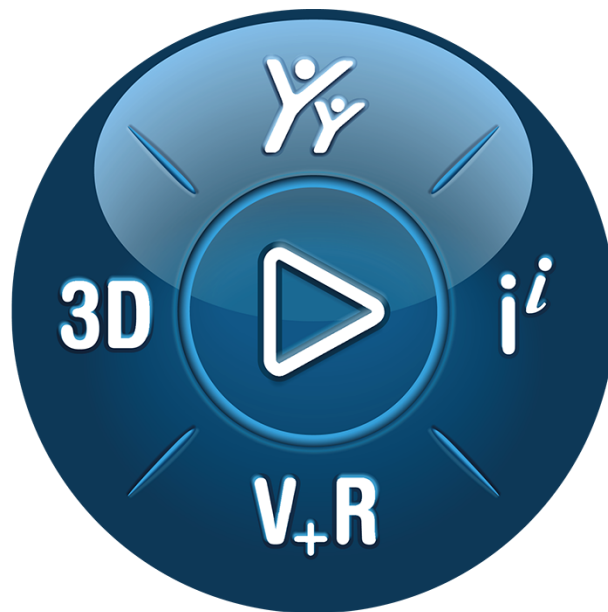


# Web API

## DELMIA Apriso 2021 Technical Guide



# 3DEXPERIENCE®

©2020 Dassault Systèmes. Apriso, 3DEXPERIENCE, the Compass logo and the 3DS logo, CATIA, SOLIDWORKS, ENOVIA, DELMIA, SIMULIA, GEOVIA, EXALEAD, 3D VIA, BIOVIA, NETVIBES, and 3DXCITE are commercial trademarks or registered trademarks of Dassault Systèmes or its subsidiaries in the U.S. and/or other countries. All other trademarks are owned by their respective owners. Use of any Dassault Systèmes or its subsidiaries trademarks is subject to their express written approval.

## Contents

1 Introduction	3
1.1 Prerequisites	3
1.2 Glossary	3
1.3 What is a RESTful API?	4
2 DELMIA Apriso Web API	5
2.1 API Key	5
2.2 Access Token	6
2.2.1 Obtaining an Access Token	6
2.3 Authorization Flows	8
2.4 Adding a Client Application	14
2.5 Example: Calling a Standard Operation	16
2.5.1 Sample Scenario	16
2.5.2 Execution Parameters	20
3 Web API Client	21
3.1 Authorization	21
3.2 Adding a Web Service Provider	22
3.3 Functions	25
3.4 Default and Custom Headers	27
3.5 Debugging	28
3.6 Example: Calling an External Web API	29
4 References	33

# 1 Introduction

DELMIA Apriso exposes a **RESTful API** (see [1.3 What is a RESTful API?](#)) which can be used by a client, such as another instance of DELMIA Apriso or a different external application, to access specific resources. For example, it is possible to publish Standard Operations to be accessible in this manner. In this Technical Guide, this API is referred to as the **DELMIA Apriso Web API**.

In addition to exposing its own web API, DELMIA Apriso is also able to call RESTful APIs made available by other providers using the **Web API Client** functionality. The Web API Client is a JavaScript API which can be used in the HTML Layout Editor's JavaScript tab in Process Builder.

Importantly, DELMIA Apriso implicitly handles **access token**-based and **API key**-based authorization, as well as the **3DPassport** authorization framework. Access token authorization is handled in compliance with the [OAuth 2.0 Authorization Framework](#).

This Technical Guide is divided into two main parts:

- ▶ Section [2 DELMIA Apriso Web API](#) provides detailed information on how to configure DELMIA Apriso to make its web API available to client applications. In this scenario, DELMIA Apriso acts as an **API provider**.
- ▶ Section [3 Web API Client](#) focuses on how to call web APIs exposed by external providers using the Web API Client. In this scenario, DELMIA Apriso acts as a **client** application with respect to the third-party APIs it consumes.

## 1.1 Prerequisites

Note that this guide is intended for a highly technical audience. Although it contains a thorough overview of all the configuration steps needed to use DELMIA Apriso in either of the capacities discussed above, the reader should already have a good understanding of the purpose of web APIs and how to use them. Preferably, they should be familiar with the various authorization flows typically used in the web API context, such as the access token flow (according to the [OAuth 2.0 Authorization Framework](#)), and the API key flow. Knowledge of the HTTP protocol, including request headers and HTTP methods (particularly GET and POST), is also essential.

## 1.2 Glossary

This section provides an overview of key terms used in this document.

**Base URL** – the common part shared by each endpoint (see below) exposed by an API (for the DELMIA Apriso Web API the base URL is `http://{server_name}/Apriso/httpServices`).

**Client application** – an application calling a web API exposed by a provider.

**Endpoint** – endpoints indicate ways to access a resource (e.g., `operations/{operationCode}`). Endpoints are appended to the base URL (see above) to form a complete resource URL (e.g., `http://{server_name}/Apriso/httpServices/operations/{operationCode}`).

**REST** (Representational State Transfer) – a resource-oriented approach to designing APIs (for more information on REST, see [1.3 What is a RESTful API?](#)).

**Scope** – scopes restrict an application's access to resources exposed by an API by determining which endpoints it can access (for more information, see [Scopes](#)).

**STS** (Secure Token Service) – DELMIA Apriso's web service where client applications can request an access token, available at: `http://{server_name}/Apriso/Portal/sts/2.0/token`. STS is used during the access token-based authorization flow.

**Web API** – an application programming interface which can be accessed using the HTTP protocol. A web API exposes specific features to client applications which can access those features using predefined endpoints (see above).

**Web service provider** – a party exposing a web API for use by external applications (clients).

## 1.3 What is a RESTful API?

Representational State Transfer (REST) is an approach to designing web services. Any API which follows this approach is called a **RESTful API** or a **REST API**. As opposed to other architectural styles (such as SOAP), RESTful APIs are designed around **resources**, which are any kind of object, data, or service that can be accessed by the client. Typically, the client uses the HTTP protocol to interact with the resources by making requests (GET, POST, PUT, DELETE, etc.) to the **endpoints** (or URLs) exposed by the provider of the API.

Developed using these design principles, the DELMIA Apriso Web API exposes specific resources using endpoints such as:

```
http://{server_name}/Apriso/httpServices/operations
```

In this example, a GET request to the `operations` endpoint would return a list of all Standard Operations published through the Web API on that particular DELMIA Apriso server. Note that all endpoints share the same **base URL**: `http://{server_name}/Apriso/httpServices`.

**i** A proxy server can be used when the RESTful API is called. The proxy server must be configured in the `CentralConfiguration.xml` file.

**i** For a complete list of endpoints exposed by the DELMIA Apriso Web API, see the [Web API Reference](#).

## 2 DELMIA Apriso Web API

This section explains how to configure DELMIA Apriso so that an external application or service, called a "**client**", is able to call its web API. It also contains an overview of the authorization mechanisms used by the web API, presents the relevant configuration files, and provides detailed examples of requests and responses.

Before using the DELMIA Apriso Web API, an external application or service has to prove that it is authorized to access the resources exposed by that API. It does so by presenting specific credentials, which DELMIA Apriso validates against the ones stored in its configuration files and either grants the application access or refuses the request.

**i** All the credentials needed by a client application to access the DELMIA Apriso Web API, such as the API key, client ID, and client secret, are stored in the `ClientApplications.xml` file on the DELMIA Apriso server. For more information on how to register an application as a client application, see [2.4 Adding a Client Application](#).

The DELMIA Apriso Web API supports two modes of authorization: **API key** and **access token**. The details of the two authorization flows are presented in the subsections below.

### Default Response Format

The DELMIA Apriso Web API uses `application/json` as the default response format. Use the `Accept: application/xml` header if you want to receive data in the `application/xml` format.

