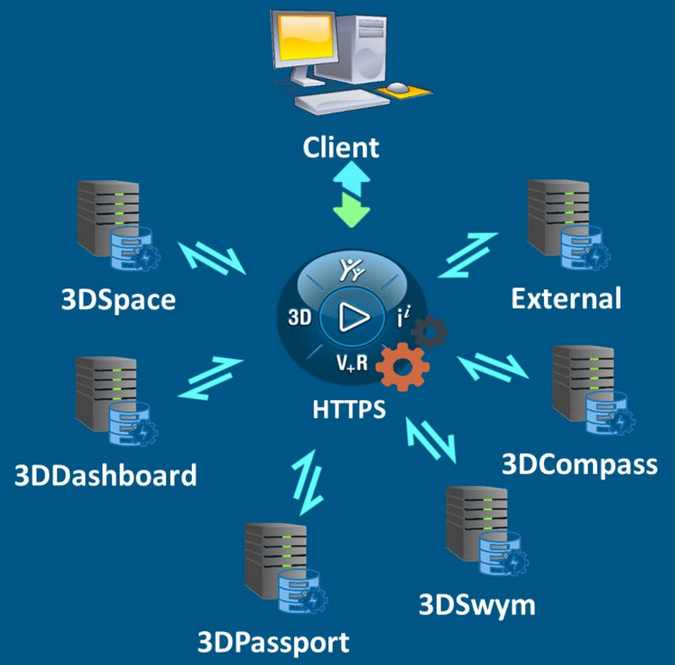
3DEXPERIENCE Web-Services

R2024x



About this course

Course objectives

Upon completion of this course you will be able to:

- Concepts about web-services and their different kinds
- Web-Services supported by platform
- How to develop web-services for 3DEXPERIENCE platform using Java
- Role of Annotations to create Java web-services
- Usage of OOTB web-services that are exposed by 3DEXPERIENCE platform
- Concepts of Enterprise Integration Framework.

Target audience

- Services Software Architect, Services Software Expert

Prerequisites

- The participant must have knowledge about:
  - Java or .NET/C#

11 Hours

Disclaimer

The content of this training material has been created with great care. Dassault Systèmes takes no responsibility for the topicality, correctness, completeness or quality of the data and information. Liability claims against Dassault Systèmes, which refer to material or immaterial nature caused by use or disuse of the data and information provided through the use of incorrect and incomplete data and information are basically excluded.

Dassault Systèmes reserves the right to change parts of this training material and information, or the entire offer without prior notice, add to, delete or cease publication temporarily or permanently.

All data and information are offered free and without obligation.

Caution / Warning

This document is based on a various internal DS content for **3D**EXPERIENCE platform R2024x.

In some cases, this document may contain forward looking statements based on current expectations, forecasts and assumptions that involve risks and uncertainties.

Upon further communication this document is classified to be DS Confidential. It should not be distributed without DS authorization.

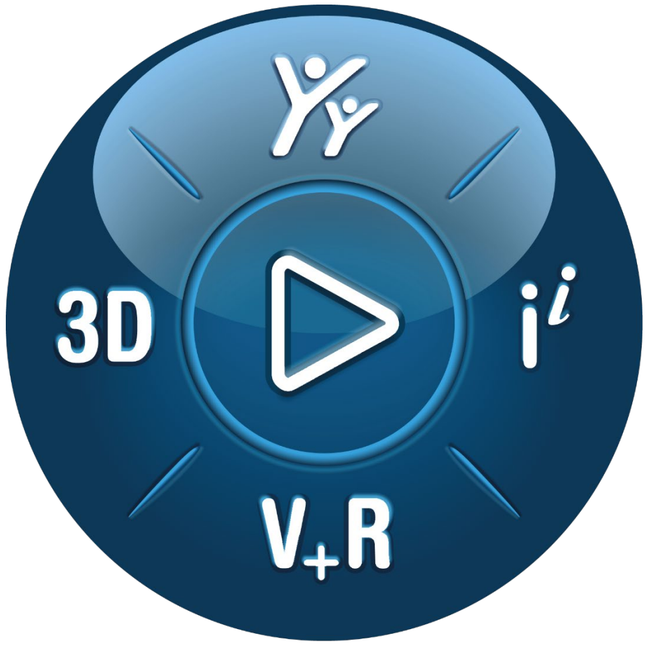
Changes in Course with Respect to R2023x

