy manually the session
      - Sequence diagram for sign with sessione and OTP SMS
      - Sequence diagram for sign with session with OTP App (da valutare)
    - Summarize
  - Populate the "buffer"
  - Signature Preferences
    - PadES Preferences
      - SignerImage
    - Cades Preferences
    - Xades Preferences
    - Level
  - How apply the timestamp
- Manage error in SWS
  - Method getErrors
- Examples (source code)


# Introduction

After install and configure you virtual appliance SWS, now you can use their method to sign or apply timestamp. SWS have two interfaces SOAP or REST. SOAP is used for files under 50MB and REST interface is used for files over 50MB.

SWS can manage some signature device like:

- automatic signature (her name start with AHI or AHIP followed by numbers)
- eSeal (her name start with SHI or SHIP followed by numbers)
- remote signature (her name start RHI or RHIP followed by numbers)
- disposable signature (her name start with RHI or RHID followed by numbers)
- long lived signature (her name start with RHIL or RHILD followed by numbers)

Only during the integration, you can see:

- eSeal like a automatic signature
- disposable, longlived like a remote signature

And the remote signature like an extension of automatic signature, because beyond username and password require the OTP code.

SWS supports three differents types of signature:

- Pades: valid only for PDF files
- Xades: valid only for XML files
- Cades: valid for every type of files

Apply timestamp on files (according to standard RFC3161)

For every type of signature and timestamp, there is a dedicated web method, which will be described in the the next sections.

In this user guide the examples will be shown using "SoapUI". This is a free tool which can be installed on every OS. With this tool, is possible to create SOAP request which invoke the differents web methods.

During the integration, the application client of SWS should recreate the same XML soap request created on SoapUI with his program language.

# Soap Interface

For test SOAP interface, you can make request with SoapUI, following this steps:

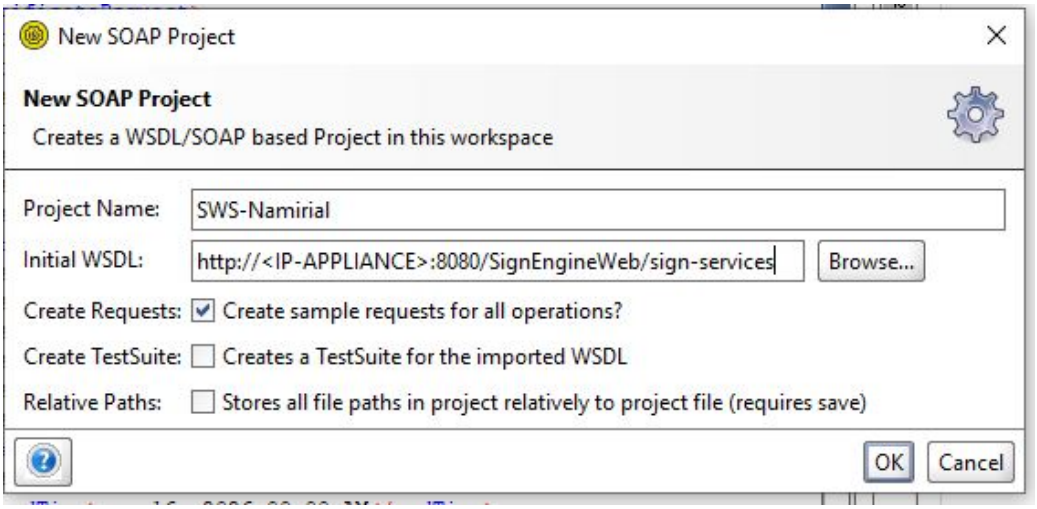
Download and install SoapUI from this link:

https://www.soapui.org/tools/soapui/

Once complete the installation:

open SoapUI  File  New Soap Project

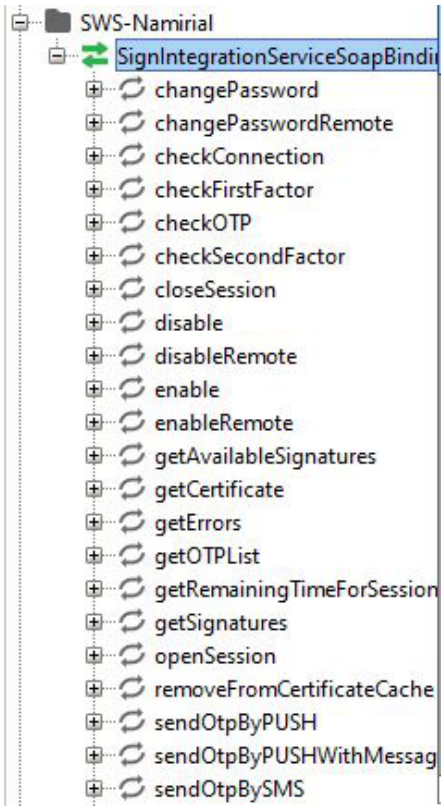
And add appliance SWS method to SoapUI, like in this image:




In text box “Initial WSDL” use this URL:

http://<IP-APPLIANCE>:8080/SignEngineWeb/sign-services?wsdl

And you will obtain the list of method like this:




## Methods for automatic and remote signature

The principal method used to sign (valid for remote and automatic signature), they are:

**signPAdES** Used for sign only PDF files

**signCAdES** Used for sign every type of files

**signXAdES** Used for sign XML files

**getSignatures** permit to obtain the number of signature made since certicate creation

**getAvailableSignatures** permit to obtain the numbers of signatures (valid only for device NOT pay per use, else it will generate an exception)

Every method require the Credentials object, in the next section will see how populate this field.