### 2.1 API Key

In this authorization scenario, the client application authenticates its requests to the API using an **API key** and a **client ID**. The two unique sequences of characters are attached to the request using, respectively, the `Authorization` header with the `ApiKey` scheme, and the `X-Client-Application` header:

```
GET http://{server_name}/Apriso/HttpServices/hello/apiKeySecured HTTP/1.1
Host: {server_name}
Authorization: ApiKey F2AXB652-8A37-4F5A-8A2E-1BDE1484DAE5
X-Client-Application: 0F22E127-206E-4BE3-A1FC-1078F65A743A
```

The server then examines the request and validates the headers against the API key and client ID defined in the `ClientApplications.xml` file:

```
<ClientApplication name="Your application">
  <ClientId>0F22E127-206E-4BE3-A1FC-1078F65A743A</ClientId>
  <ApiKey>F4AXB652-8A37-4F5A-8A2E-1BDE0484DAE5</ApiKey>
</ClientApplication>
```


If the server fails to validate the provided credentials, it will refuse to authorize the request (HTTP 401 Unauthorized). If authentication succeeds, the server will respond with HTTP 200 OK:

```
HTTP/1.1 200 OK
Cache-Control: no-cache
Pragma: no-cache
Content-Type: application/json; charset=utf-8
Expires: -1
Server: Microsoft-IIS/10.0
api-supported-versions: 2
Content-Length: 25

"Correct API Key passed!"
```

## 2.2 Access Token

The access token flow ensures much greater security than the API key since it additionally requires the client application to pass Apriso Classic Portal authentication. It also offers greater flexibility by restricting the application's access only to certain "scopes", i.e. types of resources (see [Scopes](#)).

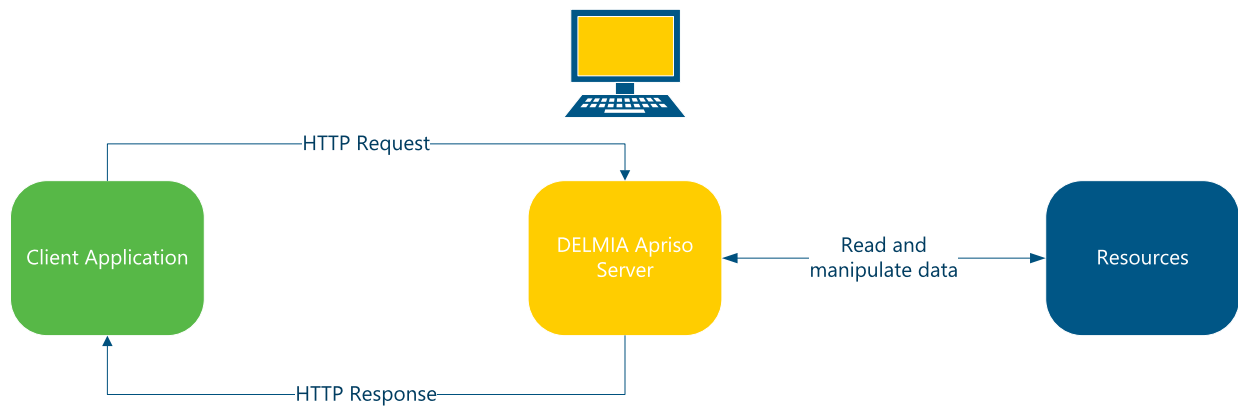
 Apriso Classic Portal has been deprecated.

### 2.2.1 Obtaining an Access Token

To gain access to the resources exposed through the API, the client application needs to authenticate its requests with an access token. To obtain the token, the application first has to send an appropriate request to the Secure Token Service (STS) available at the following address: `http(s)://{server_name}/Apriso/Portal/sts/2.0/token` (STS is an implementation of the [OAuth 2.0 Authorization Framework](#)).

Three parties are therefore involved in the authorization process:

- ▶ Client application
- ▶ DELMIA Apriso server (STS)
- ▶ Resources



## Scopes

Scopes restrict the application's access to resources by determining which endpoints it can access. For example, in order to call Standard Operations exposed via the DELMIA Apriso Web API, the client application needs the `standard_operations` scope so that it can use the operations endpoint (the full resource URL would be as follows: `http://{server_name}/Apriso/httpServices/operations`).

The following scopes are available in DELMIA Apriso out of the box:

- ▶ `standard_operations`
- ▶ `personalization`
- ▶ `cache`
- ▶ `data_paging`
- ▶ `message_processing`

**i** See the [Web API Reference](#) to learn which scope or scopes to use for specific resources.

**i** Scopes for which the client application can request an access token are configured in the `ClientApplications.xml` file, which is discussed in extensive detail in [2.4 Adding a Client Application](#).

The following sample request generates an access token valid for the `standard_operations` scope:

```
GET http://{server_name}/Apriso/Portal/sts/2.0/token?client_id=0F22E127-206E-4BE3-A1FC-1078F65A743A&response_type=token&redirect_uri=http%3A%2F%2F{server_name}%2FApriso%2Fmodules%2Foauth%2Foauth_callback.html&scope=standard_operations&state=samplestring HTTP/1.1
Host: {server_name}
Accept: application/json
```

## Authorization

The client application can obtain an access token from STS using any of the four supported authorization flows:

- ▶ [Implicit Grant](#) (temporary user authorization)
- ▶ [Authorization Code](#) (refreshable user authorization)
- ▶ [Resource Owner Password Credentials](#) (the user's credentials are passed in the request body)
- ▶ [Client Credentials](#) (application authorization)

**i** For an in-depth discussion of the four authorization flows, see [2.3 Authorization Flows](#).

Once the access token has been obtained, the application can use it to make requests to the API. The token should be placed in the Authorization header with the Bearer scheme:

```
GET http://{server_name}/Apriso/HttpServices/hello/tokenSecured HTTP/1.1
Host: {server_name}
Authorization: Bearer
ZX1KMGVYQW1PaUpLVjFRaUxDSmhir2NpT21KSVV6STFOaUo5LmV5SnpZMj13W1NJNk1uQmxjbk52YmdGc2FYcGhkR2x
2Ym14emRHRnVaR0Z5WkY5dmNHVn1ZWfJwYdI1ek1pd21ZMnhwW1c1MFgybGtJam9pTUVZNU1rVXhNamN0UVRBM1JTMD
BRa1V6TFVFeFJRtXRNVEEzT0VZMk5UVTNORE5CSW13aWfuUnBJam9pUVVST1NVNs1MQ0psY1hWcGNHMWxib1FpT21Ja
UxDSnNhD05sYm50bElqb21NaU1zSW1semN5STZJa0Z3Y21semJ5SXNjbUYxWkNjNk1rRndjbWx6Yn1Jc01tVjRjQ0k2
TVRVeU1USXd0ek15TW13aWJtSm1Jam94TlRJeE1qQXp0ek15Z1EuY3BWTzZ5amR3YmpoWVhVUEUwaVJsaGpxN0h5UW1
nRFNoYVVRaW81ZFR0WQ==
```

If the server fails to validate the provided credentials, it will refuse to authorize the request (HTTP 401 Unauthorized). If authentication succeeds, the server will respond with HTTP 200 OK:

```
HTTP/1.1 200 OK
Cache-Control: no-cache
Pragma: no-cache
Content-Type: application/json; charset=utf-8
Expires: -1
Server: Microsoft-IIS/10.0
api-supported-versions: 2
Content-Length: 32

"Correct token passed as ADMIN!"
```

## 2.3 Authorization Flows

This section contains an overview of the four main flows which can be used to obtain an access token:

- ▶ [Implicit Grant](#)
- ▶ [Authorization Code](#)



- ▶ Resource Owner Password Credentials
- ▶ Client Credentials

**i** For detailed information on these authorization flows, refer to the [OAuth 2.0 Authorization Framework](#).

**⚠** Apriso Classic Portal referenced in this document has been deprecated.

## Implicit Grant

- ▶ HTTP method: GET
- ▶ Requires Classic Portal authentication: Yes
- ▶ Data transmission form: Query string in URL
- ▶ Response form: Redirection to passed redirect URL
- ▶ Required parameters:
  - ▷ `response_type` – has to be set to "token" (lowercase letters)
  - ▷ `client_id` – has to be equal to your registered `client_id`
  - ▷ `redirect_uri` – has to be equal to one of the redirect URLs bound to the passed `client_id`
  - ▷ `scope` – has to be set to one or more (comma separated) scopes bound to the passed `client_id`
- ▶ Recommended parameters:
  - ▷ `state` – can be a random string; this value will be sent back to the requesting application with the response message to confirm that it is sent in response to your application's request; offers additional security

