Informatica®
OAuth2

OAuth2 Authentication Guide for Data as a Service

Informatica OAuth2 Authentication Guide for Data as a Service
OAuth2

NOTICES

This Informatica product (the "Software") includes certain drivers (the "DataDirect Drivers") from DataDirect Technologies, an operating company of Progress Software Corporation ("DataDirect") which are subject to the following terms and conditions:

1. THE DATADIRECT DRIVERS ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT.
2. IN NO EVENT WILL DATADIRECT OR ITS THIRD PARTY SUPPLIERS BE LIABLE TO THE END-USER CUSTOMER FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, CONSEQUENTIAL OR OTHER DAMAGES ARISING OUT OF THE USE OF THE ODBC DRIVERS, WHETHER OR NOT INFORMED OF THE POSSIBILITIES OF DAMAGES IN ADVANCE. THESE LIMITATIONS APPLY TO ALL CAUSES OF ACTION, INCLUDING, WITHOUT LIMITATION, BREACH OF CONTRACT, BREACH OF WARRANTY, NEGLIGENCE, STRICT LIABILITY, MISREPRESENTATION AND OTHER TORTS.

The information in this documentation is subject to change without notice. If you find any problems in this documentation, report them to us at infa_documentation@informatica.com.

Informatica products are warranted according to the terms and conditions of the agreements under which they are provided. INFORMATICA PROVIDES THE INFORMATION IN THIS DOCUMENT "AS IS" WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING WITHOUT ANY WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND ANY WARRANTY OR CONDITION OF NON-INFRINGEMENT.

Publication Date: 2023-09-28

Table of Contents

Preface

The *OAuth2 User Guide* provides information about the OAuth2 authorization protocol that various applications use. This guide also describes how to use different methods to obtain an access token.

Informatica Resources

Informatica provides you with a range of product resources through the Informatica Network and other online portals. Use the resources to get the most from your Informatica products and solutions and to learn from other Informatica users and subject matter experts.

Informatica Network

The Informatica Network is the gateway to many resources, including the Informatica Knowledge Base and Informatica Global Customer Support. To enter the Informatica Network, visit https://network.informatica.com.

As an Informatica Network member, you have the following options:

- Search the Knowledge Base for product resources.
- View product availability information.
- Create and review your support cases.
- Find your local Informatica User Group Network and collaborate with your peers.

Informatica Knowledge Base

Use the Informatica Knowledge Base to find product resources such as how-to articles, best practices, video tutorials, and answers to frequently asked questions.

To search the Knowledge Base, visit https://search.informatica.com. If you have questions, comments, or ideas about the Knowledge Base, contact the Informatica Knowledge Base team at KB_Feedback@informatica.com.

Informatica Documentation

Use the Informatica Documentation Portal to explore an extensive library of documentation for current and recent product releases. To explore the Documentation Portal, visit https://docs.informatica.com.

Informatica maintains documentation for many products on the Informatica Knowledge Base in addition to the Documentation Portal. If you cannot find documentation for your product or product version on the Documentation Portal, search the Knowledge Base at https://search.informatica.com.

If you have questions, comments, or ideas about the product documentation, contact the Informatica Documentation team at infa_documentation@informatica.com.

Informatica Product Availability Matrices

Product Availability Matrices (PAMs) indicate the versions of the operating systems, databases, and types of data sources and targets that a product release supports. You can browse the Informatica PAMs at https://network.informatica.com/community/informatica-network/product-availability-matrices.

Informatica Velocity

Informatica Velocity is a collection of tips and best practices developed by Informatica Professional Services and based on real-world experiences from hundreds of data management projects. Informatica Velocity represents the collective knowledge of Informatica consultants who work with organizations around the world to plan, develop, deploy, and maintain successful data management solutions.

You can find Informatica Velocity resources at http://velocity.informatica.com. If you have questions, comments, or ideas about Informatica Velocity, contact Informatica Professional Services at ips@informatica.com.