Some of the most important value additions made to the R2024x release of this course, with respect to the R2023x course, are as below:

- Enhanced the following topics
  - *Replaced JX-RS by Jakarta in eclipse code*
  - *Introduced bulkFetch*

TABLE OF CONTENTS

Introduction to 3DEXPERIENCE platform



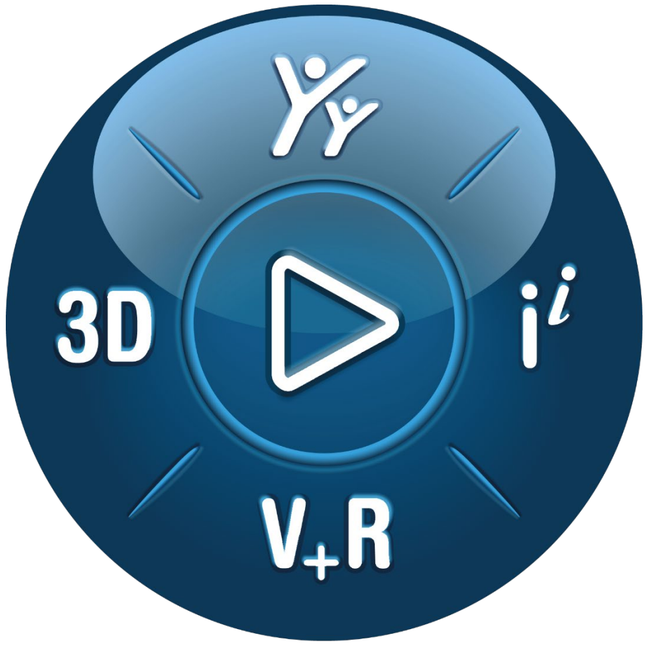
3DEXPERIENCE®

In this lesson we are going to be introduced to the basics of **3D**EXPERIENCE platform architecture. We shall:

- ❑ See pictorially how different services come together to form the **3D**EXPERIENCE platform
- ❑ Know about the n-tier architecture of the **3D**EXPERIENCE platform
- ❑ Know about how individual platform services interact with each other, and with external services

30 Minutes

3DEXPERIENCE platform Architecture



3DEXPERIENCE®

Here are the topics to be covered:

- ❑ 3DEXPERIENCE platform Architecture

3DEXPERIENCE Platform Apps and Services

3DDrive
Secure file storage on the cloud with 3DPassport.

3DComment
Share your thoughts & POV. Engage in discussion with peers.

3DNotification
Deliver notifications from platform as well as apps.

3DGlobe
interact with a terrestrial globe Manage Georeferenced objects

3DPassport
Identity management and authentication across all apps.

3DCompass
Personalized and role-based access to all apps.

3DSwym
Social collaboration through communities.

3DMessaging
Real time messaging, from chat to 3D snapshot.

3DSpace
Manage and share data, documents, related information.

Event Publishing

3DPlay
Visualize designs and review in real-time in browser

3DDashboard
Listen, track and govern flawlessly.

3DSearch & 6WTags
Intelligent search to re-use and re-purpose intellectual assets.