Example request:

```
GET http://{server_name}/Apriso/Portal/sts/2.0/token?client_id=0F22E127-206E-4BE3-A1FC-1078F65A743A&response_type=token&redirect_uri=http%3A%2F%2F{server_name}%2FApriso%2FApriso%2Fmodules%2Foauth%2Foauth_callback.html&scope=standard_operations&state=somespecyficstring HTTP/1.1
Host: {server_name}
Accept: application/json
```

Example response:

```
HTTP/1.1 302 Found
Cache-Control: no-cache, no-store
Pragma: no-cache
Expires: -1
Location: http://{server_name}/Apriso/callback.html#access_token=ZX1KMGVYQWlPaUpLVjFRaUxDSmhiR2NpT2lKSvV6STFOaUo5LmV5SnZMjl3WlNjNkluQmxjbk52YmdGc2FYcGhkR2x2Ym14emRHRnVaR0Z5WkY5dmNHVn1ZWfJwYdI1ek1pd2lZMnhwWlc1MFgybGtJam9pTUVZNU1rVXhNamN0UVRBMlJTMDBRa1V6TFVFeFJrTXRNVEEzT0VZMk5UVTNORE5CSWl3aWFuUnBJam9pUVVST1NVNSlMQ0psY1hWcGNHMWxib1FpT2lJaUxDSnNhD05sYm50bElqb2lNaU1zSW1semN5STZJa0Z3Y21semJ5SXNjbUYxWkNjNk1rRndjbWx6Yn1Jc01tVjRjQ0k2TVRveU1USXdoek15TWl3aWJtSm1Jam94TlRJeE1qQXp0ek15Z1EuY3BwTzZ5amR3YmpoWVhVUEUwaVJsaGpxN0h5UW1nRFRoYVVRaW81ZFR0&token_type=Bearer&expires_in=3600&state=somespecyficstring
Content-Length: 0
```

## Authorization Code

### Step 1 (Authorization Request)

- ▶ HTTP method: GET
- ▶ Requires Classic Portal authentication: Yes
- ▶ Data transmission form: Query string in URL
- ▶ Response form: Redirection to passed redirect URL
- ▶ Required parameters:
  - ▷ `response_type` – has to be set to "code" (lowercase letters)
  - ▷ `client_id` – has to be equal to your registered `client_id`
  - ▷ `redirect_uri` – has to be equal to one of the redirect URLs bound to the passed `client_id`
  - ▷ `scope` – has to be set to one or more (comma separated) scopes bound to the passed `client_id`
- ▶ Recommended parameters:
  - ▷ `state` – can be a random string; this value will be sent back to the requesting application with the response message to confirm that it is sent in response to your application's request; offers additional security

Example request:

```
GET http://{server_name}/Apriso/Portal/sts/2.0/token?client_id=0F22E127-206E-4BE3-A1FC-1078F65A743A&response_type=code&redirect_uri=http%3A%2F%2F{server_name}%2FApriso%2FApriso%2Fmodules%2Foauth%2Foauth_callback.html&scope=personalization&state=somespecificstring HTTP/1.1
Host: {server_name}
Accept: application/json
```

Example response:

```
HTTP/1.1 302 Found
Cache-Control: no-cache
Pragma: no-cache
Expires: -1
Location: http://{server_name}/Apriso/aprison/modules/oauth/oauth_callback.html?code=5f5579b1-1e92-491c-a24f-be11ac514dff&state=somespecificstring
Content-Length: 0
```

### Step 2 (Access Token Request)

- ▶ HTTP method: POST
- ▶ Requires Classic Portal authentication: No
- ▶ Data transmission form: application/x-www-form-urlencoded in request body
- ▶ Response form: HTTP response content
- ▶ Required parameters:
  - ▷ `grant_type` – has to be set to "authorization\_code" (lowercase letters)
  - ▷ `code` – use the code obtained in Step 1
  - ▷ `client_id` – has to be equal to your registered `client_id`

- ▷ `client_secret` – has to be set to your registered `client_secret`
- ▷ `redirect_uri` – the same one as in Step 1
- ▶ **Recommended parameters:**
  - ▷ `state` – can be a random string; this value will be sent back to the requesting application with the response message to confirm that it is sent in response to your application's request; offers additional security

Example request:

```
POST http://{server_name}/Apriso/Portal/sts/2.0/token HTTP/1.1
Content-Type: application/x-www-form-urlencoded
Host: {server_name}
Content-Length: 287

client_id=0F22E127-206E-4BE3-A1FC-1078F65A743A&client_secret=1E274C15-CC57-4DB9-A740-BB3930FD07CE&code=5f5579b1-1e92-491c-a24f-be11ac514dff&grant_type=authorization_code&redirect_uri=http%3A%2F%2F{server_name}%2FApriso%2FApriso%2Fmodules%2Foauth%2Foauth_callback.html&state=somespecificstring
```

Example response:

```
HTTP/1.1 200 OK
Cache-Control: no-cache
Pragma: no-cache
Content-Type: application/json; charset=utf-8
Expires: -1
Content-Length: 470

{"access_token":"eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJzY29wZSI6InBlcnNvbWZsaXphdGlubiIsImNsaWVudF9pZCI6IjBGMjJFMjI3LTlIiwuNkUtNEJFMjY1BMUzDLTEwNzhGNjVBNzQzQSIsImp0aSI6IkFETU10IiwiaWF0IjE1aXBtZW50IjoiiwibGljZW5zZSI6IjIiLCJpc3MiOiJBCHJpc28iLCJhdWQiOiJBCHJpc28iLCJleHAiOiJlMjY1NTU0MzU0Im5iZiI6MTUyNjU1NDgzNX0.TS6jeoz6NtimbzxYat15NGGEY5ntA00j5HoRutmito","token_type":"Bearer","expires_in":600,"refresh_token":"4d916b97-dfc6-494c-a59b-dbb73b057663","state":"somespecificstring"}
```

## Refresh Token

- ▶ HTTP method: POST
- ▶ Data transmission form: `application/x-www-form-urlencoded` in request body
- ▶ Response form: HTTP response content
- ▶ **Required parameters:**
  - ▷ `refresh_token` – use the `refresh_token` obtained in Step 2
  - ▷ `grant_type` – has to be set to `"refresh_token"` (lowercase letters)
- ▶ **Recommended parameters**
  - ▷ `state` – can be a random string; this value will be sent back to the requesting application with the response message to confirm that it is sent in response to your application's request; offers additional security

Example request:

```
POST http://{server_name}/Apriso/Portal/sts/2.0/token HTTP/1.1
Content-Type: application/x-www-form-urlencoded
Host: {server_name}
Content-Length: 100

refresh_token=4d916b97-dfc6-494c-a59b-dbb73b057663&grant_type=refresh_token&state=somespecificstring
```

Example response:

```
HTTP/1.1 200 OK
Cache-Control: no-cache
Pragma: no-cache
Content-Type: application/json; charset=utf-8
Expires: -1
Content-Length: 415

{"access_token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJzY29wZSI6InBlcnNvbWFSaXphdGlvbSIsImNsawVudF9pZCI6IjBGMjJFMtI3LTlWnkUtNEJFMy1BMUZDLTEwNzhGNjVBbnZqQSIsImp0aSI6IkFETU10IiwiaWF0IjoiIiwibGllZW5zZSI6IjIiLCJpc3MiOiJBcHJpc28iLCJhdWQlOiJBcHJpc28iLCJleHAiOjE1MjY1NTU5OTIsIm5iZiI6MTUyNjU1NTM5Mn0.j39LfGJGZ_x2LC6MtPk3hi3Anlj7SFsTWUzfrjoz-oM","token_type":"Bearer","expires_in":600,"state":"somespecificstring"}
```