Informatica Marketplace

The Informatica Marketplace is a forum where you can find solutions that extend and enhance your Informatica implementations. Leverage any of the hundreds of solutions from Informatica developers and partners on the Marketplace to improve your productivity and speed up time to implementation on your projects. You can find the Informatica Marketplace at https://marketplace.informatica.com.

Informatica Global Customer Support

You can contact a Global Support Center by telephone or through the Informatica Network.

To find your local Informatica Global Customer Support telephone number, visit the Informatica website at the following link:
https://www.informatica.com/services-and-training/customer-success-services/contact-us.html.

To find online support resources on the Informatica Network, visit https://network.informatica.com and select the eSupport option.

CHAPTER 1

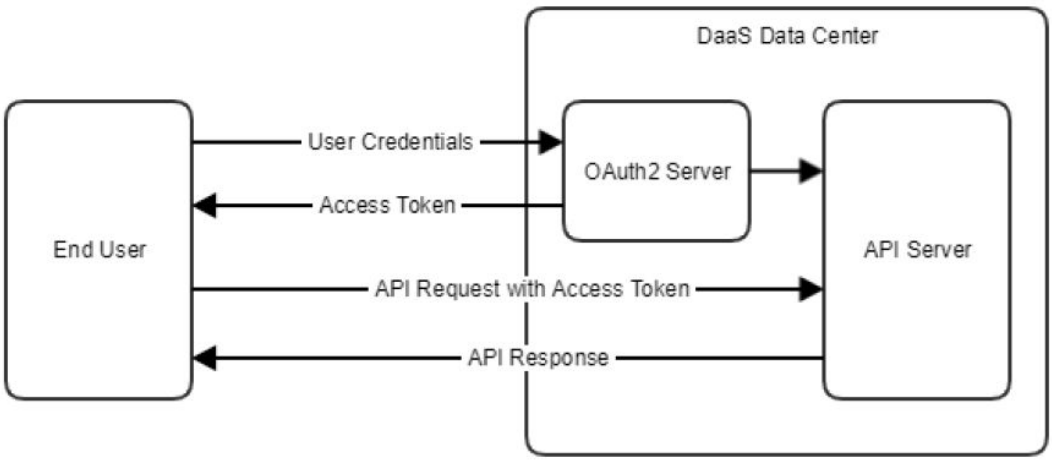
OAuth2 Authentication

This chapter includes the following topics:

Overview

OAuth2 is a common authentication method used by many web sites and applications today. Informatica Data as a Service products such as Address Verification, Email Verification, and Global Phone Validation use OAuth2 URLs. An end user provides their credentials to get an access token, which is used for authorization from then on. The token will eventually expire and no longer be valid.

For Informatica DaaS products instead of using a user name / password or license key on every API call, the end user would use their credentials in a call to the OAuth2 server to retrieve an access token. The user then provides that token on API calls until it expires, in which case they must call the OAuth2 server again to get a new one.



OAuth2 URLs

Informatica DaaS maintains four OAuth2 servers - two in the United States and two in Europe. Their functionality is the same, and a customer may use either one. Note that the use of HTTPS is required.

United States URLs

Use the following URLs to connect to the OAuth2 interface:

REST endpoints:

- https://oauth2.strikeiron.com/oauth2/v2/token
- https://oauth2.strikeiron.com/oauth2/v2/verify

Swagger documentation: https://oauth2.strikeiron.com/swagger/v2/swagger.json

Europe URLs

Use the following URLs to connect to the OAuth2 interface:

REST endpoints:

- https://eu-oauth2.informaticadaas.com/oauth2/v2/token
- https://eu-oauth2.informaticadaas.com/oauth2/v2/verify

Swagger documentation: https://eu-oauth2.informaticadaas.com/swagger/v2/swagger.json

Selecting an Endpoint

Use an **oauth2/v2/token/av** endpoint to retrieve a token for Address Verification.