### Method signPades

This parameters required (IN) and the output (OUT) of this method can be specified with this table:

| signPadesList | | | |
|---|---|---|---|
| **Name** | **Type** | **Description** | **IN/OUT** |
| credentials | Credentials | See the section Credentials for see how populate this object | IN |
| bufferList | List<byte[]> | List of byte array which you want sign | IN |
| PAdESPreferences | PAdESPreferences | Specify the details of PadesSignature. See the section PadesPreferences for populate di object | IN |
| | List<byte> | List of byte array containg the file just signed | OUT |

### Method signCades

This parameters required (IN) and the output (OUT) of this method can be specified with this table:

| signPadesList | | | |
|---|---|---|---|
| **Name** | **Type** | **Description** | **IN/OUT** |
| credentials | Credentials | See the section Credentials for see how populate this object | IN |

| | | | |
|---|---|---|---|
| bufferList | byte[] | byte array which you want sign | IN |
| CAdESPreferences | CAdESPreferences | Specify the details of PadesSignature. See the section CadesPreferences for populate this object | IN |
| | byte[] | List of byte array containg the file just signed | OUT |

### Method signXades

This parameters required (IN) and the output (OUT) of this method can be specified with this table:

| signXadesList | | | |
|---|---|---|---|
| **Name** | **Type** | **Description** | **IN/OUT** |
| credentials | Credentials | See the section Credentials for see how populate this object | IN |
| bufferList | byte[] | byte array which you want sign | IN |
| XAdESPreferences | XAdESPreferences | Specify the details of XadesSignature. See the section XadesPreferences for populate this object | IN |
| | byte[] | byte array containg the file just signed | OUT |

## Methods only for remote signature

If you are signing with remote signature, you can use also this methods:

**getOTPList** permit to obtain the list of OTP associate to your remote signature (exactly the OTP is associated to the holder of certificate. For example if you have two or more remote signature associate to same holder, you can use this OTP for every remote signature).

**sendOtpBySMS** it will send the SMS containig the OTP code.

**openSession** permit to obtain the token (like a string) for sign instead to insert new OTP code on every signature). The token is valid for three minutes from generation.

**getRemainingTimeForSession** it return the time until the session is valid

**closeSession** if you want destroy the token before three minutes (howewer will expire after three minutes)

### Method getOtpList

| getOtpList | | | |
|---|---|---|---|
| **Name** | **Type** | **Description** | **IN/OUT** |
| credentials | Credentials | See the section Credentials for see how populate this object | IN |
| | List<OTP> | List of OTP associate to the Credentials | OUT |

### Method sendOtpBySMS

| sendOtpBySMS | | | |
|---|---|---|---|
| **Name** | **Type** | **Description** | **IN/OUT** |
| credentials | Credentials | See the section Credentials for see how populate this object | IN |

At the end of this method the customer will receive the SMS with OTP code to use.

### Method openSession

| openSession | | | |
|---|---|---|---|
| **Name** | **Type** | **Description** | **IN/OUT** |
| credentials | Credentials | See the section Credentials for see how populate this object | IN |
| | String | Sessionkey to use for sign | OUT |

At the end of this method the customer will receive the SMS with OTP code to use.

### Method getRemainingTimeForSession

| getRemainingTimeForSession | | | |
|---|---|---|---|
| **Name** | **Type** | **Description** | **IN/OUT** |
| credentials | Credentials | See the section Credentials for see how populate this object | IN |

| | | | |
|---|---|---|---|
| | int | remaing seconds until the session is valid | OUT |

### Method closeSession

| closeSession | | | |
|---|---|---|---|
| **Name** | **Type** | **Description** | **IN/OUT** |
| credentials | Credentials | See the section Credentials for see how populate this object | IN |

At the end of this method the session will be destroyed

## Method for apply timestamp

SWS offer method to apply timestamp and enquiry (only for Namirial account)

**timestamp** it permits to obtain the file with timestamp is possible to choose between two types TSR or TSD. The option TSR mean the timestamp is another files, while the TSD mean the timestamp signature is in the same file.

**getAvailableTimestamps** it permits to obtain the timestamp available ONLY for Namirial account

After this introduction, below will be described every method with input required.

### Method timestamp

| timestamp | | | |
|---|---|---|---|
| **Name** | **Type** | **Description** | **IN/OUT** |
| content | byte[] | byte array where apply the timestamp | IN |
| preferences | TimeStampPreferences | preferences about timestamp url, username, password ecc.. | IN |
| | byte[] | timestamp in binary format | OUT |

This method can be used with all timestamp account (not only Namirial) they must use standard RFC3161.

NOTE: Since SWS v2.5.44 this method support Adobe Timestamp on timestampPreferences you should set "outputAsPDF=true".

### Method getAvailableTimestamps (since SWS v2.5.44)

| **getAvailableTimestamps** | | | |
|---|---|---|---|
| **Name** | **Type** | **Description** | **IN/OUT** |
| preferences | TimeStampPreferences | timestamp url, username, password | IN |
| | Long | number of timestamp available. With account payperuse will be generated an Exception | OUT |

NOTE: TimestampUrl can be set to:

| **TIMESTAMP URL** | **Environment** |
|---|---|
| https://timestamp.namirialtsp.com | PROD |
| https://timestamp.test.namirialtsp.com | TEST |

# How Sign the file

For sign the file with SWS every method require this parameters:

- Credentials: contain the value about signature device
- Preferences: contain the signature details like page, appereance ecc..., Level of signature (B, T, LT ecc...). There are different type of preferences PadesPreferences, CadesPreferences, XadesPreferences
- buffer: file which you want sign

In the sections will see how set this parameters

## Credentials Object

All the methods used for sign (signPAdES..., signCAdES..., signXAdES) they use the Credentials object, like you can see in this image:

```
<credentials>
   <idOtp>?</idOtp>
   <otp>?</otp>
   <password>?</password>
   <securityCode>?</securityCode>
   <sessionKey>?</sessionKey>
   <username>?</username>
</credentials>
```

How popolate this fields?

## For automatic and remote signature

For every type of signature (automatic signature and remote signature) you must fill this two fields:

**username**: contain the device name it start with RHI..., AHI... or SHI...

**password**: contain the PIN associated to device (read from blind envelope or set by the customer)

## Only for remote signature

While only if you you are using the remote signature (username starts with RHI...) you should fill this fields:

**idOtp**: (optional) it specify the idOtp which you want use for sign. If you don't want set the idOtp, SWS will use automatically the default OTP. You can use the method getOTPList for obtain the idOtp.

**Otp**: it contains the OTP code recived by SMS or read on app Namirial

**sessionKey**: it contain the token (like a string) received from method openSession

**securityCode**: this parameter must not be set. It is used only in certain situation

### How works method getOTPList?

With this method you can obtain the OTP list which can be use with specified username, and you will can populate the variable Credentials.idOtp.

This method it require only the username.

