About 3DSpace Web Services

Technical Article

Abstract

3DSpace web services are RESTFUL defined with JAX-RS/JSON infrastructures [1]. This article recalls the common REST Web services principles and points to the 3DSpace specificities.

Protocol HTTPs

3DSpace web services enable you access to confidential data of both the company and the employees. Therefore 3DSpace web services use high security tools to prevent themselves from attacks [2]. HTTPS protocol is one of them. Consequently requests must be made using HTTPS protocol instead of HTTP to ensure security of data transfer.

URL

URL and Tenant

The format of the URL is expressed as follows:

```
https://<authority_server>
```

The contents of `authority_server` is slightly different between on-premises and cloud environments.

On-premises, there is only one platform instance [3], in other words one tenant. Let's say the URL is as follows:

```
https://<server_dependant>
```

On cloud, there are several platform instances [3], in other words several tenants. The URL is as follows:

```
https://<tenant_id><server_dependant>
```

About tenant identifier and case sensitivity

An URL is usually case insensitive [5]. It means whatever the case of `authority_server` it should work to reach the server. But with 3DSpace the case is important. Indeed 3DSpace authentication checks the tenant in the URL, and this check is case sensitive. Since the tenant identifier is an identifier of platform instance which can contain upper-case as lower-case letters, the HTTP request URL MUST exactly matches the expected URL.

Let's suppose the platform instance is `DSQAL005`

```
https://dsqal005-....3dx-xxx.3ds.com/enovia/resources/...  is wrong
https://DSQAL005-...3dx-xxx.3ds.com/enovia/resources/...  is correct
```

If you are not sure of the tenant identifier case in the URL, the solution is to use the `tenant` query parameter with the tenant identifier (I.e. platform instance) as value:

```
https://dsqal005-...3dx-xxx.3ds.com/enovia/resources/...?tenant=DSQAL005  is now correct
https://DSQAL005-...3dx-xxx.3ds.com/enovia/resources/...?tenant=DSQAL005  is still correct
```

Note: You do not have to test the current running environment (on-premises or on cloud), in all cases append the `tenant` query parameter. For on-premises solution (the value is "OnPremise") the parameter is just ignored by 3DSpace server APIs.

Retrieving 3DSpace URL from a widget app

Interactively, inside a widget application, the JavaScript API DS/i3DXCompassServices/i3DXCompassServices returns the 3DSpace URL(s).

Just below an extract of javascript code using the API to retrieve the 3DSpace URL for a given platform instance (tenant):

```
...
i3DXCompassServices.getServiceUrl( {
        serviceName: '3DSpace',
        platformId : widget.getValue("x3dPlatformId") ,
        onComplete : function (URL_3DSpace) {...}  ,
        onFailure  : ...
} ) ;
...
```

The referenced technical article [3] goes further with this API and the notion of platform instance.

Note: The 3DSpace URLs returned by this AMD module are always lower-case letters. Therefore if your code is multi-environment (cloud and onpremises), or cloud only, do not forget to append to the URL the *tenant* query parameter (*widget.getValue("x3dPlatformId")*) for reason mentioned above.

Below an example if the resource path does not contain a query parameter:

```
var myPATH=URL_3DSpace + resource_path + ?tenant=tenantID
```

And then if the resource path already contains a query parameter:

```
var myPATH=URL_3DSpace + resource_path + &tenant=tenantID
```

The referenced use case [7] may help you to be familiar with those API.

Conclusion

On cloud environment, use the `tenant` query parameter. Its value - case sensitive- is the plaform instance.

---

paths

The **root path** is the URL section between the 3DSpace URL, and the **application path**. The latter represents the relative path of the resource.

```
https://<3DSpace_authority_server><root_path><application_path>
```

For all 3DSpace web services, the root path is */resources*. Let's take an example: if the application path is /person, the final URL to reach a person resource is:

```
https://<3DSpace_authority_server>/resources/person
```

The combination of the root and the application path, is named the **path**.

```
https://<3DSpace_authority_server><path>
```

The 3DSpace web service API reference pages mention the URL path which always starts as follows: `/resources/...`

Login API

For a batch, before using a 3DSpace web service, a 3DEXPERIENCE platform and 3DSpace login must be performed. The technical article "Performing Service Login and Logout" [8] gives all the details.

The string to pass as value of the `service` query parameter for the 3DPassport login API is the CSRF token URL which the tenant (*XXX*) is appended for cloud solution.

```
String SpaceLoginURL=URLEncoder.encode(<URL_3DSpace>+ "/resources/v1/application/CSRF" + "?tenant=" + XXX, "UTF-8");
String PassportloginURL= <3DPassport> + "/login?service=" + SpaceLoginURL ;
```

For on premises, there are two solutions. Either with the tenant query parameter whose the value is "OnPremise" :

```
String SpaceLoginURL=URLEncoder.encode(<URL_3DSpace> + "/resources/v1/application/CSRF" + "?tenant=OnPremise" , "UTF-8");
String PassportloginURL= <3DPassport> + "/login?service=" + SpaceLoginURL ;
```

Or without the tenant query parameter:

```
String SpaceLoginURL=URLEncoder.encode(<URL_3DSpace> + "/resources/v1/application/CSRF", "UTF-8");
String PassportloginURL= <3DPassport> + "/login?service=" + SpaceLoginURL ;
```

Methods

The usual HTTP methods (also named verbs) and their meaning:

Verb	Usual Meaning
GET	for fetching of one or several resources
PUT	for updating a resource with full data

DELETE	for deleting a resource
POST	for creating a resource
PATCH	for updating a resource with partial data

In each web service documentation, beside the resource path, the method is mentioned. Just below extract of List Persons:

Method	Path
GET	/resources/modeler/pno/person

Request

This section deals with the information to be proceeded by the server: the input parameters, the format for the inputs and the response, and the security context value.

The input parameters can be passed in the URL (as query parameters), and/or into a body. The other additionnal information are passed by header fields.

Query Parameters

A query parameter is part of the URL. The format is as follows:

```
<path>?<param_name_1>=<param_val_1>&<param_name_2>=<param_val_2>...
```

Append ? after the URL path to start the first parameter, and add & between each others. Below an example with the List Persons API

```
/resources/modeler/pno/person?pattern=*Admin*&select=firstname&select=email
```

About the parameter names

The parameter name is **case sensitive**. If the expected parameter is "select", but "Select" is passed, the parameter is not taken into account. It is considered as missing by the API. But the HTTP request only fails if the parameter is mandatory, otherwise the default behavior for the *missing* parameter is applied.

In the previous example, if 'pattern' is written 'Pattern', the HTTP request fails since 'pattern' is mandatory. But if 'select' is written 'Select', the response does not take into account the select clause parameters.

About the parameter values

Parameters are part of the URL, so their values must be **UTF8 URL Encoded**.

So for example (Get Collaborative Space) such URL is invalid:

```
/resources/modeler/pno/collabspace/3DS Collab Space
```

While this one is valid:

```
/resources/modeler/pno/collabspace/3DS%20Collab%20Space
```

Each space is replaced with "%20" .

There are functions to ensure the right format for the parameter values. Let's see two kinds of such functions:

- In java, with URLEncoder (java.net.URLEncoder), param_value=URLEncoder.encode(param,"UTF-8")
- In JavaScript, with encodeURIComponent , param_value=encodeURIComponent(param)

About the parameter cardinality

A parameter can be optional or mandatory and it can occur once, or several times. In the web service documentation the convention is as follows:

Cardinality	Meaning
1	Mandatory. Only one occurence.
0|1	Optional. If exists, only one occurence.
1|N	Mandatory. Multiple occurences are possible.
0|N	Optional. Multiple occurences are possible.

When a parameter is expected once, and several are passed, only the first is used. The others are ignored.

```
/resources/modeler/pno/collabspace?pattern=*3DS*&pattern=*admin*
```

In the example just above, the second pattern parameter is ignored.

tenant query parameter

On Cloud environment, it is mandatory to append the `tenant` query parameter. Its value, case sensitive, is the platform instance.

Body

Usually POST , PUT and PATCH verbs request to pass the input data into a body. The format of the body is usually a **JSON** object.

```
{
    "param_name_1": "param_val_1",
    "param_name_2": {
        "param_name_21" : "param_val_21" , .
        },
}
```

Header Fields

There are five header fields to take into account for a 3DSpace HTTP request. Only the first is not conditional.