Use an **oauth2/v2/token/si** endpoint to retrieve a token for other DaaS products.

Use an **oauth2/v2/verify** endpoint to submit a job request.

Requesting a Token

There are four different ways to request an access token but they all accomplish the same thing. An example of each is given below, and the end user is free to use whichever one works best for them. Note that the credentials in these examples are not valid and the user should substitute their own valid credentials for the calls to work.

There are two different ways to authenticate when requesting a token. There is the standard **user name / password** combination, or alternatively the **user name / license key** combination. The license key option is typically used if you have purchased more than one license for the same product and need to specify which one to use. For example, some customers have a single account with one Email Verification license key for their Marketing department and another for their Sales department. If that customer authenticated with a user name / password, the DaaS system has to guess which Email Verification license to use for the transaction. By providing the license key there is no question of which license to deduct transactions from.

There are examples of both authentication methods included below. Please note this difference in authorization between the two methods when using OAuth2:

- When authenticating with a user name / password combination, the resulting token is valid for any license in that account.
- When authenticating with a user name / license key combination, the resulting token is valid for only that specific license.

GET with Credentials as Query String Parameters

This first method is also the simplest, and you can even test it out in your web browser. Construct a URL in this format to retrieve an access token, replacing the bold text with your credentials:

https://oauth2.strikeiron.com/oauth2/v2/token?client_id=[user name]&client_secret=[password or license key]&grant_type=client_credentials

For example, if your user name is **user@example.com** and your password is **auth123** then the call would look like this:

https://oauth2.strikeiron.com/oauth2/v2/token?
client_id=user@example.com&client_secret=auth123&grant_type=client_credentials

Alternatively, if you wanted specify license key **6BC4029C9A94FBC5581F** for this account the call would look like this:

https://oauth2.strikeiron.com/oauth2/v2/token?
client_id=user@example.com&client_secret=6BC4029C9A94FBC5581F&grant_type=client_credentials

A complete HTTP request would appear like this:

```
GET /oauth2/v2/token?
client_id=user@example.com&client_secret=auth123&grant_type=client_credentials HTTP/
1.1
Host: oauth2.strikeiron.com
```

Requesting a Token

Use an **oauth2/v2/token/av** endpoint to request a token for Address Verification.

Use an **oauth2/v2/token/si** endpoint to request a token for other DaaS products.

URL Encoding

When using this particular method, if your user name or password contain any HTTP reserved characters then you will need to URL encode them. This is easily done programmatically or by using a free resource such as www.urlencoder.org. If your credentials contain white space or any of these characters you will need to encode them:

```
! * ' ( ) ; : & = + $ , / ? # [ ]
```

For instance, if your password was **auth&123?** that would cause a problem with the request since both & and ? have special meanings in a URL. Once encoded, this password would appear as **auth%26123%3F** which is safe to include in your request:

https://oauth2.strikeiron.com/oauth2/v2/token? client_id=user@example.com&client_secret=auth%26123%3F&grant_type=client_credentials

Note that you should only encode the specific field such as the password, not the entire URL.

Security Concerns

While this is the easiest method to get an access token, it is also the least secure. Passing your credentials as query string parameters in a URL is generally frowned upon by security experts. Since HTTPS is required to request a token your credentials and all other query string parameters are encrypted during transit, but might show up as clear text in web server logs or other places.

