ICT&L REVENUE

Revenue
Cáin agus Custaim na hÉireann
Irish Tax and Customs

PAYE Modernisation

REST Connectivity Handshake Guide

Contents

Version	**1.0**
Version Date	**30/10/2018**

Version History			
Version	**Change Date**	**Section**	**Change Description**
V 1.0	30/10/2018	All	Document published.

Audience

This document is for any software provider who has chosen to build or update their products to allow for PAYE Modernisation.

Document context

This document provides a technical overview of how to integrate with Revenue's REST web services including how to sign requests validly. This document is designed to be read in conjunction with the REST/JSON example files as well as the rest of the Revenue Commissioners' PAYE Modernisation documentation suite including the relevant technical documents.

Document References	
Reference	**Document Link**
1. Documents Homepage	Documents Homepage

1. Introduction

This document details the REST PAYE Modernisation web services specification for the following web services:

- REST Connectivity Handshake

The Documents Homepage specified in Document References is the home to all technical documentation, specification, and examples for the above web services which has been made available to enable payroll software developers to update their software packages to be compatible with PAYE reporting obligations from January 2019. Path locations specified in this document are relative to this Homepage.

This document assumes familiarity with the REST web services above. A full description of each of these can be found in the 'PAYE Modernisation Description of Web Service Examples' Document under 'PAYE Web Service Examples' on the Documents Homepage.

The OpenAPI Specification, originally known as the Swagger Specification, is a specification for machine-readable interface files for describing, producing, consuming, and visualizing RESTful Web services. The specification can be described using the YAML or JSON format. Revenue provides the specification in JSON format or, alternatively, it can be viewed using the ReDoc UI framework. The JSON file can be downloaded and imported into any Open API UI framework such as the Swagger UI framework. Links are provided for both the JSON format file and the ReDoc UI framework under 'PAYE Web Service Specifications (REST/JSON)' on the Documents Homepage. 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. Calling the Services

The web services for the PAYE Modernisation messages are described in the REST OpenAPI Specification JSON file provided under 'PAYE Web Service Specifications (REST/JSON)' on the Documents Homepage.

The OpenAPI, REST Security version specifications we are following include:

Open API Specification Version 3.0: https://github.com/OAI/OpenAPI-Specification/blob/master/versions/3.0.0.md

HTTP Signatures Version 08: https://tools.ietf.org/html/draft-cavage-http-signatures-08

2.1. REST Endpoints

The PAYE Modernisation web service endpoints are detailed below.

Description	HTTP Method	Endpoint URL	Additional Information	Links

REST Connectivity Handshake	GET	https://www.ros.ie/paye-employers/v1/rest/handshake?employerRegistrationNumber={employerRegistrationNumber}&softwareUsed={softwareUsed}&softwareVersion={softwareVersion}&agentTain={agentTain}	Query Parameters • softwareUsed • softwareVersion • employerRegistrationNumber(optional) • agentTain (optional)	

2.2. Digital Signatures

The PAYE Modernisation web services will require a digital signature. This will be the digital signature of the declarant.

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. Request Validation

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

1. Authentication
2. Authorisation
3. JSON API validation

Step 1 of the validation process tries to verify the authenticity of the message by checking it has been signed with a valid digital signature. Step 2 focuses on authorising the credentials and finally, step 3 checks the message is valid using JSON API validation.

Please note that Step 2 will only be performed if an Employer's PAYE tax registration number is provided as part of the request or if an Agent TAIN and Employer's PAYE tax registration number is provided as part of the request.

If there are any errors encountered during the above processes, the response will include a HTTP status code of either 401 (Unauthorised) if authentication fails, 403 if authorisation fails and 400 (Bad Request) if JSON API validation fails. The message body will provide more information on the details of the problem. Where an error code is returned to the client, no processing will occur for that message.

If the message passes JSON API validation, authentication, and authorisation but the resource cannot be found the response will include a HTTP status code of 404. If the resources query parameters are not as required or are missing, the response will include a HTTP status code of 400. Where an error code is returned to the client, no processing will occur for that message.

3.2. Successful Response

The "Connectivity" web service will return a HTTP 200 OK response if called with a validly signed request and neither of the two optional parameters (employerRegistrationNumber, agentTain) have been included in the request.