Field Name	Field value	Why
Accept	"application/json;charset=UTF-8"	The response format. Mandatory in all context
Content-Type	"application/json;charset=UTF-8"	The input body format. Mandatory if the HTTP request contains a JSON body.
SecurityContext	"RoleValue.OrganizationValue.CollaborativeSpaceValue"	The information to control the data storage, and access. Some 3DSpace web services may not require it, but usually it is mandatory. Without special mention in the API documentation, the security context is expected. For a safe use, the value must be UTF8 URL encoded. Indeed, a collaborative space name can contain chars with an ascii code above 255. Without an UTF8 encoding the REST Web service call will fail. Moreover, from a browser the failure can occur before sending the HTTP request since some browsers prevent such call.
DS-Request-ID	Usually an UUID	A client side value to identify the request. If this field is not set, the server returns its own value.
ENO_CSRF_TOKEN	A string returned by Get CSRF Token web service	This is the CSRF token to use for PUT, PATCH,POST and DELETE web services. The user group PN0 web services and the 6W web services require it. The token is valid all along the java session. After a new 3DSpace login, the token is no more valid.

Example of headers writen in JavaScript:

```
headers: {
    "Accept"          : "application/json;charset=UTF-8",
    "SecurityContext":  encodeURIComponent("Owner.My org2.My Collab Space") ,
    "Content-Type"    : "application/json;charset=UTF-8",
}
```

About response format

By default all 3DSpace web services support the JSON as response format. If the API can support an additional format, usually "application/xml", it is mentioned in the API reference page.

Response

The HTTP response is formed with three elements: a status code, a body, and header fields.

Status Code

3DSpace web services follow the RFC 7231 specifications:

- 2xx for a successful request,
- 3xx for a redirection,
- 4xx for a client side error,
- 5xx for a server side error,

Each API page documentation mentions the successful response status code, and the specific failure response status codes. The generic error cases are listed in the"3DSpace Web Service HTTP Failures" [4] article.

Body

The response body may or not exist, it depends on the API specifications. Its format is defined by the Accept header field. If this format is not compatible with the API specifications, the HTTP request fails with 406 (Not Acceptable) as status code.

Header Fields

Let's see those response header fields:

- Before processing a backend response, check the response body MIME type. In case of error, it may be different from the request `Accept` header field. It is mentioned with the `Content-Type` header field
- Each should web service should specify its cache control policy [5]. The `Cache-Control` header field conveys the information.
- `DS-Request-ID` is returned with the client side value, or if any, the backend one. The backend value is an UUID.

In Short

3DSpace exposes web services which are RESTful compliant. This article explains how to build up such a HTTP request, and how to understand the response. In case of divergence with DS rules, it is mentioned in the API documentation.

References

[1] RESTful Web Services & JAX-RS
[2] 3DSpace Role and Openness
[3] About Service URL and Platform Instance
[4] 3DSpace Web Service HTTP Failures
[5] HTTP Caching
[6] Case-Sensitive URL's
[7] Consuming 3DSpace Web Service
[8] Performing Service Login and Logout

History

Version: **2** [Aug 2023] Document updated for mentioning the query parameter tenant is mandatory on cloud
Version: **1** [April 2016] Document re-created

About 3DSpace Web Services

Technical Article

Abstract

3DSpace web services are RESTFUL defined with JAX-RS/JSON infrastructures [1]. This article recalls the common REST Web services principles and points to the 3DSpace specificities.

Protocol HTTPs

3DSpace web services enable you access to confidential data of both the company and the employees. Therefore 3DSpace web services use high security tools to prevent themselves from attacks [2]. HTTPS protocol is one of them. Consequently requests must be made using HTTPS protocol instead of HTTP to ensure security of data transfer.

URL

URL and Tenant

The format of the URL is expressed as follows:

```
https://<authority_server>
```

The contents of `authority_server` is slightly different between on-premises and cloud environments.

On-premises, there is only one platform instance [3], in other words one tenant. Let's say the URL is as follows:

```
https://<server_dependant>
```

On cloud, there are several platform instances [3], in other words several tenants. The URL is as follows:

```
https://<tenant_id><server_dependant>
```

About tenant identifier and case sensitivity

An URL is usually case insensitive [5]. It means whatever the case of `authority_server` it should work to reach the server. But with 3DSpace the case is important. Indeed 3DSpace authentication checks the tenant in the URL, and this check is case sensitive. Since the tenant identifier is an identifier of platform instance which can contain upper-case as lower-case letters, the HTTP request URL MUST exactly matches the expected URL.

Let's suppose the platform instance is `DSQAL005`

```
https://dsqal005-....3dx-xxx.3ds.com/enovia/resources/...  is wrong
https://DSQAL005-...3dx-xxx.3ds.com/enovia/resources/...  is correct
```

If you are not sure of the tenant identifier case in the URL, the solution is to use the `tenant` query parameter with the tenant identifier (I.e. platform instance) as value:

```
https://dsqal005-...3dx-xxx.3ds.com/enovia/resources/...?tenant=DSQAL005  is now correct
https://DSQAL005-...3dx-xxx.3ds.com/enovia/resources/...?tenant=DSQAL005  is still correct
```

Note: You do not have to test the current running environment (on-premises or on cloud), in all cases append the `tenant` query parameter. For on-premises solution (the value is "OnPremise") the parameter is just ignored by 3DSpace server APIs.

Retrieving 3DSpace URL from a widget app

Interactively, inside a widget application, the JavaScript API DS/i3DXCompassServices/i3DXCompassServices returns the 3DSpace URL(s).

Just below an extract of javascript code using the API to retrieve the 3DSpace URL for a given platform instance (tenant):

```
...
i3DXCompassServices.getServiceUrl( {
        serviceName: '3DSpace',
        platformId : widget.getValue("x3dPlatformId") ,
        onComplete : function (URL_3DSpace) {...}  ,
        onFailure  : ...
} ) ;
...
```

The referenced technical article [3] goes further with this API and the notion of platform instance.

Note: The 3DSpace URLs returned by this AMD module are always lower-case letters. Therefore if your code is multi-environment (cloud and onpremises), or cloud only, do not forget to append to the URL the *tenant* query parameter (*widget.getValue("x3dPlatformId")*) for reason mentioned above.

Below an example if the resource path does not contain a query parameter:

```
var myPATH=URL_3DSpace + resource_path + ?tenant=tenantID
```

And then if the resource path already contains a query parameter:

```
var myPATH=URL_3DSpace + resource_path + &tenant=tenantID
```

The referenced use case [7] may help you to be familiar with those API.

Conclusion

On cloud environment, use the `tenant` query parameter. Its value - case sensitive- is the plaform instance.

---

paths

The **root path** is the URL section between the 3DSpace URL, and the **application path**. The latter represents the relative path of the resource.

```
https://<3DSpace_authority_server><root_path><application_path>
```

For all 3DSpace web services, the root path is */resources*. Let's take an example: if the application path is /person, the final URL to reach a person resource is:

```
https://<3DSpace_authority_server>/resources/person
```

The combination of the root and the application path, is named the **path**.

```
https://<3DSpace_authority_server><path>
```

The 3DSpace web service API reference pages mention the URL path which always starts as follows: `/resources/...`

Login API

For a batch, before using a 3DSpace web service, a 3DEXPERIENCE platform and 3DSpace login must be performed. The technical article "Performing Service Login and Logout" [8] gives all the details.

