Revenue
Cáin agus Custaim na hÉireann
Irish Tax and Customs

Customs & Excise

Customs & Excise REST Web Service Integration Guide

The information in this document is provided as a guide only and is not professional advice, including legal advice. It should not be assumed that the guidance is comprehensive or that it provides a definitive answer in every case.

Contents

Version Control		
Version	**Date**	**Change**
0.1	04/10/2019	Initial document published
0.2	28/01/2022	Updated reference to AES
0.3	22/07/2025	Updated to reflect changes introduced in 06/09/2025

Document References

Reference
1. *"Customs & Excise Web Services Common Specification"*
2. *"Web Services for C&E Enquiries"*
3. *"Customs & Excise Web Services for Payer and Importer Reports"*
4. Full list of ROS errors.

Abbreviations Used in This Document

Abbreviation	Description
AES	Automated Export System
AIS	Automated Import System
C&E	Customs and Excise
EDE	Excise Duty Entry
EMCS	Excise Movement Control System
EORI	Economic Operators Registration and Identification
ERV	Export Release Verification
NCTS	New Computerised Transit System
PIT	Public Interface Testing
RORO	Roll-On Roll-Off system
ROS	Revenue Online Service
UCC	Union Customs Code

Audience

This document is for any software provider who has chosen to build or update their products to consume REST web services provided by Revenue Customs and Excise division.

1 Introduction

This document provides a technical overview of how to integrate with Revenue Customs and Excise REST web services, both JSON and XML, including how to sign requests validly.

It is applicable to the following REST web services:

- Handshake
- AIS Submit
- AES Submit
- NCTS (P5) Submit
- EDE Submit
- EMCS Submit
- Mailbox Collect
- Mailbox Acknowledge
- Transaction ID
- Balance Enquiry
- Exchange Rate Enquiry
- Export Release Verification
- RORO
- C&E Transactions Reports

This document is designed to be read in conjunction with the REST/JSON/XML example files as well as the Documents specified in Document References section, which detail the technical documentation, specification, and examples for the above web services.

This document assumes familiarity with the REST web services.

2 Calling the Services

Calls to Customs and Excise web services MUST be digitally signed using a digital certificate that has been previously retrieved from ROS. The signature MUST conform to modified version of HTTP Signatures Protocol specification. Detailed description can be found in the Digital Signatures section below.

2.1 C&E REST Endpoints

The C&E web services provide the following endpoints (Live):

Description	Http Method	Endpoint URL	Req. Payload Type	Resp. Payload Type	Additional Information
Handshake	GET POST	https://www.ros.ie/customs/webservice/v1/rest/handshake	XML JSON	XML JSON	General handshake to verify connectivity and correctness of digital signature.
AIS Submit	POST	https://www.ros.ie/customs/webservice/v1/rest/aisSubmit	XML	XML	Automated Import System
AES Submit	POST	https://www.ros.ie/customs/webservice/v1/rest/aesSubmit	XML	XML	Automated Export System
NCTS (P5, P6) Submit	POST	https://www.ros.ie/customs/webservice/v1/rest/nctssSubmit	XML	XML	New Computerised Transit System - Phase 5.
EDE Submit	POST	https://www.ros.ie/customs/webservice/v1/rest/edeSubmit	XML	XML	Excise Duty Entry.
EMCS Submit	POST	https://www.ros.ie/customs/webservice/v1/rest/emcsSubmit	XML	XML	Excise Movement Control System.
Mailbox Collect	POST	https://www.ros.ie/customs/webservice/v1/rest/mailboxCollect	XML	XML	Collect Customs and Excise response. More details available in *"Customs & Excise Web Services Common Specification"* document.