## Resource Owner Password Credentials

- ▶ HTTP method: POST
- ▶ User credentials authentication: Standard Authentication or LDAP
- ▶ Data transmission form: application/x-www-form-urlencoded in request body
- ▶ Response form: HTTP response content
- ▶ Required parameters:
  - ▷ grant\_type – has to be set to "password" (lowercase letters)
  - ▷ login\_type – has to be set to "standard" or "ldap" (lowercase letters; "standard" will be chosen if no value is provided)
  - ▷ username – the name of the user whose credentials will be used
  - ▷ password – the password of the user whose credentials will be used
  - ▷ scope – has to be set to one or more (comma separated) scopes bound to the passed client\_id
  - ▷ client\_id – has to be equal to your registered client\_id
- ▶ Recommended parameters:
  - ▷ state – can be a random string; this value will be sent back to the requesting application with the response message to confirm that it is sent in response to your application's request; offers additional security

Request example:

```
POST http://{server_name}/Apriso/Portal/sts/2.0/token HTTP/1.1
Content-Type: application/x-www-form-urlencoded
Host: {server_name}
Content-Length: 159

grant_type=password&login_type=standard&username={username}&password=
{password}&scope=personalization&client_id=0F22E127-206E-4BE3-A1FC-
1078F65A743A&state=somespecificstring
```

Response example:

```
HTTP/1.1 200 OK
Cache-Control: no-cache
Pragma: no-cache
Content-Type: application/json; charset=utf-8
Expires: -1
Content-Length: 415

{"access_token":"
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJzY29wZSI6InBlcnNvbWFsXphdGlvbiIsImNsawVudF9pZCI6Ij
BGMjJFMTI3LTlWnkUtNEJFMy1BMUZDLTEwNzhGNjVBNzQzQSIsImp0aSI6IkFETU10IiwiaWF0IjoiIiwib
GljZm5zZSI6IiIsImIzcyI6IkFwcmlzbyIsImF1ZCI6IkFwcmlzbyIsImV4cCI6MTUyNjU2MDQyNiwiImIjoxNTI2
NTU2ODI2fQ.ewABEByD6qBz_9Qc1a2Dt70QjUtbF_bYhtNaXDPjeNE","token_type":"Bearer","expires_
in":3600,"state":"somespecificstring"}
```

## Client Credentials

- ▶ HTTP method: POST
- ▶ Requires Classic Portal authentication: No (the token is issued for the user defined in the EmployeeID key in the <FlexNet.BackgroundProcessingContext> section in Central Configuration)
- ▶ Data transmission form: application/x-www-form-urlencoded in request body
- ▶ Response form: HTTP response content
- ▶ Required parameters:
  - ▷ grant\_type – has to be set to "client\_credentials" (lowercase letters)
  - ▷ client\_id – has to be equal to your registered client\_id
  - ▷ client\_secret – has to be set to your registered client\_secret
  - ▷ scope – has to be set to one or more (comma separated) scopes bound to the passed client\_id
- ▶ Recommended parameters:
  - ▷ state – can be a random string; this value will be sent back to the requesting application with the response message to confirm that it is sent in response to your application's request; offers additional security

Request example:

```
POST http://{server_name}/Apriso/Portal/sts/2.0/token HTTP/1.1
Content-Type: application/x-www-form-urlencoded
Host: {server_name}
Content-Length: 174

grant_type=client_credentials&client_id=0F22E127-206E-4BE3-A1FC-1078F65A743A&client_secret=1E274C15-CC57-4DB9-A740-BB3930FD07CE&scope=personalization&state=somespecificstring
```

Response example:

```
HTTP/1.1 200 OK
Cache-Control: no-cache
Pragma: no-cache
Content-Type: application/json; charset=utf-8
Expires: -1
Content-Length: 417

{"access_token":"eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJzY29wZSI6InBlcnNvbWFSaXphdGlvb2IiImNsaWVudF9pZCI6IjBGMjJFMTIzLTliWnkUtnEjFMy1BMUZDLEwNzhGNjVBNzQzQSIsImp0aSI6IiN5c3RlbnSiImVxdWlwbWVudCI6IiIsImxpbY2Vuc2UiOiIiLCJpc3MiOiJBcHJpc28iLCJhdWQiOiJBcHJpc28iLCJleHAiOjE1MjY2NDM2NjQsIm5iZiI6MTUyNjU1NzI2NH0.B5A52Y6cdsZsyGUe04Xgs5wdNALDUYCYevLW_174zrQ","token_type":"Bearer","expires_in":86400,"state":"somespecificstring"}
```

## 2.4 Adding a Client Application

Before accessing the DELMIA Apriso Web API, an external application or service needs to obtain an access token or use an API key (see [2 DELMIA Apriso Web API](#)). To do so, the application or service must first be registered as a client application. This section explains how to add a client application in an access token-based flow and an API-key based flow.

All registered client applications and services are listed in the `ClientApplications.xml` file available in the following location on the DELMIA Apriso server:

▶ <drive>\Program Files\Dassault Systemes\DELMIA Apriso 2021\Website\CentralConfiguration

## Default Client Applications

By default, `ClientApplications.xml` already contains definitions of three clients which are able to obtain access tokens from the Secure Token Service (STS):

- ▶ FlexNetProcessBuilder
- ▶ DELMIA Apriso Portal
- ▶ DELMIA Apriso `${WebAddress}` (where `${WebAddress}` is resolved to server name)

**i** If you change the <ClientId> of DELMIA Apriso Portal, you also have to modify the scripts.js file found in <drive>\Program Files\Dassault Systemes\DELMIA Apriso 2021\WebSite\Portal\Scripts, where you need to change the value of TokenServiceClientId to that of <ClientId>.

## New Client Application

To register a client application with DELMIA Apriso, add an entry in the `ClientApplications.xml` file and restart DELMIA Apriso services.

An entry for a new client application should have the following general structure:

```
<ClientApplication name="{application_name}">
  <ClientId>{client_id}</ClientId>
  <ClientSecret>{client_secret}</ClientSecret>
  <ApiKey>{api_key}</ApiKey>
  <RedirectUri>
    <Uri>{redirect_uri}</Uri>
  </RedirectUri>
  <Scopes>
    <Scope name="{scope_name}" />
  </Scopes>
  <SupportedGrants>
    <Grant>{grant_type}</Grant>
  </SupportedGrants>
</ClientApplication>
```

Each section has a specific meaning:

- ▶ `<ClientId>` is the public ID of the client application which will be used for identification (the preferred value format is GUID)
- ▶ `<ClientSecret>` is a private key assigned to the client application which is used when requesting an access token (the preferred value format is GUID)
- ▶ `<ApiKey>` is a private key used for API key authorization (the preferred value format is GUID)
- ▶ `<RedirectUri>` is a list of valid URLs that the Secure Token Service (STS) will send the access token to (one of the URLs must be equal to `${WebRootURL}/Apriso/modules/oauth/oauth_callback.html`)
- ▶ `<Scopes>` is a list of valid scopes that the client application can request when obtaining a token (see [Scopes](#) for a full list of default scopes)
- ▶ `<SupportedGrants>` is a list of authorization flows supported by the client application. One or more of the following grant types can be used:
  - ▷ Implicit – the implicit grant type is used to obtain access tokens (it does not support the issuance of refresh tokens)
  - ▷ AuthorizationCode – the authorization code grant type is used to obtain both access tokens and refresh tokens
  - ▷ Password – this grant type is suitable for clients capable of obtaining the resource owner's credentials (username and password)
  - ▷ ClientCredentials – the client can request an access token using only its client credentials

**i** For detailed information on the authorization flows supported under **OAuth 2.0**, see the [OAuth 2.0 Authorization Framework](#).