GET with Credentials Encoded in the Header

Instead of including your credentials in the URL, you can include them in an HTTP header. This is more secure than including them the URL. The header must be in this format, replacing the bold text with encoded credentials:

```
Authorization: Basic [base64 encoded credentials]
```

To encode credentials, you base64 encode the user name / password combination or the user name / license key combination separated by a colon: **username:password** or **username:licensekey**

For example, using the same user name and password from the previous section the string to encode would be: **user@example.com:auth123**

Similarly, using the same user name and license key from the previous section the string to encode would be: **user@example.com:6BC4029C9A94FBC5581F**

This string may be base64 encoded programmatically or by using a free resource such as www.base64encode.org. Once encoded, the resulting string will look something like this: dXNlckBleGFtcGxlLmNvbTphdXRoMTIz

A complete HTTP header would then appear like this, with the key of Authorization and a value indicating basic authentication with your encoded credentials:

```
Authorization: Basic dXNlckBleGFtcGxlLmNvbTphdXRoMTIz
```

With this header defined, initiate an HTTP GET operation to the token service. Note that the URL must still contain the query string parameter grant_type=client_credentials for a GET operation like this:

https://oauth2.strikeiron.com/oauth2/v2/token?grant_type=client_credentials

The complete HTTP request would look something like this:

http://www.urlencoder.org

http://www.base64encode.org

```
GET /oauth2/v2/token?grant_type=client_credentials HTTP/1.1
Host: oauth2.strikeiron.com
Authorization: Basic dXNlckBleGFtcGxlLmNvbTphdXRoMTIz
```

POST with Credentials Encoded in the Header

You may use the same HTTP **Authorization** header discussed in the previous section, but with an HTTP POST operation rather than a GET. The only difference is that rather than including the **grant_type** parameter as a query string parameter in the URL, you would include it in the body of the message instead. In this case, set the **Content-Type** header to **application/x-www-form-urlencoded**. The HTTP request would look something like this:

```
POST /oauth2/v2/token HTTP/1.1
Host: oauth2.strikeiron.com
Authorization: Basic dXNlckBleGFtcGxlLmNvbTphdXRoMTIz
Content-Type: application/x-www-form-urlencoded
grant_type=client_credentials
```

POST with Credentials in the Body

The final option is to place all the required information in the body of a POST request. This will look similar to the previous option but instead of an Authorization header you put the credentials in as additional form elements. A request using the user name / password combination would look like this:

```
POST /oauth2/v2/token HTTP/1.1
Host: oauth2.strikeiron.com
Content-Type: application/x-www-form-urlencoded
```

```
grant_type=client_credentials
client_id=user@example.com
client_secret=auth123
```

Similarly, using the user name / license key combination would look something like this:

```
POST /oauth2/v2/token HTTP/1.1
Host: oauth2.strikeiron.com
Content-Type: application/x-www-form-urlencoded
grant_type=client_credentials
client_id=user@example.com
client_secret=6BC4029C9A94FBC5581F
```

Token Response

If the credentials are valid, any of the above calls will result in an HTTP 200 response that includes JSON with the token information:

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
{
"access_token": "e66ce8de3d87454eb236211b4005d570",
"token_type": "",
"expires_in": 863999,
"example_parameter": ""
}
```

The returned fields are:

- **access_token**: the access token to be used in API calls
- **token_target**: target can be any product
- **token_type**: will always be blank
- **expires_in**: number of seconds until this token expires - in this case 863999 seconds = approximately 10 days
- **example_parameter**: will always be blank

A failed request will result in an HTTP 400 response that includes JSON with error information:

```
HTTP/1.1 400 Bad Request
Content-Type: application/json;charset=UTF-8
{
"error":"invalid_request"
}
```

The **error** value could be any one of the following:

- **invalid_request**: the client_id or client_secret field is missing
- **invalid_client**: the credentials supplied were invalid
- **invalid_grant**: the grant_type was not included or not set to client_credentials
- **unauthorized_client**: this user is not authorized to request an access token
- **unsupported_grant_type**: the grant_type was set to a valid value but not client_credentials
- **500 Internal Server Error**: an unexpected error occurred on the OAuth2 server

Using a Token

Once you have an access token, there are four different ways to use it when calling an Informatica DaaS API. In all four methods you place the token in the UserID field and leave the Password field blank.

SOAP Request

When using the SOAP protocol, place the access token in the **UserID** input field with no password:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ws="http://
ws.strikeiron.com" xmlns:str="http://www.strikeiron.com/">
<soapenv:Header>
<ws:LicenseInfo>
<ws:RegisteredUser>
<ws:UserID>e66ce8de3d87454eb236211b4005d570</ws:UserID>
<ws:Password/>
</ws:RegisteredUser>
</ws:LicenseInfo>
</soapenv:Header>
<soapenv:Body>
<str:VerifyEmail>
<str:Email>user@domain.com</str:Email>
<str:Timeout>5</str:Timeout>
</str:VerifyEmail>
</soapenv:Body>
</soapenv:Envelope>
```