User Group:
Collaborate with users and groups of users

Collaborative Spaces Control | Communities Control Center | Enterprise Control Center | Members Control Center | Drives Control Center | 3DPassport Control Center

Enterprise & Control Centers
Manage members, role-based access and usage analytics.
Configure app behavior to suit your individual needs

NEW

Sandbox Control Center

eSignature Control Center

Extended Collaborations Control Center

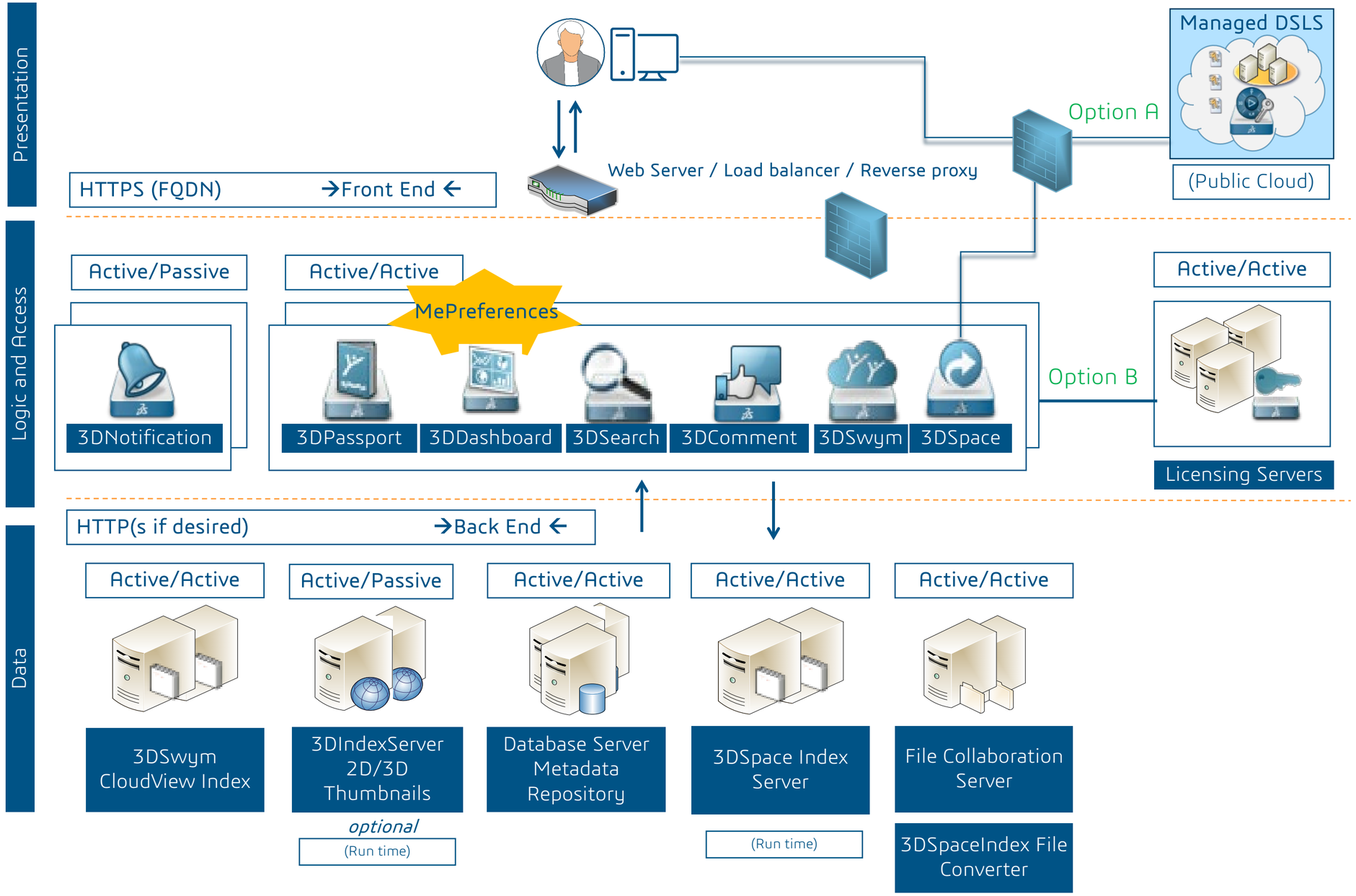
User Groups Control Center

3DEXPERIENCE Architecture

Active/Passive (Fail-over):
In the event of a Failure of an application server the session is dispatched on another application server.

Load balancing:
Ability to distribute load to different Application servers or File servers.