Mailbox Acknowledge	POST	https://www.ros.ie/customs/webservice/v1/rest/mailboxAcknowledge	XML	XML	Acknowledge the collection of Customs and Excise responses. More details available in *"Customs & Excise Web Services Common Specification"* document.
Transaction ID (reliable messaging)	POST	https://www.ros.ie/customs/webservice/v1/rest/transactionID	XML	XML	Obtain tokens for Reliable Messaging. More details available in *"Customs & Excise Web Services Common Specification"* document.
Balance Enquiry	GET	https://www.ros.ie/customs/webservice/v1/rest/balance/enquiry/{accountholder}	N/A	JSON	Path variable: • accountholder – EORI number of the account queried. More details available in *"Web Services for C&E Enquiries"* document.
Exchange Rate Enquiry	GET	https://www.ros.ie/customs/webservice/v1/rest/exchange-rate/enquiry/{date}	N/A	JSON	Path variable: date - the year and month in the form yyyyMM for which the exchange rates are being requested More details available in *"Web Services for C&E Enquiries"* document.
Export Release Verification	POST	https://www.ros.ie/customs/webservice/v1/rest/export/releaseVerification	XML	XML	Submit ERV requests.
RORO	GET POST PUT	https://www.ros.ie/customs/webservice/v1/rest/roro-control/ . . .	N/A JSON	JSON	
C&E Payer and Importer Reports	GET	https://www.ros.ie/customs/webservice/v1/rest/transactions/ . . .	N/A	JSON	Details on each individual report endpoint can be found in *"Customs & Excise Web Services for Payer and Importer Reports"* document.

3 Interpreting the Responses

Each web service returns a response message to the client. The type of the response depends on:

- Functional nature of the web service (e.g. AIS Submit vs Mailbox Collect)
- Type of request/response payload (XML vs JSON)
- Success vs error response
- Type of errors (Authentication/authorisation error vs functional/validation error)

This is outlined in the table below:

Endpoint	Response Payload Type	Success Response	Error Response	Notes
Handshake	JSON	*{"connectionStatus": "SUCCESS"}*	Validation Errors object with ROS Error	
Handshake	XML	Message Acknowledgement with Status SUCCESS	Message Acknowledgement with ROS Error	
AIS, AES, NCTS, EDE, EMCS Submit	XML	Message Acknowledgement containing Transaction ID	Message Acknowledgement with ROS Error	ROS Error is returned for authentication, authorisation and internal service errors. Functional errors in submitted messages are returned as responses in Customs Mailbox
Mailbox Collect	XML	Mailbox Collect Response	Message Acknowledgement with ROS Error	ROS Error is returned for authentication, authorisation and internal service errors. There are no functional errors.
Mailbox Acknowledge	XML	Mailbox Acknowledge Response	Message Acknowledgement with ROS Error	ROS Error is returned for authentication, authorisation and internal service errors. There are no functional errors.
Transaction ID (reliable messaging)	XML	Transaction ID Response	Message Acknowledgement with ROS Error	Returned for authentication, authorisation and internal service errors.
Transaction ID (reliable messaging)	XML	Transaction ID Response	Transaction ID Response with functional error	Returned for functional errors.
Export Release Verification	XML	Export Release Verification Response	Message Acknowledgement with ROS Error	Returned for authentication, authorisation and internal service errors.

Export Release Verification	XML	Export Release Verification Response	Export Release Verification Response	Returned for functional errors.
Balance Enquiry	JSON	Relevant functional response object	Validation Errors object with ROS Error or functional error	ROS Error is returned for authentication, authorisation and internal service errors.
Exchange Rate Enquiry	JSON	Relevant functional response object	Validation Errors object with ROS Error	ROS Error is returned for authentication, authorisation and internal service errors. There are no functional errors.
C&E Transactions Reports	JSON	Relevant functional response object	Validation Errors object with ROS Error	ROS Error is returned for authentication, authorisation and internal service errors. There are no functional errors.
RORO	JSON	Relevant functional response object	Validation Errors object	Returned for all errors. Authentication, authorisation and internal service errors contain a ROS error. Functional errors contain relevant functional error message.

More detailed description of success and functional error responses are available in the documentation for the relevant system.

ROS Error codes possible in REST responses are presented in the table below.

Error Code	Description
ROS-300-02	Issue with requests media type.
ROS-300-10	Issue with the request's timestamp.
ROS-300-20	Issue with request's digital signature.
ROS-300-30	Issue with request's digest.
ROS-300-50	Certificate holder does not have permissions to submit this request.
ROS-100-00	Unrecognised digital certificate used.
ROS-100-10	Digital certificate used to sign the request is expired.
ROS-100-20	Digital certificate used to sign the request is revoked.
ROS-100-30	Digital certificate used to sign the request is invalid.
FRQ-100-10	Request submitted too soon after the previous one.
REL-100-10	Transaction ID request was invalid.
ROS-300-00	Unexpected error in processing the request. Please try again later.

4 Digital Signatures

Any ROS 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 HTTP Signatures specification as specified in Section 4.1 below.