The string to pass as value of the `service` query parameter for the 3DPassport login API is the CSRF token URL which the tenant (*XXX*) is appended for cloud solution.

```
String SpaceLoginURL=URLEncoder.encode(<URL_3DSpace>+ "/resources/v1/application/CSRF" + "?tenant=" + XXX, "UTF-8");
String PassportloginURL= <3DPassport> + "/login?service=" + SpaceLoginURL ;
```

For on premises, there are two solutions. Either with the tenant query parameter whose the value is "OnPremise" :

```
String SpaceLoginURL=URLEncoder.encode(<URL_3DSpace> + "/resources/v1/application/CSRF" + "?tenant=OnPremise" , "UTF-8");
String PassportloginURL= <3DPassport> + "/login?service=" + SpaceLoginURL ;
```

Or without the tenant query parameter:

```
String SpaceLoginURL=URLEncoder.encode(<URL_3DSpace> + "/resources/v1/application/CSRF", "UTF-8");
String PassportloginURL= <3DPassport> + "/login?service=" + SpaceLoginURL ;
```

Methods

The usual HTTP methods (also named verbs) and their meaning:

Verb	Usual Meaning
GET	for fetching of one or several resources
PUT	for updating a resource with full data

DELETE for deleting a resource
POST for creating a resource
PATCH for updating a resource with partial data

In each web service documentation, beside the resource path, the method is mentioned. Just below extract of List Persons:

Method	Path
GET	/resources/modeler/pno/person

Request

This section deals with the information to be proceeded by the server: the input parameters, the format for the inputs and the response, and the security context value.

The input parameters can be passed in the URL (as query parameters), and/or into a body. The other additionnal information are passed by header fields.

Query Parameters

A query parameter is part of the URL. The format is as follows:

```
<path>?<param_name_1>=<param_val_1>&<param_name_2>=<param_val_2>...
```

Append ? after the URL path to start the first parameter, and add & between each others. Below an example with the List Persons API

```
/resources/modeler/pno/person?pattern=*Admin*&select=firstname&select=email
```

About the parameter names

The parameter name is **case sensitive**. If the expected parameter is "select", but "Select" is passed, the parameter is not taken into account. It is considered as missing by the API. But the HTTP request only fails if the parameter is mandatory, otherwise the default behavior for the *missing* parameter is applied.

In the previous example, if 'pattern' is written 'Pattern', the HTTP request fails since 'pattern' is mandatory. But if 'select' is written 'Select', the response does not take into account the select clause parameters.

About the parameter values

Parameters are part of the URL, so their values must be **UTF8 URL Encoded**.

So for example (Get Collaborative Space) such URL is invalid:

```
/resources/modeler/pno/collabspace/3DS Collab Space
```

While this one is valid:

```
/resources/modeler/pno/collabspace/3DS%20Collab%20Space
```

Each space is replaced with "%20" .

There are functions to ensure the right format for the parameter values. Let's see two kinds of such functions:

- In java, with URLEncoder (java.net.URLEncoder), param_value=URLEncoder.encode(param,"UTF-8")
- In JavaScript, with encodeURIComponent , param_value=encodeURIComponent(param)

About the parameter cardinality

A parameter can be optional or mandatory and it can occur once, or several times. In the web service documentation the convention is as follows:

Cardinality	Meaning
1	Mandatory. Only one occurence.
0|1	Optional. If exists, only one occurence.
1|N	Mandatory. Multiple occurences are possible.
0|N	Optional. Multiple occurences are possible.

When a parameter is expected once, and several are passed, only the first is used. The others are ignored.

```
/resources/modeler/pno/collabspace?pattern=*3DS*&pattern=*admin*
```

In the example just above, the second pattern parameter is ignored.

tenant query parameter

On Cloud environment, it is mandatory to append the `tenant` query parameter. Its value, case sensitive, is the platform instance.

Body

Usually POST , PUT and PATCH verbs request to pass the input data into a body. The format of the body is usually a **JSON** object.

```
{
    "param_name_1": "param_val_1",
    "param_name_2": {
        "param_name_21" : "param_val_21" , .
        },
}
```

Header Fields

There are five header fields to take into account for a 3DSpace HTTP request. Only the first is not conditional.

Field Name	Field value	Why
Accept	"application/json;charset=UTF-8"	The response format. Mandatory in all context
Content-Type	"application/json;charset=UTF-8"	The input body format. Mandatory if the HTTP request contains a JSON body.
SecurityContext	"RoleValue.OrganizationValue.CollaborativeSpaceValue"	The information to control the data storage, and access. Some 3DSpace web services may not require it, but usually it is mandatory. Without special mention in the API documentation, the security context is expected. For a safe use, the value must be UTF8 URL encoded. Indeed, a collaborative space name can contain chars with an ascii code above 255. Without an UTF8 encoding the REST Web service call will fail. Moreover, from a browser the failure can occur before sending the HTTP request since some browsers prevent such call.
DS-Request-ID	Usually an UUID	A client side value to identify the request. If this field is not set, the server returns its own value.
ENO_CSRF_TOKEN	A string returned by Get CSRF Token web service	This is the CSRF token to use for PUT, PATCH,POST and DELETE web services. The user group PN0 web services and the 6W web services require it. The token is valid all along the java session. After a new 3DSpace login, the token is no more valid.

Example of headers writen in JavaScript:

```
headers: {
    "Accept"          : "application/json;charset=UTF-8",
    "SecurityContext":  encodeURIComponent("Owner.My org2.My Collab Space") ,
    "Content-Type"    : "application/json;charset=UTF-8",
}
```

About response format

By default all 3DSpace web services support the JSON as response format. If the API can support an additional format, usually "application/xml", it is mentioned in the API reference page.

Response

The HTTP response is formed with three elements: a status code, a body, and header fields.

Status Code

3DSpace web services follow the RFC 7231 specifications:

- 2xx for a successful request,
- 3xx for a redirection,
- 4xx for a client side error,
- 5xx for a server side error,

Each API page documentation mentions the successful response status code, and the specific failure response status codes. The generic error cases are listed in the"3DSpace Web Service HTTP Failures" [4] article.

Body

The response body may or not exist, it depends on the API specifications. Its format is defined by the Accept header field. If this format is not compatible with the API specifications, the HTTP request fails with 406 (Not Acceptable) as status code.

Header Fields

Let's see those response header fields:

- Before processing a backend response, check the response body MIME type. In case of error, it may be different from the request `Accept` header field. It is mentioned with the `Content-Type` header field
- Each should web service should specify its cache control policy [5]. The `Cache-Control` header field conveys the information.
- `DS-Request-ID` is returned with the client side value, or if any, the backend one. The backend value is an UUID.

In Short

3DSpace exposes web services which are RESTful compliant. This article explains how to build up such a HTTP request, and how to understand the response. In case of divergence with DS rules, it is mentioned in the API documentation.

References

[1] RESTful Web Services & JAX-RS
[2] 3DSpace Role and Openness
[3] About Service URL and Platform Instance
[4] 3DSpace Web Service HTTP Failures
[5] HTTP Caching
[6] Case-Sensitive URL's
[7] Consuming 3DSpace Web Service
[8] Performing Service Login and Logout

History

Version: **2** [Aug 2023] Document updated for mentioning the query parameter tenant is mandatory on cloud
Version: **1** [April 2016] Document re-created

CAS Log In

Role

Performs single sign on by generating a service ticket for a provided service URL.

Request

Method	Path
GET	/login

Query Parameters

Name	Required	Value	Description
service	NO	An URL service. The format is URL-encoded.	The URL of the service to be redirected with. The service URL must use a trusted domain declared in 3DPassport configuration.

Response

The response code can be:

- 302, a redirection

  The url of redirection can be:

  - If the request contains a valid CASTGC cookie, and has a valid url service as input

    The url of redirection is the input url service in adjonction with a service ticket.
  - If the request contains a valid CASTGC cookie, but there is no url service as input parameter, or an invalid one

    The url of redirection is my profile page of the 3DPassport
  - If the request does not contains a valid CASTGC cookie, but there is still a 3D Passport session.