Depending on which authorization mechanism (API key or access token) the client application will be using, certain sections are obligatory:

Section	API key	Access token
<ClientId>	✓	✓
<ClientSecret>		✓
<ApiKey>	✓	
<RedirectUri>		✓
<Scopes>		✓
<SupportedGrants>		✓

Once the client application is registered, restart DELMIA Apriso services. The application will then be able to request an access token from STS or call the API using the provided API key.

## 2.5 Example: Calling a Standard Operation

If a Standard Operation has been published as a REST Web Service in Process Builder's Web Service Manager, it can be called using the `operations` endpoint. The name of the Standard Operation being called should be placed after the name of the service, e.g. `http://{server_name}/Apriso/httpServices/operations/{operationCode}`.

**i** For more information on publishing Standard Operations in Process Builder, see the **Web Services Manager** topic in the [Process Builder Help](#).

**i** You can use a tool such as Fiddler or Postman to test GET and POST requests.

### Security

Three security levels are supported when calling Standard Operations published through the DELMIA Apriso Web API:

- ▶ Public
- ▶ API Key
- ▶ Access Token

#### 2.5.1 Sample Scenario

This sample scenario uses a simple Standard Operation (called "GetEmployee") with an SQL Query Function which returns the employee's name based on the provided ID:



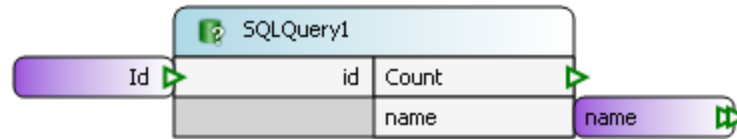


Figure 1 Sample Standard Operation

The Operation uses the following SQL query:

```
select name from employee where id = @Id
```

The Operation has been published as a REST web service in the Web Services Manager (for more information, see the **Web Services Manager** topic in the [Process Builder Help](#)).

For the sake of simplicity, the Operation's security level has been set to "Public", so that no authorization headers are required (for more details on authorization headers, see [2.1 API Key](#) and [2.2 Access Token](#)).

The screenshot shows the 'REST Web Service Properties' dialog box with the following details:

- General**
  - Published service name: GetEmployee
  - URL: http://PLMDEV001/Apriso/httpServices/operations/GetEmployee
  - ☒ Enable Web Service
- Security**
  - Security level: Public
  - Allowed Apriso Roles (no Roles selected means access for everybody):
  - Link...
- Details**
  - Operation code (only Active or Prototype operations): GetEmployee
  - Buttons: Test, Refresh, Link...
- Inputs / Outputs**

Name	Type
<b>Inputs</b>	
Id	Char
<b>Outputs</b>	
name	List of Char

Figure 2 REST Web Service Properties

To obtain more information about the Operation, make a GET request, using the Accept header to determine the Content-Type of the response (application/json or application/xml):

```
GET
http://{server_name}/Apriso/httpServices/operations/GetEmployee HTTP/1.1
Accept: application/json
Host: {server_name}
```

**i** All requests are made to the default revision of the Operation, unless a specific revision is provided in the URL, e.g. `http://{server_name}/Apriso/httpServices/operations/GetEmployee/REV.001.001`.

The API will give a response specifying the Operation's expected Inputs (Id: Integer) and Outputs (Name: ListOfChar), as well as a sample payload:

```
HTTP/1.1 200 OK
Cache-Control: no-cache
Pragma: no-cache
Content-Type: application/json; charset=utf-8
Expires: -1
Server: Microsoft-IIS/10.0
api-supported-versions: 1
X-AspNet-Version: 4.0.30319
X-Powered-By: ASP.NET
Date: Wed, 17 Jan 2018 13:21:42 GMT
Content-Length: 227

{
  "Inputs": [
    "Id: Integer"
  ],
  "Outputs": [
    "Name: ListOfChar"
  ],
  "ExecutionParameters": {
    "Inputs": {
      "Id": 1
    },
    "ExecutionMode": 1,
    "JobPool": "DEFAULT",
    "Retries": 1,
    "SleepTime": 1000,
    "SynchronousQueue": "DEFAULT",
    "Timeout": 5000
  }
}
```

The ExecutionParameters section contains a sample payload which you can use as a template for the body of your POST request.

When making the request, use the Accept header again to determine the format of the response: application/json OR application/xml. Content-Type should correspond to the format used in the body of your request.

The required Inputs should be placed in the request's body. Other parameters, such as ExecutionMode OR Timeout, are optional – if you do not provide them explicitly, defaults will be used (see [2.5.2 Execution Parameters](#) below for more details on available execution parameters):

```
POST
http://{server_name}/Apriso/httpServices/operations/GetEmployee HTTP/1.1
Accept: application/json
Content-Type: application/json
Host: {server_name}
Content-Length: 234

{
  "Inputs":{
    "Id":1
  },
  "ExecutionMode":1,
  "JobPool":"DEFAULT",
  "Retries":1,
  "SleepTime":1000,
  "SynchronuousQueue":"DEFAULT",
  "Timeout":5000
}
```

The API returns a list of Outputs. In this case, it is the Name (System Administrator) associated with the provided Id (1):

```
HTTP/1.1 200 OK
Cache-Control: no-cache
Pragma: no-cache
Content-Type: application/json; charset=utf-8
Expires: -1
Server: Microsoft-IIS/10.0
api-supported-versions: 1
X-AspNet-Version: 4.0.30319
X-Powered-By: ASP.NET
Date: Wed, 17 Jan 2018 13:36:36 GMT
Content-Length: 45

{
  "Outputs":{
    "Name":[
      "System Administrator"
    ]
  }
}
```

## 2.5.2 Execution Parameters

```
<Provider name="OWM" friendlyName="Open Weather Map">
  <ApiKey>
  <Headers/>
  <QueryString>q=London&appid=d0e841fadf87e22e9f1def6d7407fe77</QueryString>
</ApiKey>
</Provider>
```

Apart from the expected Inputs, you can also provide a number of other execution parameters which will be passed on to Job Executor. If an execution parameter is not provided, the default value will be used (see [Table 1 Execution Parameters](#) below).

**i** For more information about how jobs are executed in DELMIA Apriso, see the [Background Job Processing Technical Guide](#).

Parameter	Description	Default
ExecutionMode	<ul style="list-style-type: none"> <li>▶ 0 – <b>In Process</b> (will be executed synchronously in the IIS process)</li> <li>▶ 1 – <b>Synchronous</b> (will be executed synchronously by Job Executor; 200 OK is returned if the job is successfully <u>executed</u>)</li> <li>▶ 2 – <b>Asynchronous</b> (will be executed asynchronously by Job Executor; 200 OK is returned if the job is successfully <u>queued</u>)</li> </ul>	1 – Synchronous
JobPool	The name of the Job Pool to be used.	empty
Retries	How many times the job is to be retried in case of failure.	1
SleepTime	The interval between each job execution attempt (in milliseconds).	1000
SynchronousQueue	The name of the Synchronization Queue to be used.	empty
Timeout	The maximum amount of time allowed for the duration of a single job execution (in milliseconds).	5000

Table 1 Execution Parameters

## 3 Web API Client

DELMIA Apriso is able to call external APIs using the **Web API Client**, which is a JavaScript API which facilitates the consumption of APIs protected with an API key or an access token. It can be used in the HTML Layout Editor's JavaScript tab in Process Builder (see the [Process Builder Help](#)).

**i** Any entity which makes a web API available for use by other parties is called a **web service provider**. In order to be able to call an external API from the Web API Client, its provider must be added to the `WebServiceProviders.xml` file (for more information, see [3.2 Adding a Web Service Provider](#)).