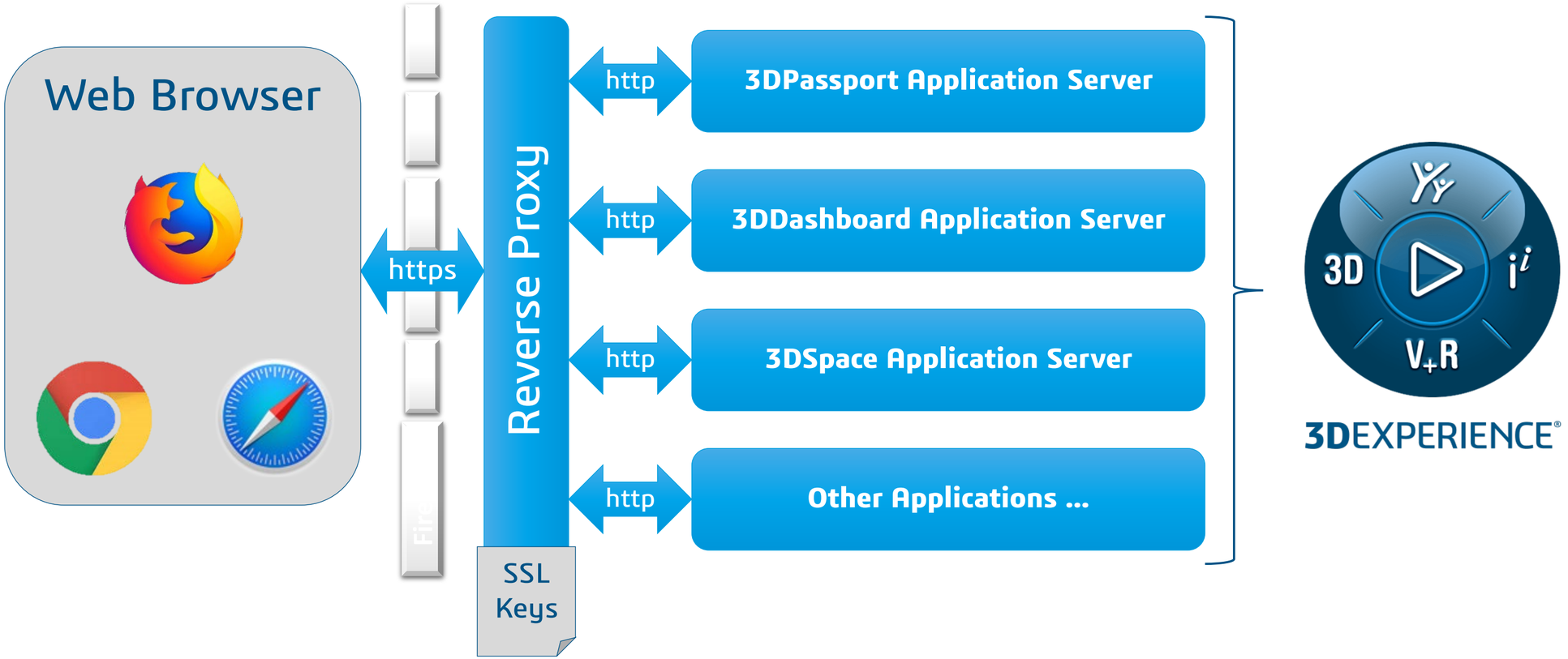
Modularity:
The degree to which a system's components may be separated and recombined