    The url of redirection is my profile page of the 3DPassport
- 200,

  The request does not contain CASTGC cookie (or an invalid one), and there is no 3DPassport session. The response is the login page
- Error

Example

The web services has been launched while it is the first connection to the 3DPassport. There is no CASTGC cookie yet, and no 3DPassport session. The response code is 200, and the XML formatted response body is the login page.

```
[GET] <3DPASSPORT_URL>/login?service=<3DEXPERIENCE_SERVICE_URL_ENCODED>

##SERVER RESPONSE [200] OK
##--------------------

Has been redirected. Last Redirect URL : <3DEXPERIENCE_SERVICE_URL>?ticket=ST-1567-nElHZdbs4RO9WDhHrjXM-cas

#RESPONSE HEADER :
#--------------------
HTTP/1.1 200 OK
Date: Thu, 19 Oct 2017 14:32:54 GMT
Server: Microsoft-IIS/7.0
X-UA-Compatible: IE=edge
Content-Type: text/html;charset=UTF-8
Content-Length: 610
Set-Cookie:
Access-Control-Allow-Credentials: true
Access-Control-Allow-Methods: GET, POST, OPTIONS, HEAD, PUT, DELETE, PATCH
Access-Control-Allow-Headers: accept,x-requested-method,origin,x-requested-with,x-request,cache-control,content-type,SecurityContext,x-utc-offset,X-DS-CSRFTOKEN,X-DS-IAM-CSRFTOKEN,keep-alive,
Access-Control-Expose-Headers: X-DS-IAM-CSRFTOKEN
Access-Control-Max-Age: 600
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive

#RESPONSE BODY
#--------------------
<!DOCTYPE html>
<meta http-equiv="X-UA-Compatible" content="IE=edge" />

    <html>
    <head>
        <meta name="keywords" content="favicon, create, icon, favicon.ico, web, page, generate, bookmark, favorite, make, logo, site, internet, explorer, mozilla, firefox, convert, picture, i
        <link rel="shortcut icon" href="favicon.ico" type="image/x-icon" />
    </head>
    <body>
    <script language="javascript">
        //XSSOK
        document.location.href="common/emxSecurityContextSelection.jsp";
    </script>
    </body></html>
```

Get CSRF Token

Role

Generates a CSRF token valid for 6W and PN0 web services whose the verb is PUT, PATCH, POST or DELETE. The token is valid all along the java session. After a new 3DSpace login, the token is no more valid.

Request

Method	Path
GET	/resources/v1/application/CSRF

Responses

Success

The OK response (status code 200) returns those keys:

Key	Description	Value
		a JSON structure with those keys
csrf	CSRF token name/value	Key / Description / Value name / The CRSF token name. / "ENO_CSRF_TOKEN" value / The CSRF token value / String

Other keys are meaningless.

Example

```
##CLIENT REQUEST
##----------------------------------------------------------------
[GET] https://.../resources/v1/application/CSRF

##SERVER RESPONSE [200]
##--------------------

#RESPONSE HEADER :
#--------------------
HTTP/1.1 200
Date: Wed, 21 Nov 2018 14:25:28 GMT
Server: Microsoft-IIS/7.0
i3dx-requestid: 2a316af7-65df-433e-9be8-ecff6a3fc510
Cache-Control: no-cache,no-store,no-transform,must-revalidate,max-age=0
Content-Type: application/json
Content-Length: 143
Access-Control-Allow-Credentials: true
Access-Control-Allow-Methods: GET, POST, OPTIONS, HEAD, PUT, DELETE, PATCH
Access-Control-Allow-Headers: accept,x-requested-method,origin,x-requested-with,x-request,cache-control,content-type,SecurityContext,x-utc-offset,X-DS-CSRFTOKEN,X-DS-IAM-CSRFTOKEN,keep-alive,
Access-Control-Expose-Headers: X-DS-IAM-CSRFTOKEN
Access-Control-Max-Age: 600
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive

#RESPONSE BODY
#--------------------
{"success":true,
 "statusCode":200,
 "csrf":{
    "name":"ENO_CSRF_TOKEN",
    "value":"D3KG-QXIU-MTS9-6490-STAL-9WTZ-BQBU-E5Y1"
 },
 ...
}
```

6W Web Service Principles

Technical Article

Abstract

This document describes common behavior and general implementation details for 3DEXPERIENCE web services that are developed using these principles. If a 3DEXPERIENCE web service identifies 6W web service principles as its development methodology, then this document will provide additional details as you consume these web services.

But first, the reader should read About 3DSpace Web Services [1] article.

Overview

Please read this technical document in its entirety as it provides the basis for ENOVIA rest services that are developed following the 6W Web Service Principles. While the Swagger rest documentation provides the payload and request details for each end point, the following documentation provides general and usage information that will help you as you consume these rest services.

Authentication / SecurityContext

All rest service end points must be called with a valid authenticated user. Please refer to the 3DPassport APIs as needed for authentication.

In addition, the request can optionally identify the Security Context that should be considered while executing the rest service.

Security Context is typically not required for read operations, but may be required for write/create operations to specify the object's primary ownership vector (POV) information.

The security context must be valid and also associated to the context user. An invalid security context will result in a 401 return code.

To pass the security context for a given request, it has to be specified using request header field.

Field Name	Field value	Remarks
SecurityContext	"RoleValue.OrganizationValue.CollaborativeSpaceValue"	For a safe use, the value must be UTF8 URL encoded. Indeed, a collaborative space name can contain chars with an ascii code above 255. Without an UTF8 encoding the REST Web service call will fail. Moreover, from a browser the failure can occur before sending the HTTP request since some browsers prevent such call.

Locale / Language

By default, error message and/or NLS values when applicable will be returned in English. However, the language can be specified as part of the request by specifying the language in the header.

Field Name	Field value	Remarks
Accept-Language	the user language	The language must be supported by the **3D**EXPERIENCE; otherwise, English will still be the default.

Payload

The supported payload is based on JSON. The same payload structure and properties are typically used for both read and write operations, although certain properties are read only or create only which will be indicated in the reference documentation. Invalid or read-only properties are ignored during update operations.

A typical response, whether read or write, provides a number of properties to indicate success, return/status code, csrf token and an array of data objects which will be further explained below.

```
{
    "success": true,
    "statusCode": 200,
    "csrf": {
        "name": "ENO_CSRF_TOKEN",
        "value": "K72A-44UG-MPZD-I0MR-5TYW-ZHH8-4DZB-BEYC"
    },
    "data": [
        dataobject1,
        dataobject2,
        ...
    ]
}
```

CSRF

CSRF stands for Cross-Site Request Forgery and is required for update operations, unless stated otherwise, in support of Web Security best practices.

The CSRF token will be returned in the JSON payload when a GET operation is performed. The same CSRF token returned in the GET request must be sent back when an update operation is performed. In the case, the client does not have the token; e.g., a GET request has not been performed, the CSRF token can be specifically obtained using the following rest service end point:

```
<SERVICE_URL>/resources/v1/application/CSRF
```

The CSRF token returned by the rest service as follows:

```
{
  "csrf": {
    "name": "ENO_CSRF_TOKEN",
    "value": "EXAM-PLEW-R1CD-BNF7-G8UC-KT52-MGUA-UBE1"
  }
  ...
}
```

While the CSRF token is returned in the JSON response, it can be sent back during an update operation in the header or the JSON payload.

Field Name	Field value	Remarks
ENO_CSRF_TOKEN	EXAM-PLEW-R1CD-BNF7-G8UC-KT52-MGUA-UBE	example value

If an invalid CSRF token is passed or not provided altogether during an update operation that requires a CSRF token, a 403 error along with the valid CSRF token are returned.

Data Objects

The data property identifies the data objects returned by the rest service. The data property is always an array of objects, even if the response contains just one object. Likewise, for update operations, the data is also sent as an array even if only one object is being modified.

Depending on the rest service end point, the data object is composed of a number of basic properties, data elements, rel elements, children and related objects. Not all services will include all these properties, but may be included as applicable. Also, while the data objects typically represent actual business objects inside ENOVIA, they don't have to be actual business objects but simply an object/resource and sub resources returned by the rest service.

- basic properties: id, type, etc.
- data elements: additional properties and attributes associated with the resource
- rel elements: additional "relationship" properties and attributes associated with the resource/sub resource
- children: In case the resource is structured, the rest service could return the structure using the children property
- related data: sub resources or related data associated to the resource. The sub resources are identified by a property key and the values is an array of data objects
- Example:

```
{
    "data": [
        {
            "id": "string",
            "relid": "string",
            "cestamp": "string",
            "type": "string",
            ...
            "dataelements": {
                "property1": "string",
                "property2": ["string", "string2"],
                ...
            },
            "relelements": {
                "property1": "string",
                "property2": ["string", "string2"],
                ...
            },
            "relateddata": {
                "subresource 1": [
```

```
                        dataobject1,
                        dataobject2,
                        ...
                ],
                "subresource 2": [
                        dataobject1,
                        dataobject2,
                        ...
                ]
            },
            "children": [
                dataobject1,
                dataobject2,
                ...
            ]
        },
        dataobject2,
        dataobject3,
        ...
    ]
}
```

The specific resource properties and sub resources (related data) is based on the rest service end point that is executed. Also, depending on the rest service end point, the sub resource data may be directly retrieved or updated for a given resource id. In addition, newer rest service implementations using this methodology may not contains dataelements and relateddata properties, but more of a flat JSON structure where the dataelements and relateddata properties are provided on the same dataobject directly.