Figure 3 Web API Client – HTML Layout Editor's JavaScript tab in Process Builder

### 3.1 Authorization

Depending on the authorization framework implemented by the API's provider, one of the following client objects should be used in the HTML Layout Editor's JavaScript tab:

- ▶ `Apr.WebAPI.OAuthClient` – for an access token-based flow compliant with the OAuth 2.0 specification (this is an [Implicit Grant](#) flow)
- ▶ `Apr.WebAPI.ApiKeyClient` – for an API key-based flow
- ▶ `Apr.WebAPI.ThreeDPassportClient` – for 3DPassport authentication

Each client's constructor accepts the name of the provider to be used, for example:

```
var client = new Apr.WebAPI.OAuthClient("apriso");
```

This name must correspond exactly to the provider's name in `WebServiceProviders.xml`:

```
<Provider name="apriso" friendlyName="DELMIA Apriso Web API">
  <OAuth>
    <Implicit>
      <AuthorizationUri>${WebRootURL}/Portal/sts/2.0/token</AuthorizationUri>
      <ClientId>A06FDE50-B9C8-41E1-9D14-F7D3737F96E1</ClientId>
      <RedirectUri>${WebRootURL}/Apriso/modules/oauth/oauth_callback.html</RedirectUri>
      <Scope>personalization,standard_operations,cache,data_paging</Scope>
    </Implicit>
    <ClientCredentials>
      ...
    
```

Once the correct client has been instantiated, it can be used to make requests to the API:

```
var client = new Apr.WebAPI.OAuthClient("apriso");
var url = "http://serverName.dsone.3ds.com/apriso/httpservices/operations/operationCode";
client.Get(url, function(response) {
  console.log(response);
}, function() {
  console.log("ERROR");
});
```

## 3.2 Adding a Web Service Provider

Any entity which makes a web service (an API) available for use by other parties is called a web service provider. To be able to call an external web service, you must first add its provider to `WebServiceProviders.xml`, along with any information required to call the web service, as described farther down in this section.

The `WebServiceProviders.xml` file is available in the following location on the DELMIA Apriso server:

► <drive>\Program Files\Dassault Systemes\DELMIA Apriso 2021\Website\CentralConfiguration

To register a web service provider with DELMIA Apriso, first register your application with the provider, obtain the necessary configuration details (such as the endpoint, client ID, API key, etc.) from the provider, and finally add an entry in the `WebServiceProviders.xml` file and restart DELMIA Apriso services.

**i** Before registering a provider, you should familiarize yourself with its documentation, particularly with the methods of authenticating your requests.

### Default Web Service Providers

By default, `WebServiceProviders.xml` already contains the "apriso" provider, which enables calling the DELMIA Apriso Web API (for more information, see [2 DELMIA Apriso Web API](#)).

## New Web Service Provider

To register a new web service provider with DELMIA Apriso, add an entry in the `WebServiceProviders.xml` file and restart DELMIA Apriso services.

**i** The name of the provider, which must be unique, is used by the Web API Client. `friendlyName` is used by the Web Service Function in Process Builder (for more information, see [Process Builder Help](#)).

An entry for a web service provider may contain one or more of the following sections corresponding to the authorization methods it supports:

- ▶ `<OAuth>` for access token-based authentication
- ▶ `<ApiKey>` for API key-based authentication
- ▶ `<ThreeDPassport>` for 3DPassport authentication

Assuming a provider supports all three available authorization mechanisms (an access token, an API key, and 3DPassport), its entry in `WebServiceProviders.xml` would look like this:

```

<Provider name="myProvider" friendlyName="My Provider">
  <OAuth>
    <Implicit>
      <AuthorizationUri>${WebRootURL}/Portal/sts/2.0/token</AuthorizationUri>
      <ClientId>A06FDE50-B9C8-41E1-9D14-F7D3737F96E1</ClientId>
      <RedirectUri>${WebRootURL}/Apriso/modules/oauth/oauth_callback.html</RedirectUri>
      <Scope>personalization,standard_operations,cache,data_paging</Scope>
    </Implicit>
    <ClientCredentials>
      <AuthorizationUri>${WebRootURL}/Portal/sts/2.0/token</AuthorizationUri>
      <Body>client_id=A06FDE50-B9C8-41E1-9D14-F7D3737F96E1&client_secret=77843C4F-18F0-4432-BBFD-25467BCEF116&grant_type=client_credentials&scope=personalization,standard_operations,cache,data_paging</Body>
      <AuthorizationHeader/>
      <QueryString/>
      <HttpMethod>POST</HttpMethod>
    </ClientCredentials>
  </OAuth>
  <ApiKey>
    <Headers>
      <Header>Authorization: ApiKey EE81825C-C6A0-47E4-9916-75A059B38DE9</Header>
      <Header>X-Client-Application: A06FDE50-B9C8-41E1-9D14-F7D3737F96E1</Header>
    </Headers>
    <QueryString/>
  </ApiKey>
  <ThreeDPassport>
    <AuthorizationUri>https://{address}/iam/login</AuthorizationUri>
    <Server>
      <Username>user</Username>
      <Password>password</Password>
    </Server>
  </ThreeDPassport>
</Provider>

```

## <OAuth>

The `<OAuth>` section contains information required to call an access token-protected API. Currently, two authorization flows are supported: `<Implicit>` and `<ClientCredentials>`. Any other flows will be ignored. For detailed information on authorization flows, see [2.3 Authorization Flows](#).

- ▶ `<AuthorizationUri>` is the URL of the provider's token service
- ▶ `<Implicit>` – the implicit authorization flow (this is the default flow for the Web API Client)
  - ▷ `<ClientId>` is the public ID of your registered client application or service, which will be used for identification
  - ▷ `<RedirectUri>` is the URL where the access token will be sent  
(`${WebRootURL}/Apriso/modules/oauth/oauth_callback.html`)
  - ▷ `<Scope>` is a comma-delimited list of the requested scopes (for more information on what scopes are, see [Scopes](#))



- ▶ **<ClientCredentials>** – the client credentials authorization flow (this is the default flow for the Web Service Function in Process Builder; it is also used for machine-to-machine authentication)
  - ▷ **<Body>** is the body of the request (use query-string formatting, i.e. "field1=value1&field2=value2")
  - ▷ **<AuthorizationHeader>** is the provider-specific authorization header
  - ▷ **<QueryString>** is the query string to be attached to the request (if required by the provider). Use "&" to separate individual parameters (e.g. parameter1=value1&parameter2=value2).
  - ▷ **<HttpMethod>** is the HTTP method to be used, which can be either POST (default) or GET

### <ApiKey>

The **<ApiKey>** section contains information required to call a key-protected API.

- ▶ **<Headers>** is the list of headers to be used with the request
  - ▷ **<Header>** should have the "Field: Value" format (e.g., Authorization: ApiKey ABCD1234)
- ▶ **<QueryString>** is the query string to be attached to the request. It should be used to pass any parameters required by the provider which will not change in runtime. Use "&" to separate individual parameters (e.g. parameter1=value1&parameter2=value2).

### <ThreeDPassport>

The **<ThreeDPassport>** section contains information required to call a web service which uses 3DPassport authentication.

**i** Any server calling 3DPassport services must have HTTPS enabled. For more information, see the **Enabling HTTPS** section in the [Security Implementation Guide](#).