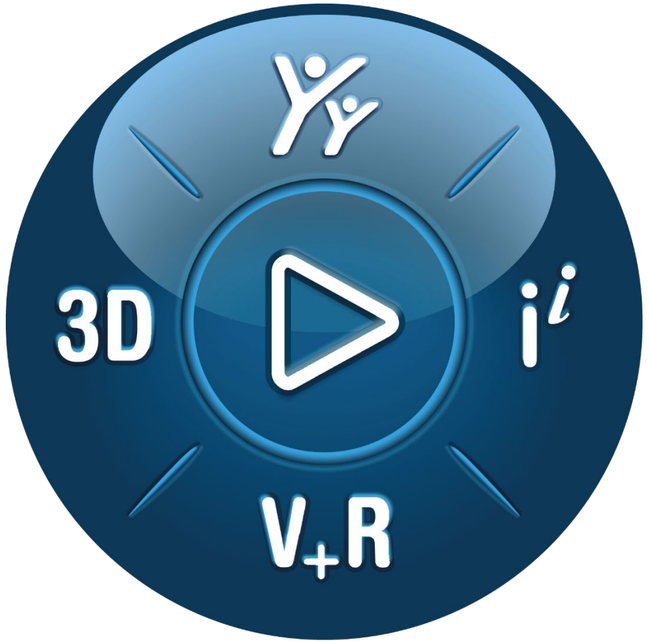
3DEXPERIENCE platform Application Interaction

- 3DEXPERIENCE platform services interact with each other, and also with external services, via **https** protocol for security reasons.

- In order for the application to be contacted by its clients, **Reverse Proxy** should be cascaded between the client and the application server for the service.



3DEXPERIENCE platform Web-Services



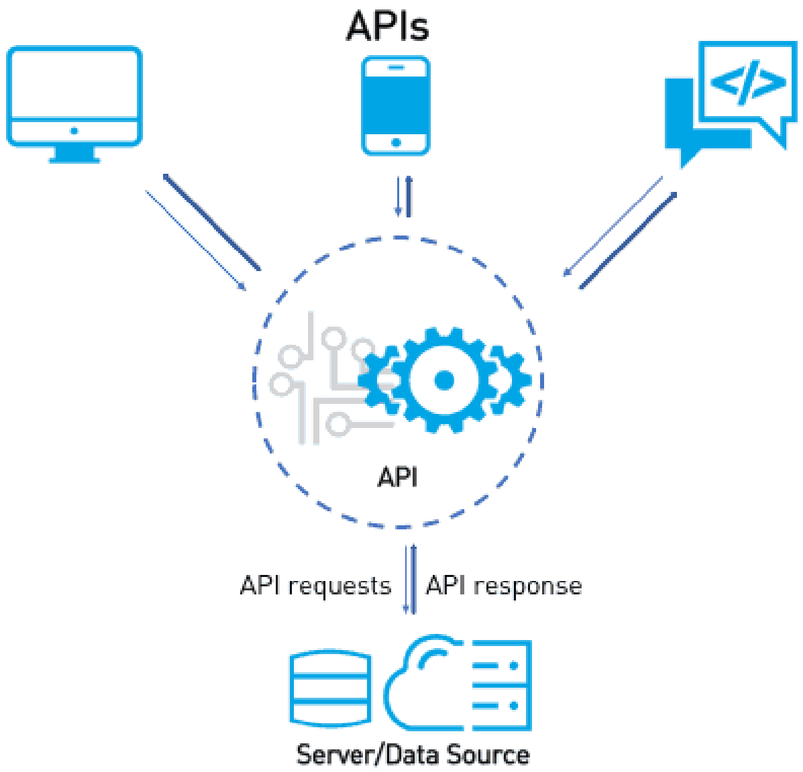
3DEXPERIENCE®

In this lesson we are going to know about the following topics:

- ❑ Basics of Web-Services
- ❑ Types of Web-Services
- ❑ How 3DEXPERIENCE platform uses and consumes web-services
- ❑ Various technologies used to build web-services
- ❑ Basics of Java web-services with respect to 3DEXPERIENCE platform

3.5 Hours

Introduction to Web-Services



Here are the topics to be covered:

- ❑ Introduction to Web-Services
- ❑ Types of Web-Services
- ❑ HTTP Methods for Web-Services
- ❑ Building a Java Web-Service for 3DEXPERIENCE platform

Web-Service – Introduction

- A Web-Service is any standard Application that is available over the network (Internet) which is capable of interacting with other Applications over the same network.

- In simple terms, a Web-Service is an application that responds to a call made by another application over the network.

