Revenue
Cáin agus Custaim na hÉireann
Irish Tax and Customs

Customs & Excise

SOAP Web Service Integration Guide

Contents

Version Control		
Version	**Date**	**Change**
0.1	04/10/2019	Initial document published
0.2	27/01/2022	Updated reference to AES

Document References

Reference
1. Web Services Specifications (incl. AES Operational January 2023 and AES)
2. Customs & Excise SOAP Web Service Integration Guide
3. Customs & Excise REST Web Service Integration Guide

1. Introduction

This document details the SOAP/XML web services specification for Customs and Excise web services. This specification is a result of the modernisation work that was done as part of the AIS and AES introduction. The modernisation includes the adoption of the latest specifications in an industry standard, such as SOAP 1.2 Specification, WS Security 1.1.1 and WSDL 2.0.

Please note: The AIS and AES web services, such as Submit Import/Export web service, are exclusively made available under the SOAP specification as described in this document, the other existing Custom and Excise web services will be still supported under the old specification that can be found at the following link: Web specifications for Customs and Excise system.

This document assumes familiarity with the SOAP/XML web services. The Web Service Description Language (WSDL) is a W3C standard for describing web services. Further details of each web service can be found in the supporting WSDL file. Each file references the necessary schemas and indicates the URL where the services may be accessed. The URL for each web service will use the HTTPS protocol to ensure the privacy of all communication between ROS and the web service client.

2. Services specifications

The SOAP web services are described through WSDL files and the schema for each message.

Further details of the web services can be found in the published Customs and Excise WSDL files.

The WSDL, SOAP and WS Security version specifications we are following include:

WSDL 2.0: WSDL 2.0

SOAP 1.2: SOAP 1.2

WS Security 1.1.1: WS Security 1.1.1

3. Interpreting the Responses

Each web service will return a response message to the client as outlined below.

3.1. Validation Errors

3.1.1 SOAP Faults

When a request is made to SOAP web service three checks are carried out before any processing can occur. These include:

1. Authentication
2. Authorisation
3. Schema validation

The message is first checked that it is signed with a valid digital signature, the credentials are authorised and then the message is validated against the schema.

If there are any errors encountered carrying out the above processes then a SOAP fault will be returned to the client. The fault string will provide more information on the details of the problem. Where a SOAP fault is returned to the client, no processing will occur for that message.

4. Digital Signatures

Any Revenue Online Services web service request that either returns confidential information or accepts submission of information must be digitally signed. This must be done using a digital certificate that has been previously retrieved from ROS.

The digital signature must be applied to the message in accordance with the WS-Security specification.

The digital signature ensures the integrity of the document. By signing the document, we can ensure that no malicious intruder has altered the document in any way. It can also be used for non-repudiation purposes.

If a valid digital signature is not attached, a SOAP Fault will be returned. The fault string will provide more information on the details of the problem.

4.1. Namespaces

The valid approach for this is using Oasis standards:

- The WS-Security namespace should be: http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd

- The WSU namespace should be: http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd

- All Id references should now be of the form wsu:Id e.g. `<x:myElement wsu:Id=”ID1” xmlns=”…” xmlns:wsu=”…”/>`

- The XML Digital Signature namespace should be: http://www.w3.org/2000/09/xmldsig#

4.2. Security

The security header contains three elements:

1. The Binary Security Token
2. The Timestamp
3. The Signature.

4.2.1 Binary Security Token

The X509 certificate used to sign the message should be included in the message as a Base64 encoded BinarySecurityToken element (`Envelope/Header/Security/BinarySecurityToken`).

The EncodingType attribute of the BinarySecurityToken should have a value of http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soapmessage-security-1.0#Base64Binary.

The ValueType attribute should have a value of http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-tokenprofile-1.0#X509v3.

4.2.2 Timestamp

The timestamp element (`Envelope/Header/Security/Timestamp`) must contain:

1. The Created date
2. The Expired date.

Both dates must conform to the following Oasis standard: https://docs.oasis-open.org/wss/v1.1/wss-v1.1-spec-errata-os-SOAPMessageSecurity.htm#_Toc118717167 and the expired date must be after the created date.

For SOAP requests the maximum time between timestamp creation and expiration is 60 seconds.

4.2.3 Signature

This is the signature that is calculated using your ROS digital certificates private key. The signature contains three elements:

1. The SignedInfo element
2. The SignatureValue element
3. The KeyInfo element.

4.2.3.1 SignedInfo

The SignedInfo element contains a CanonicalizationMethod, SignatureMethod and two Reference elements.

XML Canonicalization is used to format the XML before calculating the digest values. The SHA512 algorithm is used for signing the message.

Canonicalization: The Canonicalization Algorithm should be XML-EXC-C14N (Exclusive Canonicalization) - Canonicalization Algorithm

Signature Algorithm: The Signature Algorithm should be SHA512withRSA - http://www.w3.org/2001/04/xmldsig-more#rsa-sha512

This type of message is known as a detached signature.

Oasis Standards References: There must be two Reference elements (`Envelope/Header/Security/Signature/SignedInfo/Reference`) in the SignedInfo element.

- One Reference element should correspond to the signed Body element within the message. The Reference corresponding to the Body should have an id attribute whose value is the same as the Id attribute of the SOAP `<body>` element.
- The other Reference corresponds to the Timestamp. The Reference corresponding to the Timestamp should have an id attribute whose value is the same as the Id attribute of the SOAP `<Timestamp>` element.

Both References should have a single transform – Exclusive Canonicalization (see the URI above). The Digest Algorithm should be SHA512 - http://www.w3.org/2001/04/xmlenc#sha512.

4.2.3.2 SignatureValue

For the SignatureValue (`Envelope/Header/Security/Signature/SignatureValue`) extract the private key from the .p12 file. Please see Appendix A – Extracting from a .p12 File for instructions on how to do this.

4.2.3.3 KeyInfo

The KeyInfo contains a SecurityTokenReference element which contains a Reference corresponding to the BinarySecurityToken.
(`Envelope/Header/Security/Signature/KeyInfo/SecurityTokenReference/Reference`).

The `URI` attribute of the Reference should reference the Id attribute of the BinarySecurityToken element. For example, if the Id attribute of the BinarySecurityToken is “X509Token”, the URI attribute of the Reference sub-element should be “#X509Token”.

Appendix A – Extracting from a .p12 File

Each customer of ROS will have a digital certificate and private key stored in an industry standard PKCS#12 file.

In order to create a digital signature, the private key of the customer must be accessed. A password is required to retrieve the private key from the P12 file. This password can be obtained by prompting the user for their password.

The password on the P12 is not the same as the password entered by the customer. It is in fact the MD5 hash of that password, followed by the Base64-encoding of the resultant bytes.

To calculate the hashed password, follow these steps:

1. First get the bytes of the original password, assuming a "Latin-1" encoding. For the password "Baltimore1," these bytes are: 66 97 108 116 105 109 111 114 101 49 44 (i.e. the value of "B" is 66, "a" is 97, etc.).
2. Then get the MD5 hash of these bytes. MD5 is a standard, public algorithm. Once again, for the password "Baltimore1," these bytes work out as: 223 238 161 24 62 121 39 143 115 167 51 163 245 231 226 94.
3. Finally, create the new password by Base64-encoding the bytes from the previous step. For example, the password, "Baltimore1," this is “3+6hGD55J49zpzOj9efiXg==”.

This new password can then be used to open a standard ROS P12 file.