For example with username: RHIP20102336019765, in this request:

**REQUEST-getOTPList**

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:ser="http://service.ws.nam
/">
   <soapenv:Header/>
   <soapenv:Body>
      <ser:getOTPList>
         <credentials>
            <username>RHIP20102336019765</username>
         </credentials>
      </ser:getOTPList>
   </soapenv:Body>
</soapenv:Envelope>
```

You will obtain response like this:

**RESPONSE-getOTPList**

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
   <soap:Body>
      <ns2:getOTPListResponse xmlns:ns2="http://service.ws.nam/">
         <return>
            <idOtp>501719</idOtp>
            <serialNumber>20210113-091031RJ2L1</serialNumber>
            <type>SMS</type>
         </return>
         <return>
            <idOtp>537430</idOtp>
            <serialNumber>20210305-163726L0PYF</serialNumber>
            <type>OTP GENERATOR</type>
         </return>
         <return>
            <idOtp>537433</idOtp>
            <serialNumber>20210305-163726F0I75</serialNumber>
            <type>OTP PUSH</type>
         </return>
      </ns2:getOTPListResponse>
   </soap:Body>
</soap:Envelope>
```

During the sign process is possible to choose between this two idOtps: 501719 (associated to OTP SMS) and the idOTP: 537430 (associated to OTP GENERATOR).

Isn't possible to use OTP PUSH, they are used for other purpose, not for sign.

Now during the sign we can choose two types of idOTP: 501719 or 537430.

### Sign with OTP SMS

If you decide to sign with OTP SMS, you should use the method: **sendOTPBySMS**

This method require in input only the username (in this example the username is: RHIP20102336019765).

The soap request will be like this:

**REQUEST-sendOTPBySMS**

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:ser="http://service.ws.nam
/">
   <soapenv:Header/>
   <soapenv:Body>
      <ser:sendOtpBySMS>
         <credentials>
            <username>RHIP20102336019765</username>
         </credentials>
      </ser:sendOtpBySMS>
   </soapenv:Body>
</soapenv:Envelope>
```

And if everything is ok, in output will receive the response like this:

**RESPONSE-sendOTPBySMS**

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
   <soap:Body>
      <ns2:sendOtpBySMSResponse xmlns:ns2="http://service.ws.nam/"/>
   </soap:Body>
</soap:Envelope>
```

And on mobile phone will receive the SMS containing the OTP code (composed from 6 number) for sign, for example now we have received the code: “214196”.

The OTP code just received will be the variable Crediantls.otp during the process of sign.

### Sign with OTP GENERATOR (App)

If you decide to sign with OTP GENERATOR, you should open the Namirial OTP App and insert the OTP code showed during the sign process.

Show the guide "How to configure Namirial OTP App" (To Do/Add)

### Sign with sessionKey

With otp is possible to make only one signature, but if you have need to sign more files, with the “sessionKey” is possible. In the next section will be described how works the session.

This function is available only for remote signature, it permits to sign at most 3 minutes using the same sessionKey. You can see the session like a token provided from method “openSession”.

### How obtain the sessionKey?

The method “openSession” it permits to obtain the sessionKey.

In input it require:

- username
- password
- otp
- idOtp

Like in this example:

**REQUEST-openSession**

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
        xmlns:ser="http://service.ws.nam/">
   <soapenv:Header/>
   <soapenv:Body>
      <ser:openSession>
         <credentials>
            <idOtp>501719</idOtp>
            <otp>150259</otp>
            <password>13572468</password>
            <username>RHIP20102336019765</username>
         </credentials>
      </ser:openSession>
   </soapenv:Body>
</soapenv:Envelope>
```

In output will obtain the value of sessionKey which will be used for sign:

**RESPONSE-REMOTE-openSession**

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
   <soap:Body>
      <ns2:openSessionResponse xmlns:ns2="http://service.ws.nam/">
                <return>
                        f4lf7bq/cCxW6mTgL3iGjFEST5cEAZjgLnXvV3hUFzFHcTvjlH3FOkJy+kv/0Zsv1
                        uNK0S7L6jMqHYSspBz+CZl7h3r5IEP2FqrK7WJQTVyrNfyr/trZmDgxYOLuACyoZVUFIlnck5Lkjihui
                        sv+gZeB68Spwm+cNDdQQdUS3ngzJavHXxo9ADCX6VDIKKMe
/AY0v+R51XWE90JF5LfKEThlv1OCpQC5nhnW8WKOFOm
                        P4vM90d79JhFYGVVSZWtnTQ9Dg8pOMvg9wwxNm3uGkKKaS7oTp1ewd+eCG/uSC9k3H2w9GB6vQLHQEbn6d
                        VVMcsIqJ0RMmZ2IgraD+scb4Q==
                </return>
      </ns2:openSessionResponse>
   </soap:Body>
</soap:Envelope>
```

The sessionKey just obtained is valid for three minutes (isn't possible to edit this value!), after will expire and will needed to generate another sessionKey using the method openSession and new OTP code (isn't possible to use the same OTP already used).

### How check if the session has expired or valid

Is possible to know when the session will expire with method: “getRemainingTimeForSession”. This method require in input only:

- usernamame
- sessionKey (obtained from method "openSession"

Below the example:

**REQUEST-remote-getRemainingTimeForSession**

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
                        xmlns:ser="http://service.ws.nam/">
   <soapenv:Header/>
   <soapenv:Body>
      <ser:getRemainingTimeForSession>
         <credentials>
            <username>RHIP20102336019765</username>
                    <sessionKey>
                                                f4lf7bq/cCxW6mTgL3iGjFEST5cEAZjgLnXvV3hUFzFHcTvjlH3FOkJy+kv
/0Zsv1
                                                uNK0S7L6jMqHYSspBz+CZl7h3r5IEP2FqrK7WJQTVyrNfyr
/trZmDgxYOLuACyoZVUFIlnck5Lkjihui
                                                sv+gZeB68Spwm+cNDdQQdUS3ngzJavHXxo9ADCX6VDIKKMe
/AY0v+R51XWE90JF5LfKEThlv1OCpQC5nhnW8WKOFOm
                                                P4vM90d79JhFYGVVSZWtnTQ9Dg8pOMvg9wwxNm3uGkKKaS7oTp1ewd+eCG
/uSC9k3H2w9GB6vQLHQEbn6d
                                                VVMcsIqJ0RMmZ2IgraD+scb4Q==
                        </sessionKey>
                 </credentials>
      </ser:getRemainingTimeForSession>
   </soapenv:Body>
</soapenv:Envelope>
```

The SOAP response will be:

**RESPONSE-remote-getRemainingTimeForSession**

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
   <soap:Body>
      <ns2:getRemainingTimeForSessionResponse xmlns:ns2="http://service.ws.nam/">
         <return>167</return>
      </ns2:getRemainingTimeForSessionResponse>
   </soap:Body>
</soap:Envelope>
```

Where 167 are the seconds until the session is active. After 180s from creation will be destroyed automatically, but is good pratice close the session before will expire.