Certain data properties and related data information may be available but not returned by default unless specifically requested. See Data Refinement section for specifying the return data required by the client.

Data Refinement

The data returned by the rest service can contain various data elements and related data depending on the rest service. While the rest service will optimally retrieve the data, the response time will be based on the amount of data being retrieved. As a consumer of these rest services, the client has control on the data being returned by specifying the data properties and related data that should be retrieved.

To refine the returned data, the following URL (query) parameters can be specified as needed:

- $include: Identified the related data objects to include or exclude
- $fields: Identifies the data/rel elements to include or exclude

When identifying the $include & $fields parameters, the "all" and "none" keywords can be specified.

- $include=all: Includes all available related data information declared by the rest service even if turned off by the rest service by default. This option should not be used in actual implementations as the rest service can expand and grow and will negatively impact performance.
- $fields=all: Includes all available elements declared by the rest service even if turned off by the rest service by default. This option should not be used in actual implementations as the rest service can expand and grow and will negatively impact performance.
- $include=none: Excludes all available related data information from being returned by the rest service
- $fields=none: Excludes all available elements from being returned by the rest service

In addition to the "all" and "none" keywords, the name of the related data and property names can be used to include or exclude. To exclude a given related data or property, it can be prefixed by an "!" exclamation point before the name. Further, multiple items can be specified for inclusion and exclusion by using "," comma as a separator.

Examples:

- $include=relateddata1: All the default related data will be returned and also "relateddata1" in case it was turned off by default
- $include=relateddata1,relateddata2: All the default related data will be returned and also "relateddata1" & "relateddata2" in case they were turned off by default
- $include=!relateddata1: All the default related data will be returned except for "relateddata1". Other related data that is turned off by default will still be off
- $include=all,!relateddata1: By specifying "all", all related data will be turned on and then exclude "relateddata1" as it is prefixed by !
- $include=none,relateddata1: By specifying "none", all related data will be turned off and then include "relateddata1"

The same concept can be applied to the properties using the "none", "all", "!" and comma separated list of entities for the $fields parameter. However, since properties may be redundant across sub resources (related data), to identify the properties, it should be identified using the related data as a prefix; e.g. "relateddata1.property1". Even of the resource is many levels deep, the immediate sub resource for the property can be used to identify the field. If the sub resource is not identified, the property is assumed to be on the main resource. Any invalid entries provided for $include and $fields will be ignored.

While the rest service may be returning all the potential data by default, specifying "all" for both parameters will return all available properties and related data. However, if the data is not required by the client, it is best to include what is required or exclude what is not required. In any case, the keyword "all" should not be used for actual implementations as noted above.

Finally, specifying "none" for the fields will have no affect on the basic properties such as id, type, etc. This also enables the client to retrieve minimal information like IDs and then subsequently retrieve the additional data associated to one or more IDs later based on the UI design.

Data Object Updates

The rest service end points may support various HTTP verbs for creating, modifying and deleting data. In general, the following verbs are used as such:

- POST: Used to create one or more data objects
- PUT or PATCH: Used to modify one or data objects
- DELETE: Used to delete one or data objects

Note, the HTTP verb/action applies to all the data objects specified in the payload including sub-resources. However, the HTTP verb/action can be overwritten per data object using the property "updateAction".

The following update actions are supported when specified for a given data object:

- NONE: No update is performed on the resource object in terms of its data elements; related and children data objects will still be updated based on their update actions.
- CREATE: Create the specified resource/object.
- MODIFY: Modify the specified resource/object.
- DELETE: Delete the specified resource/object.
- CONNECT: Connect a sub resource to the parent resource; can also be specified using CREATE to create the link/connection between the resource and sub resource.
- DISCONNECT: Disconnect a sub resource from the parent resource; can also be specified using DELETE to delete the link/connection between the resource and sub resource.

Mapping of HTTP Verbs to Update Actions

- POST: updateAction="CREATE"
- PUT: updateAction="MODIFY"
- PATCH: updateAction="MODIFY"
- DELETE: updateAction="DELETE"

The "updateAction" property is convenient and sometimes necessary when performing updates of resource and sub resources data. It is convenient in that the consumer of these rest services can use a single HTTP verb for all updates like POST or PUT and override the action on a per object basis. It is necessary in certain situations, especially to specify NONE for no action when the intent is not to modify or create the main resource, but to update a sub-resource.

Example:

```
{
    "data": [
        {
            "id": "123",
            "updateAction": "NONE",   <==>      the resource will not be modified by identified by its "id" as context for the sub resource
            "relateddata": {
                "subresource 1": [
                    {
                       "updateAction": "CREATE",  <==>    the sub resource will created and linked to the parent resource identified by its "id"
                       "dataelements": {
                          "property1": "test value"
                       }
                    },
                    {
                       "id": "1234",   <==>       the sub resource associated to the parent "id" will modified
                       "updateAction": "MODIFY",
                       "dataelements": {
                          "property1": "test value"
                       }
                    }
                ],
                "subresource 2": [
                    {
                        "id": "1234",
                        "updateAction": "DELETE"   <==>  this will typically disconnect the resource unless it is a composition object in which case it will also be deleted.
                     }
                ]
            }
        },
        {
            "updateAction": "CREATE",    <==>   Create a new resource
            "dataelements": {
                "property1": "test value"
            }
        }
    ]
}
```

In summary, the "updateAction" is optional to override the HTTP verb based on the mappings above.

Response Codes

The rest services will typically result in one of these return codes:

- 200: The request was successful
- 400: The request failed due to some error condition check. A valid and translated user-friendly error message is returned
- 401: The request failed due to non-authentication or timed out request, invalid Security Context or invalid or missing CSRF token
- 500: The request failed unexpectedly due to some exception encountered. A generic user-translated error message is returned along with a technical, non-translated error message that may serve as a hint for the failure

Temporary ID (tempId)

Temporary id is a feature that provides 2 primary functions during a POST (Create) operation. When creating data, an ID will be assigned to the newly created resources and sub-resources as necessary. However, the client can supply an "id" in the payload for the following reasons:

- The "id" supplied during create of a resource/sub-resource will be returned as "tempId" to provide the client a mapping between the supplied id and the actual id that was assigned.
- In addition, it is possible that certain objects may need to reference other objects in the payload that are also being created at the same time. In this case, the client can assign their own unique identifier (id) and reference it in the payload. For example, creating 3 tasks in one payload and establishing dependencies/predecessors at the same time which requires the predecessor task id. Since all tasks are created at once, the temporary id can be assigned to each task resource and referenced to indicate the predecessor id.

Note, once the object/resource is persisted in the database, the actual id must be used.

Data Model Specialization Support

3DEXPERIENCE provides a number of mechanisms to specialize the OTB data models with custom attributes, custom extensions and custom subtypes. These capabilities are enabled using the following functions: Data Model Specialization (DMS - Cloud), Data Model Customization (TXO - On Premise), Attributes Management (Both Cloud & On Premise).

These data model customizations may be supported by the some of the REST services based on this framework. Each REST service should indicate the type of support enabled.

In general, when the REST service supports such specializations, it means the REST service can be used to retrieve as well as set these custom attributes during create and modify.