- ▶ **<AuthorizationUri>** is the 3DPassport login page
- ▶ **<Server>** contains credentials to be used for 3DPassport authentication (they will be used by the Web Service Function in Process Builder when making calls to web services using 3DPassport authentication)
  - ▷ **<Username>**
  - ▷ **<Password>**

## 3.3 Functions

The Web API Client offers four function variants for each of the supported HTTP verbs (GET, DELETE, POST, and PUT):

- ▶ GET
  - ▷ Get (**url**)
  - ▷ Get (**url**, **headers**)
  - ▷ Get (**url**, **successCallback**, **errorCallback**)
  - ▷ Get (**url**, **successCallback**, **errorCallback**, **headers**)
- ▶ DELETE
  - ▷ Delete (**url**)
  - ▷ Delete (**url**, **headers**)

- ▷ Delete ([url](#), [successCallback](#), [errorCallback](#))
- ▷ Delete ([url](#), [successCallback](#), [errorCallback](#), [headers](#))
- ▶ POST
  - ▷ Post ([url](#), data)
  - ▷ Post ([url](#), data, [headers](#))
  - ▷ Post ([url](#), data, [successCallback](#), [errorCallback](#))
  - ▷ Post ([url](#), data, [successCallback](#), [errorCallback](#), [headers](#))
- ▶ PUT
  - ▷ Put ([url](#), data)
  - ▷ Put ([url](#), data, [headers](#))
  - ▷ Put ([url](#), data, [successCallback](#), [errorCallback](#))
  - ▷ Put ([url](#), data, [successCallback](#), [errorCallback](#), [headers](#))

Where:

- ▶ [url](#) is the endpoint to be used
- ▶ [headers](#) are the request's headers

**i** A Content-Type header must be provided with all POST and PUT requests (for more information, see [3.4 Default and Custom Headers](#))

- ▶ [successCallback](#) is the function to be called if the request is successful
- ▶ [errorCallback](#) is the function to be called if the request fails
- ▶ data is the payload to be attached to the POST request

**i** Not all clients support all four verbs (see the table below).

	GET	DELETE	POST	PUT
<a href="#">OAuthClient</a>	✓	✓	✓	✓
<a href="#">ApiKeyClient</a>	✓	✓	✓	✓
<a href="#">ThreeDPassportClient</a>	✓			

## GET / DELETE

Below is an example of a GET request in an access token-based flow (the same example could be used with API key and 3DPassport authentication, where [ApiKeyClient](#) and [ThreeDPassportClient](#) would be used respectively instead of [OAuthClient](#)):

```
var client = new Apr.WebAPI.OAuthClient("apriso");
var url = "http://serverName.dsone.3ds.com/apriso/httpservices/operations/operationCode";
client.Get(url, function(response){ console.log(response); }, function(){ console.log("ERROR"); });
```

If no callbacks are specified, the function returns a promise (for more information on promises, refer to MDN web docs):

```
var client = new Apr.WebAPI.OAuthClient("apriso");
var url = "http://serverName.dsone.3ds.com/apriso/httpservices/operations/operationCode";
var getPromise = client.Get(url);
getPromise
    .then(function(response){ console.log(response); })
    .catch(function(){ console.log("ERROR"); });
```

## POST / PUT

Below is an example of a POST request in an access token-based flow (in an API key-based flow, `ApiKeyClient` would be used):

```
var client = new Apr.WebAPI.OAuthClient("apriso");
var url = "http://serverName.dsone.3ds.com/apriso/httpservices/operations/operationCode";
var data = {"Inputs":{"Example":1}};
var headers = {"Content-Type":"application/json"};
client.Post(url, data, function(response){ console.log(response); }, function(){
    console.log("ERROR"); }, headers);
```

If no callbacks are specified, the function returns a promise, for example:

```
var client = new Apr.WebAPI.OAuthClient("apriso");
var url = "http://serverName.dsone.3ds.com/apriso/httpservices/operations/operationCode";
var data = {"Inputs":{"Example":1}};
var headers = {"Content-Type":"application/json"};
var getPromise = client.Post(url, data, headers);
getPromise
    .then(function(response){ console.log(response); })
    .catch(function(){ console.log("ERROR"); });
```

## 3.4 Default and Custom Headers

When making a POST request, provide the Content-Type header, which can be `application/json` or `application/xml` depending on the format of the payload data.

If Content-Type is not defined, `application/json` is used by default.

If not explicitly overridden, the following Accept header is used with each request:


Accept: `application/json, text/javascript`.

You can also provide your own custom headers. To do so, use a JSON object or a JSON string. For example, in the code snippet below the headers variable ensures that an XML document is returned in the response:

```
var client = new Apr.WebAPI.OAuthClient("apriso");
var url = "http://serverName.dsone.3ds.com/apriso/htpservices/operations/operationCode";
var data = {"Inputs":{"Example":1}};
var headers = {"Accept":"application/xml", "Content-Type":"application/json"};
client.Post(url, data, function(response){ console.log(response); }, function(){
console.log("ERROR"); }, headers);
```

## 3.5 Debugging

When debugging the Web API Client in the DELMIA Apriso Portal or Apriso Classic Portal, you can use the developer tools in your Internet browser of choice.

 Apriso Classic Portal has been deprecated.

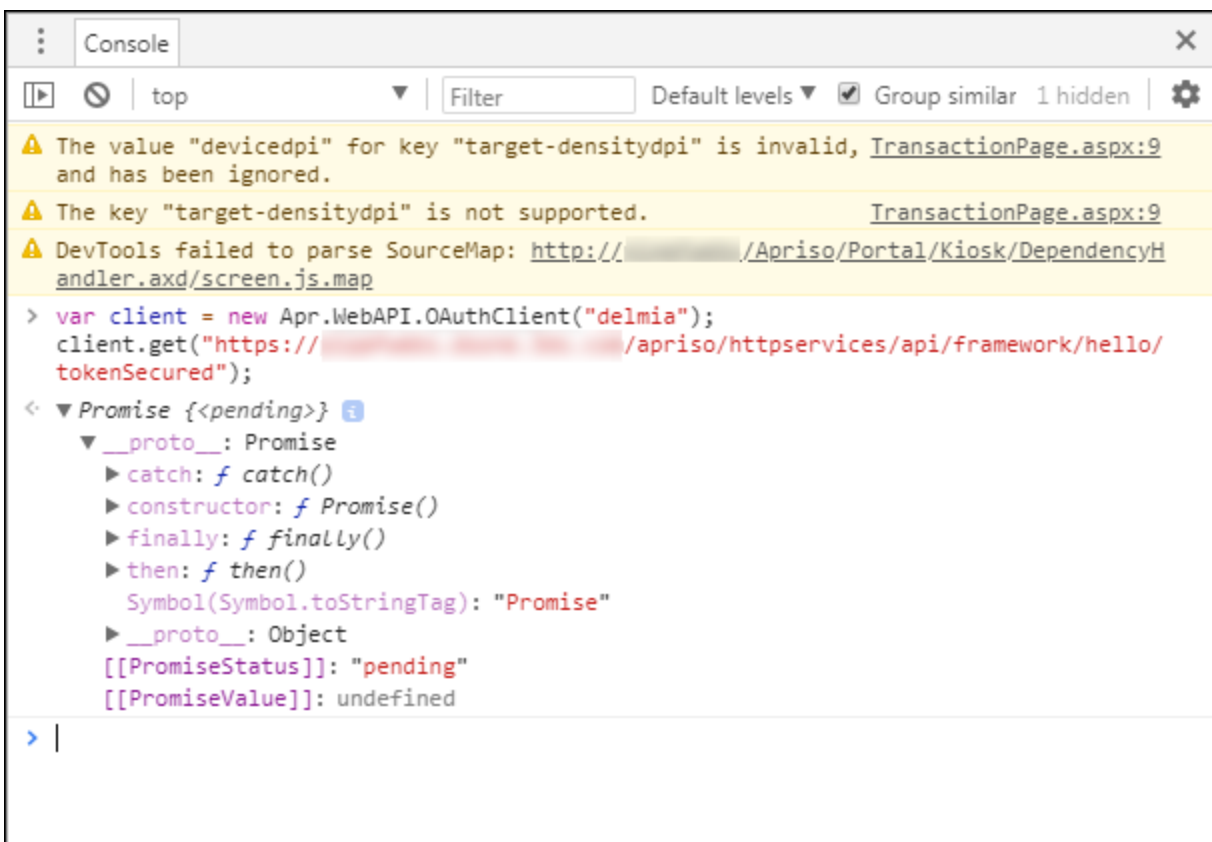


Figure 4 Web API Client debugging in the DELMIA Apriso Classic Portal

However, be aware that in the DELMIA Apriso Portal, the Web API Client script is linked at the level of the `iframe` which displays the content window (`contentWindow`). You should therefore refer to this `iframe` when using the Client (replace `#iframe_id` with the ID of the `iframe`, or use its class to identify it):

```
var iframe = $("#iframe_id");
var client = new iframe.contentWindow.Apr.WebAPI.OAuthClient("apriso");
client.Get("https://{server}/apriso/htpservices/api/framework/hello/tokenSecured");
```

## 3.6 Example: Calling an External Web API

This section contains a detailed walk-through on using an API from inside DELMIA Apriso. Two authentication scenarios are presented:

- ▶ [OAuth Authentication](#)
- ▶ [API Key Authentication](#)

To best illustrate the process in real-life conditions, two publicly available REST APIs are used as examples (Gmail and OpenWeatherMap). You should therefore become familiar with both APIs' documentation to fully understand the configuration steps presented in this section.