You can destroy the session manually before will expire with method: "closeSession"

### Destroy manually the session

The method "closeSession" require in input:

- sessionKey
- username

Below the SOAP request example:

**REQUEST-remote-closeSession**

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
        xmlns:ser="http://service.ws.nam/">
   <soapenv:Header/>
   <soapenv:Body>
      <ser:closeSession>
         <credentials>
            <sessionKey>
                                f4lf7bq/cCxW6mTgL3iGjFEST5cEAZjgLnXvV3hUFzFHcTvjlH3FOkJy+kv/0Zsv1
                                uNK0S7L6jMqHYSspBz+CZl7h3r5IEP2FqrK7WJQTVyrNfyr
/trZmDgxYOLuACyoZVUFIlnck5Lkjihui
                                sv+gZeB68Spwm+cNDdQQdUS3ngzJavHXxo9ADCX6VDIKKMe
/AY0v+R51XWE90JF5LfKEThlv1OCpQC5nhnW8WKOFOm
                                P4vM90d79JhFYGVVSZWtnTQ9Dg8pOMvg9wwxNm3uGkKKaS7oTp1ewd+eCG
/uSC9k3H2w9GB6vQLHQEbn6d
                                VVMcsIqJ0RMmZ2IgraD+scb4Q==
                        </sessionKey>
            <username>RHIP20102336019765</username>
         </credentials>
      </ser:closeSession>
   </soapenv:Body>
</soapenv:Envelope>
```

For security reason, this method doesn't generate an exception if you insert wrong sessionKey and/or username. The SOAP response will be ever like this:

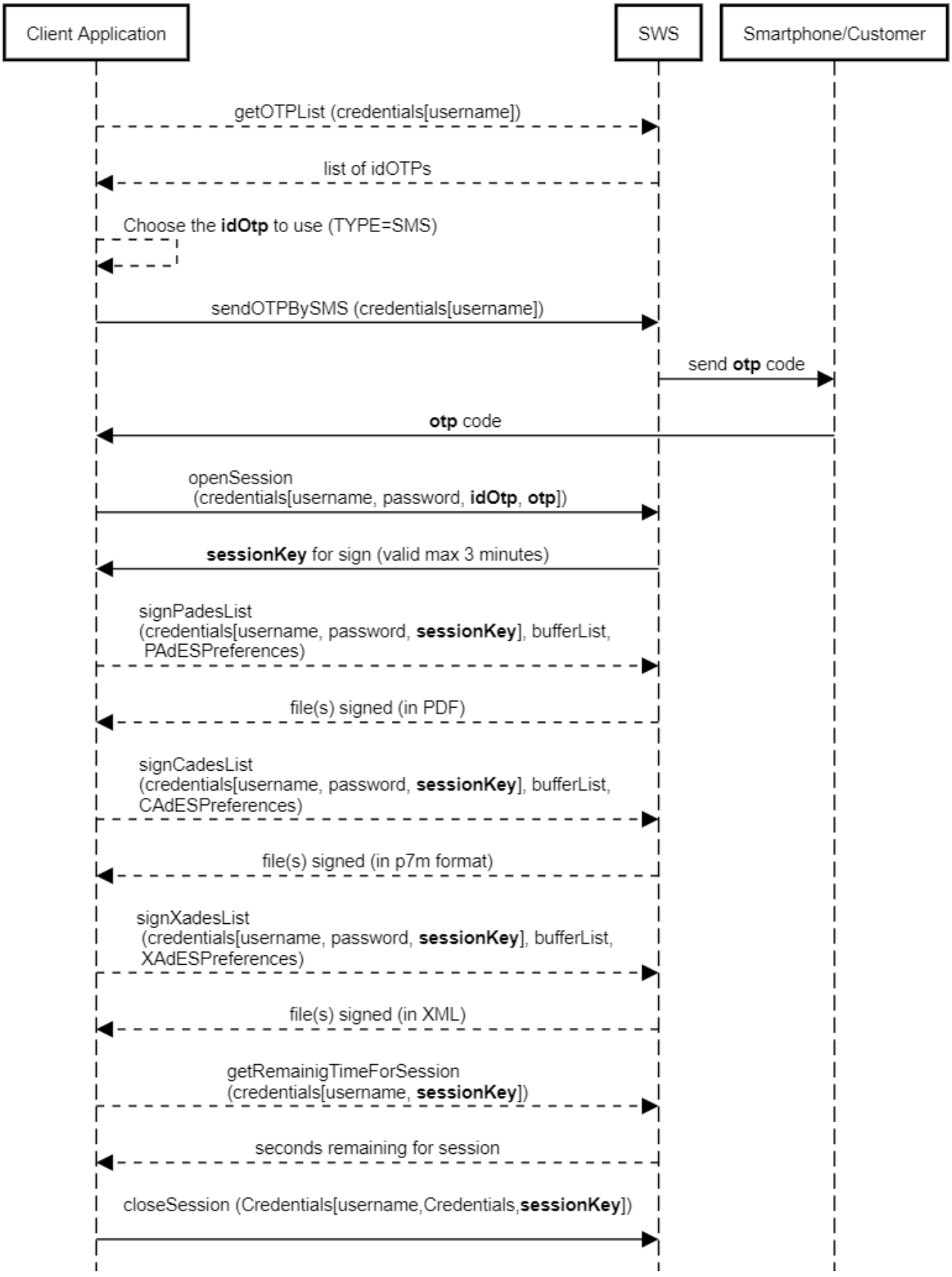
**RESPONSE-remote-closeSession**

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
   <soap:Body>
      <ns2:closeSessionResponse xmlns:ns2="http://service.ws.nam/"/>
   </soap:Body>
</soap:Envelope>
```