The following behaviors can be expected based on the type of specialization performed:

- DMS & TXO:
  - Deployment/Automatic Extension Attributes are reflected and retrieved automatically when performing GET request on the resource
  - Subtype attributes are reflected and retrieved automatically when performing GET request on the resource
  - Specialization Extension Attributes are reflected using $fields when performing GET request on the resource; since these attributes are optional, they are not retrieved unless requested.
  - In addition, since specialization extensions are optionally applied to the resources, a new JSON property (array) called "extensions" can be utilized to add and remove extensions
  - All custom attributes can be specified during create; if the attribute is a subtype attribute, the type property needs to reflect the subtype so that a subtype resource is created.
  - When specifying specialization attributes during create, the extensions property needs to identify the extension(s); e.g. "extensions" : ["specialization1", "specialization2"]
  - When modifying a resource with adding or removing a specialization extension, the extensions property can be used to add or remove the extensions and prefixing the action with the extension name; e.g. "extensions" : ["add specialization1", "remove specialization2"]. the "add" prefix can also be specified during CREATE, although it is implied
  - TXO Applicable Only: Deployment extensions that are not automatically deployed can be retrieved using read operations by specifying $fields and updated using the PUT/PATCH payload, but these extension cannot be added or removed through the REST service via the "extensions" JSON property.
- Attribute Management:
  - Custom Type Attributes are reflected automatically when performing GET request on the resource
  - Custom Type Attributes can be specified during Create and Modify operations

Experiencing 6W Web Services

We provide a client batch (a java client application) which in this order:

- Perform **3D**EXPERIENCE login (username and password are batch inputs)
- Get CRSF Token value
- Consume 3DSpace web services whose verb, path, input data are declared in a text file (batch input). The CSRF token is passed at each HTTP request.
- Perform **3D**EXPERIENCE logout

This batch is *CAAi3dxScenarioWithCSRFMain*. All HTTP requests (and their responses) are logged in trace files. How to find the code, how to launch the batch and how to understand its inputs and outputs refer to the [2] referenced article.

References

[1] About 3DSpace Web Services
[2] Experiencing 3DSpace Web Services

History

Version: **3**	March 2021	Minor updates
Version: **2**	November 2020	Minor updates, temporary id
Version: **1**	October 2018	Document created

3DSpace Role and Openness

Technical Article

Abstract

3DSpace allows people to create and manage collaborative spaces that are dedicated to content storage and collaboration on content. This article is an overview of its capabilities, and what sort of openness it proposes.

3DSpace is available on **cloud** and **on-premise**. Most modelers are deployed to both deployment types. If there are any exceptions, they are specified in the About Web Services article beneath the modeler section.

Role

3DSpace provides the following key capabilities:

- collaborative space creation,
- collaborative space properties edition,
- inviting people to collaborative spaces,
- content management in collaborative spaces,
- collaborative space privileges administration,
- compassing 3DSpace content.

The reader will find information about 3DSpace as Social and Collaboration Applications in the User Assistance documentation. The node is "Social and Collaborative | 3DSpace".

Good to know: In 3DSpace the user login name (also named username) is case sensitive, while case insensitive for 3DPassport.

Platform Isolation

3DSpace is based on the multi-tenancy [1] principle. Through the **3D**EXPERIENCE platform, a tenant is also named a **platform instance** [2].

On the public cloud solution, there are as many platform instances as customer having pay for one instance. Otherwise, for the private cloud and on-premises solutions, there is only one platform instance. For the latter case it is named 'onPremise'.

A user must be first granted to be member of a platform instance. It is the role of the **3D**EXPERIENCE platform administrator to grant people.

So, there are API [2] to get the platform instances for which the connected user is member, and to get for each platform instance the valid 3DSpace URL.

Secured Access

Like expressed by this whitepaper on our www.3ds.com web site [3], the security is on the heart of the **3D**EXPERIENCE platform. When using 3DSpace you have to be familiar with three points of security:

- **Authentication**

  3DSpace, like all **3D**EXPERIENCE platform services, requires an authentication. The protocol is based on the CAS protocol, with a centralized management through the 3DPassport [4]. So, before using 3DSpace you must perform a login [5].
- **Cross-domain**

  All browsers recommended by the **3D**EXPERIENCE platform prevent cross-domain access. The **3D**EXPERIENCE servers solve this security point by implementing CORS.

  CORS (Cross-Origin Resource Sharing) enables secured, and controlled, cross-domain communications from the browser. For example it enables *direct access* to 3DSpace web services inside a widget from 3DDashboard. Let's see the referenced article [6] dealing with HTTP requests from a widget application.
- **Security Context or Credentials**

  This point is 3DSpace specific. In order to secure the data on 3DSpace, credentials are requested when creating or editing them. The credential is based on three components: a collaborative space, an organization and an access role. This triplet is usually named the *security context*. It controls the content that user can access.

  These three components are defined by the People and Organization modeler [7] which exposes API to retrieve user credentials.

  If you are not familiar with the concept read in the User Assistance documentation the article "Installation and Administration | 3DEXPERIENCE Platform | 3DEXPERIENCE Platform - Administration | 3DSpace Administration | People & Organizations and Content - Administration | Managing Persons, Roles, Organizations, and Collaborative Spaces | User Working Environment"

Openness Agent

3DSpace service provides the capability to call its web services on behalf of someone using an openness agent [11]. This agent can also be used to subscribe to messages sent when modeler/application events are triggered.

It is only a **cloud** deployment capability.

Web Services

3DSpace exposes its resources through web services implementing the REST architecture [8]. You retrieve the exposed API below the "Web Service References" node of each modeler. Before using them, you are invited to read "About 3DSpace Web Services" [9]. The article is gathering the common rules for all 3DSpace web services.

Besides, for your own data model, or for specific reasons, it can be necessary to develop your own web services. We recommend to use the infrastructure based on JAX-RS and JSON and developed on top of 3DSpace [10]. The main advantage is to take benefit of the API checking the authentication, and returning the matrix context. This context is mandatory for handling 3DSpace resources.

Below in the Developer Assistance documentation you find use cases and articles dealing with 3DSpace web service consumption. Let's see the Consuming 3DSpace Web Services node introducing the Widget application case, and detailing an example of batch.

The internal name of the service is: **3DSpace.**

Events

3DSpace modelers/applications publish events as explained in the article named Events [12].

About *authorization* key you find in the *data JSON* [13]. its value is the security context value.

There are some spefiic behaviors about events from 3DSpace to consider when designing around them:

- Events are not yet supported during data import operations in 3DSpace.
- Some automated operations, such as the Configuration Center options to automatically share newly created content with other users, do not send status changed events. Only the created event will be sent in this case.
- Event processing in 3DSpace relies on transaction triggers. So, any operation which disables triggers during a transaction will not send any events for that transaction.

The topic name is **3dsevents.$tenantID.3DSpace.<user|admin>** where $tenantID is the platform instance. The final value (user or admin) is explained in the article Events [12].

6WTags

The 3DSpace web services do not return 6WTags.

Licensing

3DSpace requires at least the Collaborative Industry Innovator (CSV) role. Some modelers may require additional roles. It is mentioned in the licensing section of the modeler articles.

References

[1] Multi-tenancy
[2] About Service URL and Platform Instance
[3] Cloud Security Whitepaper
[4] 3DPassport and CAS Protocol
[5] Consuming 3DSpace Web Services
[6] About Widget and HTTP Request
[7] People And Organization Modeler
[8] REST Web Services
[9] About 3DSpace Web Services
[10] Creating a 3DSpace REST Web Service (article available in ENOVIAStudioApplicationJavadoc.Linux64_package)
[11] Openness Agent
[12] Events
[13] Event Message Data Format

History

Version: **1** [October 2015] Document created

3DSpace Web Service HTTP Failures

Technical Article

Abstract

This article lists the common HTTP failures when invoking a 3DSpace Web Service.

Proxy

This article, as well each API web pages, gives the HTTP failure code as returned by the API. But all these errors can be transformed by a proxy installed on your server, or on the client side. There is an example with the 3DDashboard proxy for web services called from a widget.

Failure Cases

The 3DSpace web services can fail for many reasons. The errors, grouped together in tables, have been sorted by logical code level. It starts with the generic HTTP errors, then the generic 3DSpace Web services, and finally the specific cases. Each table gives for an error its HTTP backend response. As usual the response is made of of two elements: the status code, and the potential content of the body.

Generic HTTP Failure Cases

The following table lists the most frequent errors.