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 HTTP status code of 401 (Unauthorised) will be returned. The message body will provide more information on the details of the problem.

4.1 HTTP Signatures

The HTTP signatures protocol is intended to provide a simple and standard way for clients to sign HTTP requests. A summary of the structure of a HTTP Signature is outlined below. This is a simplified explanation of the HTTP Signatures specification. The full specification can be found at Signing HTTP Messages and should be read in full. The specification defines two approaches to building a HTTP signature, "The 'Signature' HTTP Authentication Scheme" and "The 'Signature' HTTP Header", Revenue uses the latter.

At a high level, a HTTP Signature is a HTTP header that is added to a HTTP request. It is comprised of a set of components that were used to generate a digital signature and the digital signature itself.

4.1.1 HTTP Signature Sample

Below is a sample HTTP Signature header.

```
Signature: keyId="MIICfzCCAeigAwIBAgIJ... // truncated",
           algorithm="rsa-sha512",
           headers="(request-target) host date digest",
           signature="GdUqDgy94Z8mSYUjr/rL6qrLX/jmudS... // truncated"
```

4.1.2 HTTP Signature Components

The Signature HTTP header contains four components: ***keyId, algorithm, headers*** and ***signature***. Below is a description of each.

keyId	MUST contain a Base64 encoded version of the X509 certificate that accompanies the private key used to sign the message.
algorithm	Specifies the digital signature algorithm used when generating the signature. Revenue expects this to be '***rsa-sha512***'.
headers	Specifies the list of headers used when generating the signature for the message. This MUST be a lowercased, quoted list of HTTP header fields, separated by a single space character. The list order is important and MUST be specified in the order the HTTP header field-value pairs are concatenated together to create the `***signing string***` (see above). For POST and PUT requests, the digest field MUST be included.
signature	Digital signature, generated by: 1) Signing the `***signing string***` with the private key that accompanies the X509 certificate associated with the `***keyId***` field and the algorithm corresponding to the `***algorithm***` field. 2) Base 64 encoding the signature obtained as in 1) above.

4.1.3 Signature String Construction

The string to be signed is composed of three (GET) or four (POST, PUT) variables, representing HTTP Headers and their related values, separated by **colon ':'** ,an **ASCI space** and a **line break '\n' at the end of each**. These MUST use the values of each HTTP header defined in the `headers` signature field and MUST be in the order they appear in the `headers` signature field and MUST be provided before the "signature" header. If the associated HTTP header does not exist, it should be added to the HTTP request BEFORE attempting to construct this string.

Allowable values in the headers field are outlined in the table below.

Value	Mandatory
* (request-target)	Yes
Host	Yes
Date	Yes, this is the recommend way to provide the request date.
** x-date	Yes, if date header cannot be added.

*** digest	Yes, if HTTP method is of type POST or PUT
content-type	No
content-length	No

* The `(request-target)` header field is a special header field in that its value is comprised of 2 HTTP headers. It is generated by concatenating the lowercase HTTP method, an ASCII space, and the request path headers.

** The 'x-date' headers field value should ONLY be used in conjunction with the X-Date HTTP header if a Date HTTP header cannot be added to the HTTP request programmatically. The Date header has a limitation when using JavaScript in a browser to build and send a HTTP signature. The limitation is that you cannot add a 'Date' HTTP header when executing JavaScript in a browser. The native XmlHttpRequest object prohibits addition of a 'Date' HTTP header. Building the signature string that will be signed with an 'x-date' header instead of a 'date' header removes this restriction.

*** The 'Digest' HTTP header is created using the POST/PUT body/payload. The payload should be converted to a byte array, hashed using the SHA-512 algorithm and finally base64 encoded before adding it as a HTTP header. It is not required for GET requests (as there is no body/payload to create the digest from). The digest value MUST be without any prefixes.

All other header field values are created by concatenating the lowercase header field name, followed by an ASCII colon `:`, an ASCII space ` `, the header field value (leading and trailing whitespace in the header field value MUST be omitted), and an ASCII newline `\n` if the header field is not the last one as defined in the `headers` signature field.