### Sequence diagram for sign with sessione and OTP SMS

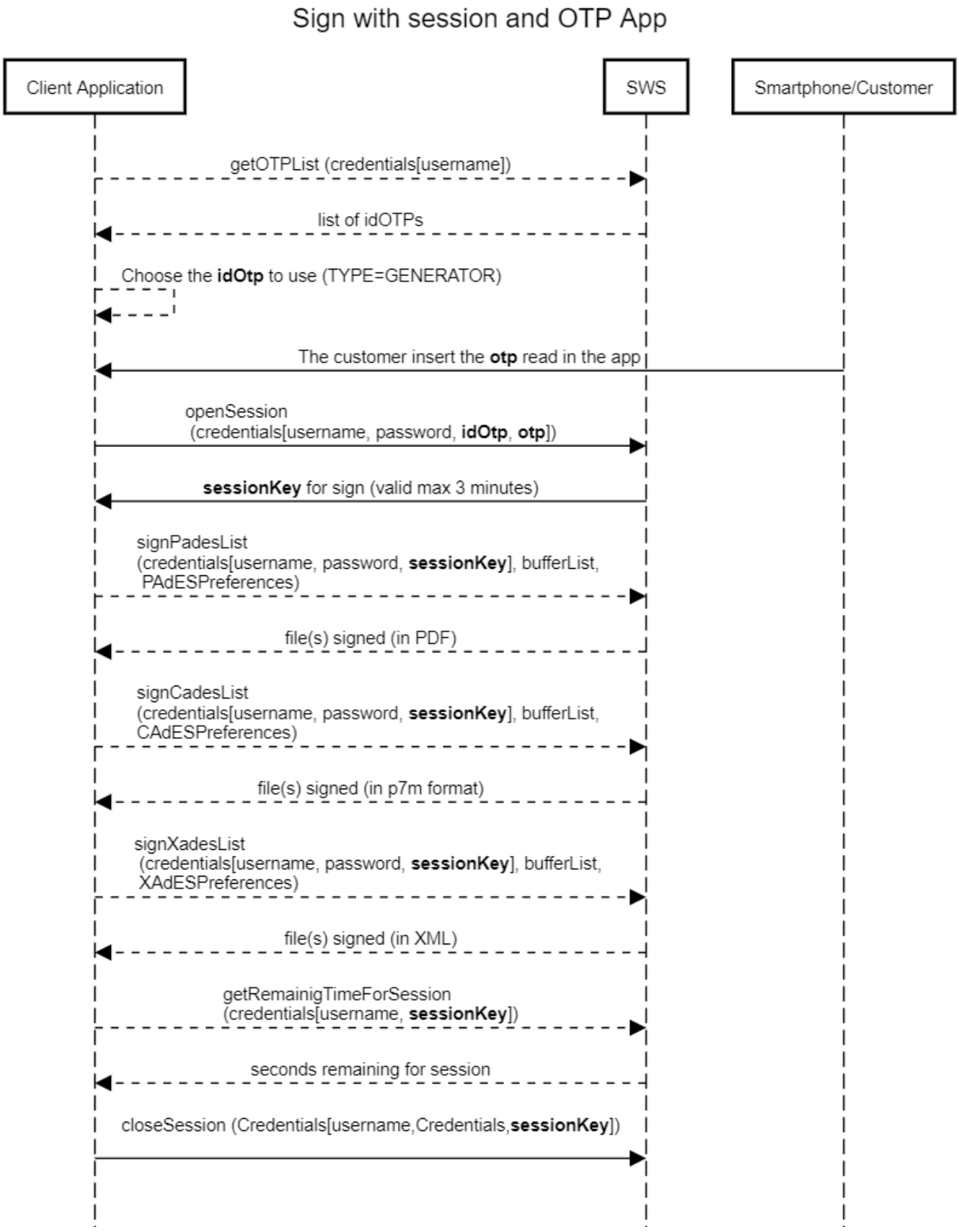
In this seguence diagram, we can sumarize the methods to call for sign using sessionKey and OTP SMS:

# Sign with session and OTP SMS



### Sequence diagram for sign with session with OTP App (da valutare)

In this seguence diagram, we can sumarize the methods to call for sign using sessionKey and OTP SMS:




## Summarize

Finally we have all requisites to populate Credentials object during the sign. Like mentioned before, the methods to sign are:

- signPAdES
- signCAdES
- signXAdES

There are the same methods with suffix "List", they accept in input a list of files to be signed. Therefore with only SOAP request is possible to sign more files (using automatic signature or sessionKey)

With this three methods is possible to sign with every type of signature (automatic and remote).

Everyone of this three methods use the Credentials object filled in the same time.

The automatic signature, require only the variables username and password in the object Credentials.

For example in automatic signature with username: AHI7609757152622 and password 13572468 the object Credentials will be populate like in the image:

**REQUEST-AUTOMATIC-Credentials**

```
<credentials>
        <username>RHIP20102336019765</username>
        <password>1357268</password>
</credentials>
```

While if you are using remote signature you should fill the other fields:

- idOtp (only if you have more idOTP received from method getOTPList)
- OTP or sessionKey (will see in the next section how populate this variable)

Suppose we want sign using with the OTP code received previously from method sendOtpBySMS.

The credentials object will be filled in this way:

**REQUEST-Credentials-Remote-OTP-SMS**

```
<credentials>
        <idOtp>501719</idOtp>
        <otp>150259</otp>
        <password>13572468</password>
        <username>RHIP20102336019765</username>
</credentials>
```

**idOtp** was obtained from method getOTPList method and **otp** is the code received from method sendOTPBySMS.

Therefore for automatic signature the credentials object is composed by:

- username
- password

While for remote signature the credentials object is composed by:

- username
- password
- otp
- idOtp (only if you have more OTP else you can set this to "-1")
- sessionKey (optional)

If you need to sign multiple files with remote signature you should use the sessionKey how already described.

Now, is complete how populate the Credentials object for methods: signPades, signCades and signXades, we can populate the object buffer.

Now we should populate the value of:

- buffer
- Prefecences of signature (there are different types for every type of signature)

## Populate the "buffer"

The buffer contain the file (in byte array) which you want sign.

For example in SoapUI the buffer is composed by the base64 of file which you want sign, like in this example:

**REQUEST-remote-signPades**

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:ser="http://service.ws.nam
/">
   <soapenv:Header/>
   <soapenv:Body>
      <ser:signPAdES>
                        <credentials>
                  <idOtp>501719</idOtp>
                  <otp>548316</otp>
                  <password>13572468</password>
                  <username>RHIP20102336019765</username>
               </credentials>
               <buffer>BASE64-FILE-TO-SIGN</buffer>
                       <PAdESPreferences>
                              <level>B</level>
                              <signerImage></signerImage>
                      </PAdESPreferences>
      </ser:signPAdES>
   </soapenv:Body>
</soapenv:Envelope>
```

You can dowload the full exampe at this link: signPadesList.xml

In output will obtain the base64 associated to file just signed like this: RESPONSE-base64-signPadesList.b64 and decoded will be this PDF: RESPONSE-signPadesList.pdf.

## Signature Preferences

The difference between signPades, signCades and signXades are based on the preferences:

signPades  use PadESPreferences

signCades  use CadESPreferences

signXades  use XadESPreferences

How populate this preferences will be describe in the next sections.

### PadES Preferences

This type of preference is used in method signPades. Their principal options are:

| PAdESPreferences | | | | | |
|---|---|---|---|---|---|
| **Name** | **Type** | **Mandatory** | **Default value** | **Description** | **Included from SWS version** |
| hashAlgorithm | String | | SHA256 | Algorithm which you want use for sign. Possibile value are: SHA1, SHA256, SHA384, SHA512 | |
| level | Level | | B | See the description of Level type | |
| signType | int | | | | |

| | | | | | |
|---|---|---|---|---|---|
| encryptInAnyCase | boolean | | false | | |
| filenameInTSD | String | | | Not used | |
| outputAsTSD | boolean | | | Not used | |
| withTimestamp | boolean | | false | Specify if you want add or not the timestamp to file signed | |
| outputBase64Encoded | boolean | | false | Set to true if you want file signed in Base64 encode | |
| timestampHashAlgo | String | | SHA-256 | Algorithm which you want to use during the process of apply timestamp. | |
| timestampUrl | String | | | URL of timestamp provider with standard RFC3161.<br><br>Namirial URL:<br><br>PROD: https://timestamp.namirialtsp.com / http://timestamp.namirialtsp.com<br><br>TEST: https://timestamp.test.namirialtsp.com / http://timestamp.test.namirialtsp.com | |
| timestampUsername | String | | | Username of timestamp credentials | |
| timestampPassword | String | | | Password of timestamp credentials | |
| lockFields | List<String> | | | | |
| needAppearanceDisabled | boolean | | false | Deprecated | |
| page | | | 1 | Indicate the page number where you want apply the signature appereance. If you want add the appereance on last page of the PDF, you should set to "-1". | |
| withTimestamp | boolean | | false | Set to true if you want apply the timestamp after the signature | |
| encryptionPassword | String | | | Specify the password PDF if present | |
| lockFields | List<String> | | | | |
| signerImage | SignerImage | | | See the description of SignerImage | |
| signerImageReference | String | | | Used for specify the template to be used. (used in old version) | |
| withSignatureField | boolean | | false | Set to true if you want apply the signature on signature field in the PDF | |

## SignerImage

The object SignerImage is composed by the following details:

| SignerImage | | | | | |
|---|---|---|---|---|---|
| **Name** | **Type** | **Mandatory** | **Default value** | **Description** | **Included from SWS version** |
| image | byte[] | | | Contains the image which you want apply on the appereance | |
| signerName | String | | | Contains the text which you want type on the appereance | |
| reason | String | | | Specify the reason about the signature | |
| textVisible | boolean | | true | permits to show or not the text on appereance | |
| textPosition | String | | | Position of the "signerName" on appereance. Is possible to choose between:<br><br>• TOP<br>• BOTTOM<br>• RIGHT<br>• LEFT | |
| x | int | | | Coordinate X of the appereance (0 is right of the page) | |
| y | int | | | Coordinate Y of the appereance (0 is on bottom of the page) | |
| width | int | | | Specify the width of the appereance | |
| height | int | | | Specify the height of the appereance | |

| | | | | | |
|---|---|---|---|---|---|
| fieldName | | | | Specify the fieldname where apply signature. This fieldName must already exist on PDF file before apply the signature | |
| fontName | String | | Times-Roman | Specify the font of the text on appereance to be used. The possible values are:<br>• Times-Roman<br>• Times-Bold<br>• Times-Italic<br>• Times-BoldItalic<br>• Helvetica<br>• Helvetica-Bold<br>• Helvetica-Oblique<br>• Helvetica-BoldOblique<br>• Courier<br>• Courier-Bold<br>• Courier-Oblique<br>• Courier-BoldOblique<br>• Symbol<br>• ZapfDingbats | |
| fontName | String | | | Specify the ttf path which contain custom font | 2.5.39 |
| imageURL | String | | | URL to obtain the logo for appereance | |
| imageVisible | boolean | | false | permits to show or not the logo on appereance | |
| fontSize | int | | 10 | permits to set the fontsize | |
| imageFilename | String | | | path of the logo on appereance | |
| scaled | boolean | | false | Set to true if you want resize the logo on appereance | |
| location | | | | place of signature | |

INSERT EXAMPLE WITH APPEREANCE

## Cades Preferences

With cades signature is possible to sign every type of file, the method signCades require:

- Credentials associated to device signature
- buffer, file which you want sign
- CAdESPreferences, the preferences about CAdES signature

In the following table you can see how set correctly the CAdESPreferences

| **CAdESPreferences** | | | | | |
|---|---|---|---|---|---|
| **Name** | **Type** | **Mandatory** | **Default value** | **Description** | **Included from SWS version** |
| filenameInTSD | | | | | |
| outputAsTSD | | | | | |
| outputBase64Encoded | boolean | | false | Encoded the file just signed in base64 | |
| timestampHashAlgo | String | | SHA-256 | Algorithm which you want to use during the process of apply timestamp. | |
| timestampPassword | | | | | |
| timestampUrl | String | | | URL of timestamp provider with standard RFC3161.<br>Namirial URL:<br>PROD: https://timestamp.namirialtsp.com / http://timestamp.namirialtsp.com<br>TEST: https://timestamp.test.namirialtsp.com / http://timestamp.test.namirialtsp.com | |
| timestampUsername | String | | | Username of timestamp credentials | |
| hashAlgorithm | String | yes | SHA256 | Algorithm which you want use for sign. Possibile value are: SHA1, SHA256, SHA384, SHA512 | |

| | | | | | |
|---|---|---|---|---|---|
| level | Level | | B | See the description of Level type | |
| withTimestamp | boolean | | false | Set to true if you want apply the timestamp after the signature | |
| counterSignature | | | | | |
| counterSignatureIndex | | | | | |
| detached | boolean | | false | Set to true if you want signature and files in two different files. The output will be the signature. | |

## Xades Preferences

With xades signature is possible to sign only XML files, the method signXades require;

- Credentials associated to device signature
- buffer, file which you want sign
- XAdESPreferences, the preferences about XAdES signature

In the following table you can see how set correctly the XAdESPreferences

| XAdESPreferences | | | | | |
|---|---|---|---|---|---|
| **Name** | **Type** | **Mandatory** | **Default value** | **Description** | **Included from SWS version** |
| filenameInTSD | | | | | |
| outputAsTSD | | | | | |
| outputBase64Encoded | boolean | | false | Encoded the file just signed in base64 | |
| timestampHashAlgo | String | | SHA-256 | Algorithm which you want to use during the process of apply timestamp. | |
| timestampPassword | | | | | |
| timestampUrl | String | | | URL of timestamp provider with standard RFC3161.<br><br>Namirial URL:<br><br>PROD: https://timestamp.namirialtsp.com / http://timestamp.namirialtsp.com<br><br>TEST: https://timestamp.test.namirialtsp.com / http://timestamp.test.namirialtsp.com | |
| timestampUsername | String | | | Username of timestamp credentials | |
| hashAlgorithm | String | yes | SHA256 | Algorithm which you want use for sign. Possibile value are: SHA1, SHA256, SHA384, SHA512 | |
| level | Level | | B | See the description of Level type | |
| withTimestamp | boolean | | false | Set to true if you want apply the timestamp after the signature | |
| detached | boolean | | false | Set to true if you want signature and files in two different files. The output will be the signature. | |
| detachedReferenceURI | String | | | | |
| signElement | String | | | | |
| signatureId | String | | | | |
| withoutSignatureExclusion | boolean | | false | Permits to sign the file with/without previous signature | |
| XPathQuery | String | | | Permetis to sign a specified path of XML | |

Below the example of Xades Signature Level B:

signXadesList-Level-B.txt

## Level

You can see how set the correct Level signature:

| Level |
|---|

| Value | Description | Apply on signature | Included from SWS version |
|---|---|---|---|
| B | in the file signed will be added the electronic signature and the signing certificate | Pades, Cades, Xades | |
| T | Like B-Level, but adds a time-stamp, respectively a time-mark that proves that the signature existed at a certain date and time | Pades, Cades, Xades | |
| LT | Like T-Level, but adds VRI (Verification Related Information) data to the DSS (Long Term) | Pades, Cades, Xades | |
| LTA | Like LT-level, but adds a document time stamp and VRI data for the TSA (Time Stamping Authority). An LTA may help to validate the signature beyond any event that may limit its validity (Long Term with Arichive Time-Stamps) | Pades, Cades, Xades | |
| LTV | (Long Term Validation) contain the OCSP/CRL response after the sign. It is used for validation after the signing certificate has been expired | Pades | |

## How apply the timestamp

Is possible to apply timestamp wit the method "timestamp", in input require:

- content: byte array of file to apply timestamp
- preferences: object with contains details about timestamp

Below the object timestamp:

| Name | Type | Mandatory | Default value | Description | Included from SWS version |
|---|---|---|---|---|---|
| filenameInTSD | | | | | |
| outputAsTSD | | | | | |
| outputBase64Encoded | boolean | | false | Encoded the file just signed in base64 | |
| timestampHashAlgo | String | | SHA-256 | Algorithm which you want to use during the process of apply timestamp. | |
| timestampPassword | | | | | |
| timestampUrl | String | | | URL of timestamp provider with standard RFC3161.<br><br>Namirial URL:<br><br>PROD: https://timestamp.namirialtsp.com / http://timestamp.namirialtsp.com<br><br>TEST: https://timestamp.test.namirialtsp.com / http://timestamp.test.namirialtsp.com | |

# Manage error in SWS

Every method can generate exception, for example caused by PIN not correct, sessioneKey expired or OTP not correct.

For example if we can try to execute the method signPAdESList using the same OTP used we obtain SOAP response with error 44, like in this response:

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
   <soap:Body>
      <soap:Fault>
         <faultcode>soap:Server</faultcode>
         <faultstring>Codice OTP errato, riprovare con il prossimo codice</faultstring>
         <detail>
            <ns2:WSException xmlns:ns2="http://service.ws.nam/">
               <error>44</error>
               <message>Codice OTP errato, riprovare con il prossimo codice</message>
            </ns2:WSException>
         </detail>
      </soap:Fault>
   </soap:Body>
</soap:Envelope>
```