If the token is invalid, then you will receive an error message that looks something like this:

```
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://
www.w3.org/2001/
XMLSchema" xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
<soap:Header/>
<soap:Body>
<soap:Fault>
<faultcode>soap:Client</faultcode>
<faultstring>accessToken does not exist</faultstring>
<faultactor/>
</soap:Fault>
</soap:Body>
</soap:Envelope>
```

REST Request with Token in the Header

Rather than including the access token in the URL, you can instead include it as an HTTP header. The header must be in this format, replacing the bold text with the token:

```
Authorization: Bearer [token]
```

Unlike the authorization header used when requesting a token, this does not have to be encoded. Using a GET request with the token in the header would look like this:

```
GET /StrikeIron/emv6Hygiene/EMV6Hygiene/VerifyEmail?
VerifyEmail.Email=user@domain.com&VerifyEmail.Timeout=5
HTTP/1.1
Host: ws.strikeiron.com
Authorization: Bearer e66ce8de3d87454eb236211b4005d570
```

While a POST would appear like:

```
POST /StrikeIron/emv6Hygiene/EMV6Hygiene/VerifyEmail HTTP/1.1
Host: ws.strikeiron.com
Authorization: Bearer e66ce8de3d87454eb236211b4005d570
Content-Type: application/x-www-form-urlencoded
VerifyEmail.Email=user@domain.com
VerifyEmail.Timeout=5
```

Remember that you can change the response format from XML to JSON by adding the **Format=JSON** parameter to the GET query string or in the body of the POST:

```
POST /StrikeIron/emv6Hygiene/EMV6Hygiene/VerifyEmail HTTP/1.1
Host: ws.strikeiron.com
Authorization: Bearer e66ce8de3d87454eb236211b4005d570
Content-Type: application/x-www-form-urlencoded
```

```
VerifyEmail.Email=user@domain.com
VerifyEmail.Timeout=5
Format=JSON
```

REST Request with Token in the Body

The final option is to include the access token as another field in the body of a POST request rather than as an HTTP header:

```
POST /StrikeIron/emv6Hygiene/EMV6Hygiene/VerifyEmail HTTP/1.1
Host: ws.strikeiron.com
Content-Type: application/x-www-form-urlencoded
LicenseInfo.RegisteredUser.UserID=e66ce8de3d87454eb236211b4005d570
VerifyEmail.Email=user@domain.com
VerifyEmail.Timeout=5
```

Token Expiration

When an access token expires and you attempt to use it, you will get an **accessToken expired** error. Using SOAP it will appear like this:

```
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://
www.w3.org/2001/
XMLSchema" xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
<soap:Header/>
<soap:Body>
<soap:Fault>
<faultcode>soap:Client</faultcode>
<faultstring>accessToken expired</faultstring>
<faultactor/>
</soap:Fault>
</soap:Body>
</soap:Envelope>
```

While a REST JSON response will look like this:

```
{
"WebServiceResponse": {
"@xmlns": "http://ws.strikeiron.com",
"Error": "accessToken expired"
}
}
```

When this happens you need to request a new token using one of the methods described earlier. Ideally, you will pay attention to the **expires_in** value returned when you first received the token and request a new one before that time runs out. You are free to request a new token at any point, so to avoid receiving errors like the above you should generate a new token shortly before your previous one expires. Note that refresh tokens are not supported.