Status Code	Message	Body Content	Reason
404	Not Found	an html text.	Invalid URL; check the path: • Its case. • There is no double "/" char except the one after the protocol.
			The error is raised by the REST WS infrastructure
		Depends on the WS	The URL does not match an existing resource. Usually the identifier of the resource is invalid. The error is raised by the applicative web service. See the WS API documentation.
406	Not Acceptable	No	The request "Accept" header field is incompatible with the API response format(s).
415	Unsupported Media Type	No	The request "Content-Type" header field is not the one expected by the API.

Generic 3DSpace Failure Cases

The following table lists all the errors specific to 3DSpace web services.

Status Code	Message	Body Content	Reason
403	Forbidden	an HTML page containing at least this message "Access to this server is not granted to user"	The user does not exist for 3DSpace URL
401	Unauthorized	A 3DSpace Infra Error body content: `{"error": {"code": "INFXXX", "message": "..."}}` where: • INF100 if unknown or expired session • INF101 if impossible to apply given security context • INF102 if the security context is missing in the request • INF103 if the security context value is empty • INF104 if the security context cannot be cleared • INF105 if the security context cannot be retrieved. It may by a license issue.	no authenticated session with 3DPassport, or an invalid/missing security context.

Specific 3DSpace Failure Cases

The main specific to 3DSpace web services are:

Status Code	Message	Body Content	Reason
400	Bad Request	3DSpace Infra Error	The error comes from the client side: examples: a mandatory input parameter is missing, an input parameter value is wrong, ...
500	Internal Server Error	Usually no content.	There is a server side error.

3DSpace Infra Error:

The content type of the response is JSON:

```
{"error": {
    "code": "XXXNumber",
    "message": "..."
  }
}
```

Where:

- `code`: a string identifying the error.
  - XXX is a 3-letter uppercase prefix identifying the category.
  - Number is a number
- `message`: a free text message

History

Version: **2** [April 2016] Document completed
Version: **1** [Feb 2016] Document created

Performing Service Login and Logout

Technical Article

Abstract

The first sections of this article detail the steps to first perform a login to a **3D**EXPERIENCE service, and then a logout. The last section gives the reference of the use case performing such scenario for a given **3D**EXPERIENCE service.

This article describes a procedure where:

- The 3DPassport URL is known
- The java class performing HTTP requests follows the re-directions

The second article Performing Service Login (other way) [5] describes another way to perform the login without previously knowing 3DPassport URL, and without following HTTP request redirection.

Foreword

Web services are executed with the *CAAURLLoader.java* class through a unique class instance during the batch execution. About this class:

- It uses the HTTP Client java class for executing the HTTP requests.
- It stores the cookies.
- **It follows the URL re-directions.**

This class instance must be unique to keep in session all the cookies created from different servers. Client side they are stored and managed by a cookie manager. For each HTTP request to a given server the cookie manager sends additionnaly all cookies coming from this target server.

In the next sections, *loadUrl* is a method of the *CAAURLLoader.java* class to launch an HTTP request. Its definition is as follows:

```
public byte[] loadUrl(String urlAsString,
                      String method,
                      String content_type,
                      byte[] post_data)
```

Where:

- urlAsString, is the web service URL.
- method, is the web service verb (GET,POST,...).
- content_type, is the request body format (application/json,...) .
- post_data, is the request body message.

And the way to retrieve the unique class instance is as follows:

```
_client.getURLLoader()
```

Where `_client` is an instance of the *CAAClient.java* class keeping the unique class instance. The client class is created in the batch main class.

The piece of code presented in the next sections are extracted from the *CAALoginLogoutClient.java* class. This class uses the *CAAURLLoader.java* class instance to execute the HTTP requests.

All mentioned CAA* java classes are located in the CAAPassportCommonClient.mj module of the CAA3DPassport.edu framework.

The article Presenting Batch Client Application Example [6] explains these classes.

Login

The login for a given **3D**EXPERIENCE service contains three steps.

Retrieving a Login Ticket

The first step is to retrieve a login ticket from the passport. We execute the /login REST web service with **action** as query parameter. It is a GET verb.

```
String Passport_URL_service = _client.getPassportServiceURL() + "/login?action=get_auth_params" ;
response = _client.getURLLoader().loadUrl(Passport_URL_service,"GET", null,null);
```

If the response is successul with "200" as code response, the response body is a JSON as follows:

```
{"response":"login","lt":"LT-6906-qCGbyLk7FFA3KjCJUwwJ9jQm0ru7rd"}
```

In the response there is the JSESSIONID cookie associated with passport service. The same cookie is used from the login until the logout for the same "client" session.

CAS Authentication

Then we perform the CAS authentication still against the passport. We execute the /login REST web service with POST as verb. The body message (form format) contains the login ticket (extracted from the previous step), and the user login and password. The latters must be URL encoded since some login, and password can contain special chars.

```
String post_data_str ;
String usernameEnc= URLEncoder.encode(username,StandardCharsets.UTF_8.toString()) ;
String passwordEnc= URLEncoder.encode(password,StandardCharsets.UTF_8.toString()) ;
post_data_str = "lt=" + lt + "&username=" + usernameEnc + "&password=" + passwordEnc ;
byte [] post_data = post_data_str.getBytes();

Passport_URL_service= _client.getPassportServiceURL()  + "/login";
String encodageForm="application/x-www-form-urlencoded;charset=UTF-8" ;
response= _client.getURLLoader().loadUrl(Passport_URL_service, "POST" , encodageForm, post_data);
```

The authentication is successful if the response consists of the "my-profile" page loading. It means checking the last redirected URL is the "my-profile" page.

```
String last_dirURL=_client.getURLLoader().getLastRedirectUrl();
boolean issue=true;
if ( ( _client.getURLLoader().getResponseCode() == 200) && (last_dirURL != null) ) {
    int index=last_dirURL.indexOf("/my-profile");
    if ( index != -1 ) issue=false ;
}
```

The test consists in to check the path of the last redirected URL contains the `/my-profile` string.

In the response there is the authentication cookie associated with passport service, the CASTGC cookie. It is used from the login until the logout for the same user, and the same client session.

CAS Service Login

Finally we can perform the login to one or several **3D**EXPERIENCE services. We execute the /login REST web service with **service** as query parameter. The value of this query parameter is an (login) URL of the **3D**EXPERIENCE service which as usual must be UTF8 URL encoded.

```
String TargetService=URLEncoder.encode(_client.get3DEXPERIENCELoginAPI() ,StandardCharsets.UTF_8.toString())
Passport_URL_service= _client.getPassportServiceURL() + "/login?service=" + TargetService ;
response= _client.getURLLoader().loadUrl(Passport_URL_service, "GET", null, null);
```

In case of successful login, the response is the response of the service passed as query parameter. The generic test consists in checking the last redirected URL contains "ticket" as query parameter.

```
last_dirURL=_client.getURLLoader().getLastRedirectUrl();
issue=true;
if ( ( _client.getURLLoader().getResponseCode() == 200) && (last_dirURL != null ) ) {
    int index=last_dirURL.indexOf("ticket=");
    if ( index != -1 ) issue=false ;
}
```

In the response there is the SESSIONID cookie associated with **3D**EXPERIENCE service. It is used from the service login until its logout for the same user, the same client session, and the same service provider. If the service provider session expires for a user, his/her SESSIONID is invalided, this step (and only this one) should be redone.

Notes:

- The two and third steps can be merged. In this case, you add the "service" query parameter with the POST verb, and the expected successful response is the one of the third step.
- The **3D**EXPERIENCE service *login* URL is depending on each service.
  - 3DSpace
  - 3DSwym
  - 3DCompass

After this step you can safely execute all web services, whatever their VERB (POST/PUT/DELETE/PATCH) on the **3D**EXPERIENCE service for which a such login has been performed. For more information about the login URL role, read the "Login API Issue" section in the referenced article [4].

Logout

The logout is unique for the session. We execute the /logout REST web service. It is a GET verb.

```
String Swym_logout = _client.getPassportServiceURL() + "/logout" ;
byte[] response = _client.getURLLoader().loadUrl(Swym_logout, "GET", null,null);
```

Checking the success of the logout - as we made - depends on the service has implemented CAS.