By default the error message is in Italian language.

Below the table description with all error messages can generate SWS during your execution method:

| **Error details** | | |
|---|---|---|
| Error number | Description | |
| | English | Italian |
| 0 | No errors found | Nessun errore riscontrato |
| 1 | Generic error | Errore Generico |
| 2 | Virtual device not found | Dispositivo virtuale inesistente |
| 3 | Virtual device locked | Dispositivo virtuale bloccato |
| 4 | Wrong credentials | Credenziali errate |
| 5 | Wrong emergency code | Codice di emergenza errato |
| 6 | Virtual device status changes denied | Modifiche allo stato del dispositivo virtuale negate |
| 7 | Signature error | Errore nella firma |
| 8 | Error creating slot | Errore nella creazione dello slot |
| 9 | Error deleting slot | Errore nella eliminazione dello slot |
| 10 | PIN change error | Errore nel cambio PIN |
| 11 | Key generation error | Errore nella generazione chiave |
| 12 | Error in key management configuration | Errore nella configurazione del sistema di gestione delle chiavi |
| 13 | Wrong company code | Codice azienda errato |
| 14 | No available slots | Nessuno slot disponibile |
| 15 | Virtual device already exists | Dispositivo virtuale gia' esistente |
| 16 | Operation performed using a wrong certificate | Operazione eseguita usando il certificato errato |
| 17 | Wrong virtual device code | Codice dispositivo virtuale errato |
| 18 | Slot already used | Slot gia' utilizzato |
| 22 | Incompatible file format for the signature type required | Richiesta una firma di file di formato non compatibile con il tipo di firma richiesto |
| 23 | Unsupported hash algorithm | Algoritmo di hash non supportato |
| 24 | Error decrypting CMS data | Errore nella decifratura del CMS EnvelopedData |
| 25 | Error importing key and certificates | Errore nell'importazione di chiave e certificati |