Example:

Assuming that the HTTP headers are sent in the following order (host, date, digest, signature), the four variables are: *(request-target) host date digest*

At the end the string value should look like so:

Signature String to be signed :

(request-target): post /customs/webservice/v1/rest/transactionID
host: softwaretestnextversion.ros.ie
date: 2020-05-22T16:19:37.697Z
digest: aTjNufDtv6U+DrL6CfpF1EMgjqic31fBeV3eU9QaC1PeOCzhpxuFYK6FxUErHQcPEL2HkOKxrpcS9cLN5u222w==

** Please note that there's a line break '\n' at the end of each row, except for the last row.
*** Note no algorithm prefix in front of the digest value.

Notes on Date header:

The GMT time zone is enforced. Requests expire after 90 minutes has elapsed based on the timestamp received on the date/x-date header. Requests made up to 90 minutes before the revenue server time are also accepted to counteract the potential for server time discrepancies and daylight savings. For example:

A message with the timestamp in ISO_8601 ("yyyy-MM-dd'T'HH:mm:ss.SSSX") format is received as follows:

2018-01-01T12:00:00.000Z

This message is valid for 90 minutes either side of it. i.e.

from 2018-01-01T10:30:00.000Z to 2018-01-01T13:30:00.000Z

Values for days and months should be two digits. As such, in the case where either the day or the month is a single digit, such as 2018-1-1, the single digit value should be prepended with a zero to make it 2018-01-01.

The accepted formats for date are:

- ISO 8601
- RFC 822, updated by RFC 1123
- RFC 850, obsoleted by RFC 1036
- ANSI C's time format

Please note that even either 'x-date' or 'date' headers are allowed, the 'date' is what we recommend, if supported by implementer's framework.

4.1.4 Signature Creation

The signature component is a base 64 encoded digital signature. The implementer uses the `algorithm` and Signature String constructed as described above.

The Signature String is signed using the algorithm corresponding to the `algorithm` field with the private key that accompanies the X509 certificate extracted from .p12 file associated with the `keyId` field (see Appendix A below on how to retrieve value form the .p12 file)

The `signature` field is then base 64 encoded.

Example:

At this point all the necessary data should be available to create the 'signature' HTTP header. The value becomes the sequence of 'keyId', 'algorithm', 'headers' and 'signature'.

- keyId – the X509 certificate as byte array base64 encoded
- algorithm – rsa-sha512
- headers – (request-target) host date digest
- signature – the base64 encoded value of the Canonicalized String signed with the private key

All the fields above are then concatenated and separated by a comma. The value should look like below:

keyId="MIIEIjC...",algorithm="rsa-sha512",headers="(request-target) host date digest",signature="So7R..."

4.1.5 Testing HTTP Signature with curl

Below is an example of curl request to PIT environment to test HTTP signature creation:

```
curl --location --request POST
'https://softwaretestnextversion.ros.ie/customs/webservice/v1/rest/transactionID' \
--header 'host: softwaretestnextversion.ros.ie' \
--header 'date: 2020-05-22T15:47:28.564Z' \
--header 'digest: aTjNufDtv6U+DrL6CfpF1EMgjqic31fBeV3eU9QaC1PeOCzhpxuF ...(truncated)' \
--header 'signature: keyId="MIIEIjCCAwqgAwIBAgIRANv90rCYLJBunAklsCHppNo ...(truncated)",
algorithm="rsa-sha512",headers="(request-target) host date digest",
signature="So7RJP9MdCLHPgT9Br3SMCu2iOT+5NIulUNzCYUVBHbDfa4BNnGv/F8Qt7CuZ...(truncated)"' \
--header 'Content-Type: application/xml' \
--data-raw '<v1:TransactionIDRequest
xmlns:v1="http://www.ros.ie/schemas/customs/transactionidrequest/v1"><v1:Transactions><v1:Numb
erOfTxIds>1</v1:NumberOfTxIds></v1:Transactions></v1:TransactionIDRequest>'
```

Note no algorithm prefix in front of the digest value.

Please note, as per above curl command, the order of the HTTP headers: host, date, and digest, is the same as in the parameter "headers" inside the HTTP header 'signature'.

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 "Password123", these bytes are: 80 97 115 115 119 111 114 100 49 50 51(i.e. the value of "P" is 80, "a" is 97, etc.).
2. Then get the MD5 hash of these bytes. MD5 is a standard, public algorithm. Once again, for the password "Password123" these bytes work out as: 66 -9 73 -83 -25 -7 -31 -107 -65 71 95 55 -92 76 -81 -53.
3. Finally, create the new password by Base64-encoding the bytes from the previous step. For example, the password, "Password123" this is “QvdJref54ZW/R183pEyvyw==”.

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