**i** Note that both scenarios are based on the assumption that you already have the necessary credentials to authorize requests (API key and client ID).

### OAuth Authentication

This sample scenario shows how to obtain a list of messages in a user's inbox from the Gmail API ( using DELMIA Apriso's Web API Client. All examples in this scenario will use a mock client ID, which should be replaced with valid credentials when making requests to the API.

**i** A Gmail account is required to complete the procedure below.

Procedure:

1. Follow the instructions in *Implementing Server-Side Authorization* at Gmail API documentation to register your application (DELMIA Apriso) and obtain a client ID.

**Client ID:**

```
887456409710-bv7a838sqvbofb61c6q8ou7u7ctiq2ig.apps.googleusercontent.com
```

2. Consult Gmail API documentation to learn which endpoint to use, how to authenticate requests, and which scopes are required to list a user's messages.

A list of all messages in a user's inbox can be obtained from the `messages` **endpoint** (replace `{user_email_address}` with a specific email address):

```
https://www.googleapis.com/gmail/v1/users/{user_email_address}/messages
```

The access token can be obtained from the **authorization URL** below:

```
https://accounts.google.com/o/oauth2/auth
```

Read operations on a user's account require the following **scope**:

```
https://www.googleapis.com/auth/gmail.readonly
```

3. Add Gmail to `WebServiceProviders.xml` (for more information, see [3.2 Adding a Web Service Provider](#)) and restart DELMIA Apriso services.

```
<Provider name="Gmail" friendlyName="Google Mail">
  <OAuth>
    <Implicit>
      <AuthorizationUri>https://accounts.google.com/o/oauth2/auth</AuthorizationUri>
      <ClientId>887456409710-
bv7a838sqvbofb61c6q8ou7u7ctiq2ig.apps.googleusercontent.com</ClientId>
      <RedirectUri>${WebRootURL}/Apriso/modules/oauth/oauth_callback.html</RedirectUri>
      <Scope>https://www.googleapis.com/auth/gmail.readonly</Scope>
    </Implicit>
  </OAuth>
</Provider>
```

4. Use the following code snippet to call the API from within an Operation in Process Builder (for more information, see [3 Web API Client](#)):

```
var client = new Apr.WebAPI.OAuthClient("Gmail");
var url = "https://www.googleapis.com/gmail/v1/users/{user_email_address}/messages";
client.Get(url, function(response){ console.log(response); }, function(){ console.log
("ERROR"); });
```

**i** Note that a pop-up will appear asking the user to log in to Gmail and authorize your application's access to their data.

5. The API should return a JSON object in response body:

```
{"ok":true,"status":200,"statusText":"","headers":{"date": "Tue, 22 May 2018 10:34:14 GMT", "content-length": 153, "expires": "Tue, 22 May 2018 10:34:14 GMT", "server": "GSEtag: Mr5G1ppow16hK9x9KiNoxDVbWS4/EPCHu2HZVNCB55qVjYkOPKCX-Yo", "content-type": "application/json; charset=UTF-8", "vary": "Origin, X-Origin", "cache-control": "private, max-age=0, must-revalidate, no-transform"}, "data": {"messages": [ { "id": "1638734affd81fa3", "threadId": "1638734affd81fa3" }, { "id": "1638734ae6741cc2", "threadId": "1638734ae6741cc2" }, { "id": "1638734ac066f6a7", "threadId": "1638734ac066f6a7" }, { "id": "1638734a82e9c460", "threadId": "1638734a82e9c460" } ], "resultSizeEstimate": 4}}
```

## API Key Authentication

This sample scenario shows how to obtain current weather data for the city of London from OpenWeatherMap API using DELMIA Apriso's Web API Client. All examples in this scenario will use a mock API key, which should be replaced with a valid key when making requests to the API.

Procedure:

1. Follow these instructions to create an account with OpenWeatherMap and obtain an API key: <https://openweathermap.org/appid>.

**API key:**

```
d0e841fadf87e22e9f1def6d7407fe77
```

2. Consult OpenWeatherMap API documentation to learn which endpoint and parameters to use, as well as how to authenticate requests.

Current weather data is available from the `weather` **endpoint**:

```
http://api.openweathermap.org/data/2.5/weather
```

The API expects a `q` parameter with the name of the city. Additionally, the API key should be provided in an `appid` parameter with each request for authentication purposes:

```
q=London&appid=d0e841fadf87e22e9f1def6d7407fe77
```

Thus, the API expects a GET request to the following URL:

```
http://api.openweathermap.org/data/2.5/weather?q=London&appid=d0e841fadf87e22e9f1def6d7407fe77
```

3. Add OpenWeatherMap to `WebServiceProviders.xml` (for more information, see [3.2 Adding a Web Service Provider](#)) and restart DELMIA Apriso services.

**i** Note that the `Headers` section is empty because the API does not expect any headers to be provided with this request:

```
<Provider name="OWM" friendlyName="Open Weather Map">
  <ApiKey>
    <Headers/>
    <QueryString>q=London&appid=d0e841fadf87e22e9f1def6d7407fe77</QueryString>
  </ApiKey>
</Provider>
```

4. Use the following code snippet to call the API from within an Operation in Process Builder (for more information, see [3 Web API Client](#)):

```
var client = new Apr.WebAPI.ApiKeyClient("OWM");
var url = "http://api.openweathermap.org/data/2.5/weather";
client.Get(url, function(response){ console.log(response); }, function(){ console.log("ERROR"); });
```

5. The API should return a JSON object in response body:

```
{ "coord": { "lon": -0.13, "lat": 51.51 }, "weather":  
[ { "id": 801, "main": "Clouds", "description": "few  
clouds", "icon": "02d" } ], "base": "stations", "main":  
{ "temp": 288.15, "pressure": 1016, "humidity": 67, "temp_min": 286.15, "temp_  
max": 289.15 }, "visibility": 10000, "wind": { "speed": 4.1, "deg": 360 }, "clouds":  
{ "all": 12 }, "dt": 1526975400, "sys":  
{ "type": 1, "id": 5091, "message": 0.0036, "country": "GB", "sunrise": 1526961536, "sunset": 152701  
8981 }, "id": 2643743, "name": "London", "cod": 200 }
```



## 4 References

### Internal Documentation

#### 1. **Web API Reference**

This documentation is generated automatically and describes in detail the Web APIs delivered with DELMIA Apriso. It can be accessed on your DELMIA Apriso server (<server name>/apriso/htpservices/help).

#### 2. **OAuth 2.0 Authorization Framework**

This document contains the specification of the OAuth 2.0 Authorization Framework. It can be found on the RFC Editor website.

#### 3. **Security Implementation Guide**

Provides an overview of DELMIA Apriso security and information on effectively securing all instances of DELMIA Apriso.

#### 4. **Central Configuration Documentation**

Describes in detail all the keys of the Central Configuration (CC) file for DELMIA Apriso. Various sections group the keys for individual modules or distinct functional areas.

#### 5. **Process Builder Help**

Provides an overview of DELMIA Apriso Process Builder (PB) and information on installing and using the application. This Help describes the user interface elements, entity maintenance, available Business Controls, and management of Processes, Operations, and Screen Flows.

#### 6. **Background Job Processing Technical Guide**

Provides an overview of DELMIA Apriso Background Job Processing and presents information on using the tool to its full potential.

### 3DS Support Knowledge Base

If you have any additional questions or doubts not addressed in our documentation, feel free to visit the **3DS Support Knowledge Base** at <https://support.3ds.com/knowledge-base/>.