In mode **no gateway** as 3DSpace, 3DPassport loads its login page.

```
boolean logout_successful=false;
if ( ( _client.getURLLoader().getResponseCode() == 200) {
    String last_dirURL=_client.getURLLoader().getLastRedirectUrl();
    if  (last_dirURL != null ) {
       int index=last_dirURL.indexOf("/login");
       if ( index != -1 ) logout_successful=true ; // great -> 200 and Login page
    }
}
```

In mode **gateway** as 3DSwym, 3DPassport redirects to the service. The latter should return a failure response code.

```
if ( client.getURLLoader().getResponseCode() => 400   ) {
    logout_successful=true ; // great -> the service has returned an error
}
```

Use Cases

Mono Connection

The use case is a client java application implying one **3D**EXERIENCE server (beside 3DPassport). Proceed as follows:

- **Build the three modules** of the CAA3DPassport.edu framework. No other framework is pre-requested.
  - CAAPassportCommonClient.mj
  - CAAPassportLoginLogout.mj

You need jakartaee-api9.jar for building them.

- **Set (export for unix env) the CLASSPATH variable** with the jar file paths:

```
set CLASSPATH=<WSPATH-RTV1>johnzon-core-jakarta.jar ;
set CLASSPATH=%CLASSPATH:<WSPATH-RTV1>jakartaee-api9.jar;
set CLASSPATH=%CLASSPATH:<WSPATH-RTV2>CAAPassportLoginLogout.jar;
set CLASSPATH=%CLASSPATH:<WSPATH-RTV2>CAAPassportCommonClient.jar;
```

Where <WSPATH-RTV2> is the path of your workspace runtime view and <WSPATH-RTV1> is the path where you have located the none provided jar.

- **Launch the exe** as follows:

```
java com.dassault_systemes.caasamples.passport3ds.CAALoginLogoutMain 3DPassport_URL login password 3D_Service_URL Login_Path API_Path output_dir
```

Where

argument	meaning
3DPassport_URL	The 3DPassport URL of a **3D**EXPERIENCE platform.
login	A valid login for the given **3D**EXPERIENCE platform.
password	The password of the login.
3D_Service_URL	The 3DEXPERIENCE service URL.
Login_Path	The path appended to 3D_Service_URL represents the URL used to perfom the service login.
API_Path	The path appended to 3D_Service_URL defines an URL to execute. It must be a GET web service.
output_dir	Directory for traces of HTTPS requests and responses.

This batch:

- Performs the service login: CAS Authentication + login to the service, and checks if the step successful.
- Executes the <3D_Service_URL><API_Path> web service.
- Performs the 3DEXPERIENCE platform logout, and checks if the step is successful.
- Re-executes <3D_Service_URL><API_Path> web service, and checks if the execution has not been a success.

For the following input data (Cloud and 3DSpace):

```
## Getting input arguments :
 catch 3DPassport URL=<3DPassportURL>
 catch user login=toto
 catch user password=...
 catch 3DEXPERIENCE Service URL=<3DSpace_URL>
 catch Service login path=/resources/modeler/pno/person?current=true&tenant=<tenantvalue>
 catch Service API path=/resources/v1/application/CSRF?tenant=<tenantvalue>
 catch output directory=e:\temp
## End Of arguments catching
```

It generates the following trace files:

- login.traces
- logout.traces
- API_EXE_BeforeLogout.traces
- API_EXE_AfterLogout.traces

Multi Connections

The use case is a client java application implying two **3D**EXERIENCE servers (beside 3DPassport). Proceed as follows:

- **Build the modules** of the CAAi3DXWebServicesClient.edu framework. No other framework is pre-requested.
  - CAAi3DXJavaClient.mj
  - CAAi3DXMultiServicesScenarioBatch.mj

You need jakartaee-api9.jar for building them.

- **Set (export for unix env) the CLASSPATH variable** with the jar file paths

```
set CLASSPATH=<WSPATH-RTV1>johnzon-core-jakarta.jar;
set CLASSPATH=%CLASSPATH:<WSPATH-RTV1>jakartaee-api9.jar;
set CLASSPATH=%CLASSPATH:<WSPATH-RTV2>CAAi3DXMultiServicesScenarioBatch.jar;
set CLASSPATH=%CLASSPATH:<WSPATH-RTV2>CAAi3DXJavaClient.jar;
```

Where <WSPATH-RTV2> is the path of your workspace runtime view and <WSPATH-RTV1> is the path where you have located the none provided jar.

- **Launch the exe** as follows:

```
java com.dassault_systemes.caasamples.i3dxsamples.CAAi3dxScenarioMultiMain  3DPassport_URL login password 3DSpaceURL 3DCompassURL output_dir  ScenarioFilePath [tenant]
```

Where

argument	meaning
3DPassport_URL	The 3DPassport URL of a **3D**EXPERIENCE platform.
login	A valid login for the given **3D**EXPERIENCE platform.
password	The password of the login.
3DSpaceURL	The URL for the first 3DEXPERIENCE service.
3DCompassURL	The URL for the second 3DEXPERIENCE service.
output_dir	Directory for traces of HTTPS requests and responses.
ScenarioFilePath	The file containing resource paths to execute for 3DEXPServ1 or 3DEXPServ2. Each web service is the concatenation of one service URL (3DEXPServ1 or 3DEXPServ2) with an URL path.
tenant	The platform instance if the cluster is Cloud

This batch:

- Performs the service logins:
  - CAS Authentication
  - login to the first service, and checks if the step successful.
  - login to the second service, and checks if the step successful.
- Executes web services defined in ScenarioFilePath file.
- Performs the 3DEXPERIENCE platform logout, and checks if the step is successful.

For the following input data (onPremise and 3DSpace and 3DCompass):

```
## Getting input arguments :
 catch 3DPassport URL=<3DPassportURL>
 catch user login=toto
 catch user password=...
 catch 3DSpace URL=<3DSpaceURL>
 catch 3DCompass URL=<3DCompassURL>
 catch output directory =e:\temp\
 catch input scenario file =ScenarioMultiService.txt
## End Of arguments catching
```

It generates the web service execution in trace files:

- login.traces
- logout.traces
- api_ScenarioList.traces

And for on cloud the traces are:

```
## Getting input arguments :
 catch 3DPassport URL=<3DPassportURL>
 catch user login=toto
```

```
 catch user password=...
 catch 3DSpace URL=<3DSpaceURL>
 catch 3DCompass URL=<3DCompassURL>
 catch output directory =e:\temp\
 catch input scenario file =ScenarioMultiServiceCloud.txt
 catch platform instance =<tenantvalue>
## End Of arguments catching
```

Using service URLs with the right case is working fine, but using the tenant as query parameter is better.

It generates the web service execution in trace files:

- login.traces
- logout.traces
- api_ScenarioList.traces

If you want to use this batch for a service having a specific login API, use this API as input for the service. You can observe the login/logout trace. But you cannot execute web services (declared in scenario file) on this service. At least because the scenario file contains resource paths which are concatened to the input service URL. So if the latter already contains a resource path, it will fail.

In Short

This article explains how you can perform a **3D**EXPERIENCE service **login** based on the CAS authentication protocole. This security step is executed knowing the 3DPassport URL, and using a **java class following the redirections**. The latter point may be seen as convenient, but for the login step it may contain a drawback. When you execute the login against 3DPassport with a given **3D**EXPERIENCE web service URL, the latter returns 302 + Location header response containing the URL of the **3D**EXPERIENCE web service + CAS ticket. If you follow the redirection the web service is executed with the CAS ticket. It is allright if the verb of the web service is GET. For the other cases (PUT,POST,...) , it is an error if the process does not keep the initial VERB. The response is the one for the right URL but the wrong verb.

So this article explains that if your first request to the **3D**EXPERIENCE service is an API with GET as verb (named the "login" API) you can safely use a soft where HTTP request redirections are followed. The next article [5] explains how to proceed to follow yourself the redirections.

For the **logout,** use the 3DPassport logout API.

References

[1] CAS Protocol
[2] Cross-Site Request Forgery
[3] Getting Started
[4] Login API Issue
[5] Performing Service Login (other way)
[6] Presenting Batch Client Application Example

History

Version: **1** [May 2014] Document created
Version: **2** [Nov 2017] Document updated for some improvments and multi service login
Version: **3** [Nov 2023] Document updated to jakarta and code refactorisation
Version: **4** [Jan 2024] Document updated after using HTTP client java