If the request includes an Employer's PAYE tax registration number (employerRegistrationNumber), a HTTP 200 Ok response will be returned if the request is validly signed and the provided tax registration number is the owner of the certificate that signed the request.

If the request includes both an Employer's PAYE tax registration number (employerRegistrationNumber) and Agent TAIN (agentTain), a HTTP 200 Ok response will be returned if the request is validly signed, the provided Agent TAIN is the owner of the certificate and the provided tax registration number is linked to the Agent TAIN. Please note that if an Agent TAIN is provided a linked Employer's PAYE tax registration number must also be provided.

A list of validation errors (if any) on the request is also included in the response. Please refer back to Section 3.1 for more information on Validation Errors.

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.

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: The keyId field must contain a Base64 encoded version of the X509 certificate that accompanies the private key used to sign the message. This field is required.

algorithm: The `algorithm` parameter is used to specify the digital signature algorithm to use when generating the signature. Revenue expects this to be 'rsa-sha512'. This field is required.

headers: The `headers` parameter specifies the list of headers used when generating the signature for the message. The parameter 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 during signing.

signature: The signature component is a base 64 encoded digital signature. The implementer uses the `algorithm` and `headers` field to form a canonicalized `signing string`. This `signing string` is then signed with the private key that accompanies the X509 certificate associated with the `keyId` field and the algorithm corresponding to the `algorithm` field. The `signature` field is then base 64 encoded.

4.1.3. Signature String Construction

In order to generate the string to be signed, the implementer MUST use the values of each HTTP header defined in the `headers` signature field, to build the signature string. Values must be in the order they appear in the `headers` signature field. If the associated HTTP header does not exist, it should be added to the HTTP request BEFORE attempting to construct this string.

Date:

The GMT time zone is enforced. Requests expire after 60 seconds has elapsed based on the timestamp received on the date/x-date header. Requests made up to 60 seconds before the revenue server time are also accepted to counteract the potential for server time discrepancies. 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 60 seconds either side of it.

i.e. from 2018-01-01T11:59:00.000Z to 2018-01-01T12:01:00.000Z

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 1/1/2018, the single digit value should be prepended with a zero to make it 01/01/2018.

- 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

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

Value	Mandatory
* (request-target)	Yes
host	Yes
date	Yes
** x-date	Yes, if date header cannot be added.
*** digest	Yes, if HTTP method is of type POST
**** content-type	No
content-length	No
x-http-method-override	If HTTP method is of type POST, HTTP header *'X-HTTP-Method-Override'* exists and *'Content-Type=application/x-www-form-urlencoded'*. See Section 2.1.1 for more detail.

* 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. See below for sample

(request-target): get paye-employers /v1/rest/handshake?
employerRegistrationNumber=1234567FA&softwareUsed=ACME&softwareVersion=1.0

** 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 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.

**** Although Content-Type does not have to be part of the Signature string, it is required as a HTTP Header if HTTP Method Type is POST. If the request is a standard POST (i.e. not a header *'X-HTTP-Method-Override'*POST, then the Content-Type should be set to 'application/json' or 'application/json;charset=UTF-8'

All other header field values are created by concatenating the lowercase header field name followed by an ASCII colon `:`, an ASCII space ` `, and the header field value. Leading and trailing whitespace in the header field value MUST be omitted. If the header field is not the last value defined in the `headers` signature field, then append an ASCII newline `\n`

4.1.4. Signature Creation

The signature component is a base 64 encoded digital signature. The implementer uses the `algorithm` and constructed Signature String. The Signature String is signed with the private key that accompanies the X509 certificate associated with the `keyId` field and the algorithm corresponding to the `algorithm` field. The `signature` field is then base 64 encoded.

5. Example Messages

Message:

https://www.ros.ie/paye-employers /v1/rest/handshake?
softwareUsed=ACME&softwareVersion=1.0

Expected Response:

HTTP 200 OK

Message:

https://www.ros.ie/paye-employers /v1/rest/handshake?
employerRegistrationNumber=1234567FA&softwareUsed=ACME&softwareVersion=1.0

Expected Response:

HTTP 200 OK

Message:

https://www.ros.ie/paye-
employers/v1/rest/handshake?employerRegistrationNumber=1234567FA&softwareUsed=Acme&sof
twareVersion=1.0&agentTain=123456J

Expected Response:

HTTP 200 OK