| | | |
|---|---|---|
| 26 | The public key in the certificate does not match the private key | Chiave pubblica nel certificato non corrisponde a quella privata |
| 27 | Web method denied for the credentials or ssl certificate used | Eseguita una chiamata a web method mediante credenziali o certificato ssl non abilitato per questa funzione |
| 28 | CA doesn't exist | La CA inserita non esiste |
| 29 | The user didn't enter all required fields for the profile | L'utente non ha inserito tutti i campi richiesti per il profilo |
| 30 | EJBCA error | Errore di EJBCA |
| 31 | Authorization denied | Autorizzazione negata |
| 32 | Error due to waiting for data approval | Errore dovuto all'attesa per l'approvazione dei dati |
| 33 | Error approving the entered data | Errore nell'approvazione dei dati inseriti |
| 34 | Illegal query | Errore per query illegale |
| 35 | Certificate already revoked | Certificato gia' revocato in precedenza |
| 36 | I / O error, caused by writing / reading / converting a file / byte array / string | Errore di I/O, causato dalla scrittura/lettura/conversione di un file/array di byte /stringa |
| 37 | Payment verification failed | Verifica di pagamento non andata a buon fine |
| 38 | No available signatures | Eseguite tutte le firme a disposizione |
| 42 | A denied feature is invoked in the current mode | E' stata richiamata una funzionalita' non permessa nella modalita' corrente |
| 43 | A denied feature is invoked in the implementation used | E' stata richiamata una funzionalita' non permessa nell'implementazione usata |
| 44 | Wrong OTP code, try again with the next code | Codice OTP errato, riprovare con il prossimo codice |
| 45 | The key isn't associated to a certificate | La chiave non ha associato un certificato |
| 46 | Unknown certificate format | E' stato passato un certificato di formato sconosciuto |
| 47 | It isn't possible to open the slot | Non e' stato possibile aprire lo slot |
| 49 | Key login error | Errore di login sulla chiave |
| 50 | Error generating the CSR | Errore nella generazione del CSR |
| 51 | The maximum number of attempts to access the virtual device is reached | Raggiunto il numero massimo di tentativi di accesso al dispositivo virtuale |
| 52 | Error decrypting | Errore nella decifra |
| 53 | The certificate has expired | Il certificato associato alla chiave e' scaduto |
| 54 | There are no tokens for automatic signature with Cosign HSM | Non sono disponibili token per la firma automatica con hsm Cosign |
| 55 | Error updating certificate in db | Errore durante l'aggiornamento del certificato nel db |
| 56 | Wrong method use | Errato utilizzo del metodo |
| 57 | Method not yet implemented | Metodo non ancora implementato |
| 58 | Error assigning the OTP | Errore durante l'assegnazione dell'OTP |
| 59 | Error assigning the static token | Errore durante l'assegnazione del token statico |
| 60 | Error deleting the account | Errore durante la cancellazione dell'account |
| 61 | Error activating the account | Errore durante l'attivazione dell'account |
| 62 | Error loading the account | Errore durante il caricamento dell'account |
| 63 | Error unlocking the account | Errore durante lo sblocco dell'account |
| 64 | Unavailable hsm licenses | Licenze per hsm esaurite |
| 65 | PIN too short | PIN troppo corto |
| 66 | Session key incorrect | Session key errata |
| 67 | Session key not specified | Session key non specificata |

| | | |
|---|---|---|
| 68 | Session key undefined | Session key non definita |
| 69 | Session key expired | Session key scaduta |
| 70 | Session key not usable | Session key non utilizzabile |
| 71 | Error generating session key | Errore durante la generazione della session key |
| 72 | Error incrementing the session counter | Errore durante l'incremento del session counter |
| 73 | Error sending OTP code | Errore durante l'invio del codice OTP |
| 74 | Error deleting session key | Errore durante la cancellazione della session key |
| 77 | Error closing session | Errore durante la chiusura della sessione |
| 78 | The number of documents to be signed differs from the number of signature preferences | Il numero di documenti da firmare differisce dal numero di preferenze di firma |
| 79 | Error detecting Security World | Errore durante il rilevamento del Security World |
| 80 | Error detecting the Module | Errore durante il rilevamento del Modulo |
| 81 | Error reading the SoftCard | Errore durante la lettura della SoftCard |
| 82 | Error writing the SoftCard | Errore durante la scrittura della SoftCard |
| 83 | Error deleting the SoftCard | Errore durante la cancellazione della SoftCard |
| 84 | Error loading SoftCard | Errore durante il caricamento della SoftCard |
| 85 | SoftCard not loaded | SoftCard non caricata |
| 86 | SoftCard already exists in the system | SoftCard gia' esistente a sistema |
| 87 | SoftCard does not exist | SoftCard inesistente |
| 88 | Error reading the key | Errore durante la lettura della chiave |
| 89 | Error writing the key | Errore durante la scrittura della chiave |
| 90 | Error deleting the key | Errore durante la cancellazione della chiave |
| 91 | Error decrypting the RSA data | Errore durante la decifratura RSA |
| 92 | Error decrypting the CMS envelope | Errore durante la decifratura CMS |
| 93 | Error creating the SoftCard | Errore durante la creazione della SoftCard |
| 94 | The size of the hash does not coincide with the expected one by the algorithm | La dimensione dell'hash non coincide con quella prevista dall'algoritmo |
| 95 | Error loading Cosign Tokens | Errore durante il caricamento dei Token Cosign |
| 96 | The system takes too much time, HSM overload. Try again | Il sistema impiega troppo tempo, HSM sovraccarico. Riprovare |
| 97 | Timeout passed | Timeout superato |
| 98 | No signature device associated to the user | Nessun dispositivo di firma remota risulta associato all'utente in questione |
| 1001 | The OTP device does not exist | Dispositivo OTP non esistente a sistema |
| 1007 | The OTP device was not activated | Il dispositivo OTP non risulta essere stato attivato |
| 1009 | Unavailable attempts for the OTP device | Superato il numero massimo di tentativi per il dispositivo OTP |
| 1016 | The OTP device was not associated to the holder | Il dispositivo OTP non risulta essere stato associato al titolare |

## Method getErrors

This method return a list of errors which can be generated from SWS in in

| Name | Type | Optional | Description | IN/OUT |
|---|---|---|---|---|
| lang | String | | String county code in 2 digit, accept only EN, IT | IN |
| errorCode | Integer | true | specify error code which you want obtain the error description | IN |
| | List<ErrorDetails> | | Return a list with error(s) description | OUT |

In this method is possible to obtain the list of all errors, without set the value of errorCode.

# Examples (source code)

Below will find the links contains the source code with example

Java:

**To add on CMS repo**

Php:

C#: https://cms.firmacerta.it/download/sws_cnet.zip

C# (for SaaS instance): https://cms.firmacerta.it/download/SignEngineWebClientSaaS.zip

**ADVANCED USE (visible or not?)**

For example signPkcs1

**VERIFY TIMESTAMP**

While for verify only timestamp, you can use this methods:

**timestampTSDVerify** It permits to validate TSD files (file and timestamp in the same file)

**timestampTSRVerify** It permits to validate TSR files (file and timestamp in two different files)