- As a response, the Web-Service can:
  - Perform a task. For example, create some new data in the Database
  - Send back a response to the calling application, in a standard exchange format. The response format can be in XML, JSON, etc.

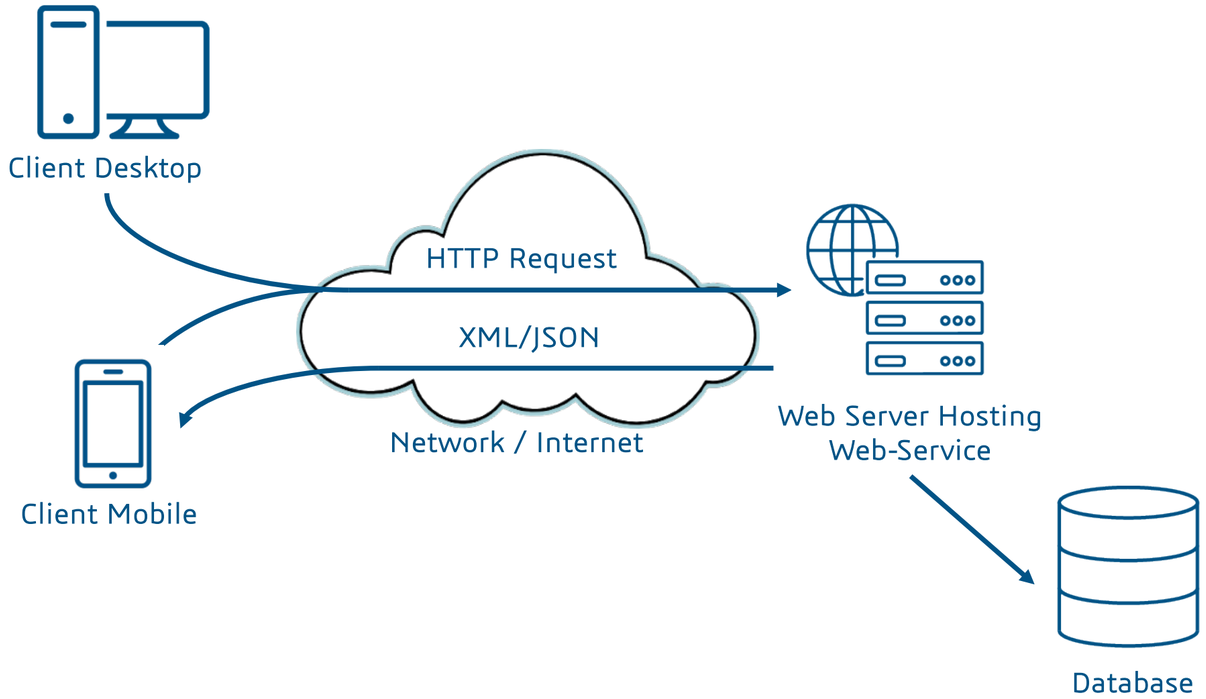
- Web-Services are typically designed to allow interoperability. This means that:
  - Web- Services are operating system independent.
    - The calling and responding applications may be hosted in environments with different operating systems.
  - Web-Services are programming language independent.
    - The calling application doesn't need to know about the programming language that was used to write the web-service. It should only be able to process the response.

- To summarize, a Web-Service is any application that:
  - Is available over the Internet or a private network
  - Uses a standard messaging system, like XML or JSON
  - Is not tied to any particular Operating System or programming language

Web-Service – Characteristics (1/3)

- In a standard Web-Service, the following hold true:
  - Standard protocols like HTTP are used for request and responses between Client machine and Server machine
  - Standard data exchange formats, like XML or JSON, are used to transfer data between Client machine and Server machine

- In practice, a Web-Service commonly provides an object-oriented Web-based interface to a database server.
  - The Web-Service may be utilized, for example, by another Web server, or by a mobile app, that provides a user interface to the end-user.



Web-Service – Characteristics (2/3)

- Web-Services are designed to be programming language independent:
  - The client application does not need to know about programming language that was used to develop the Web-Service

- For example, using Web-Services, a Java application can interact with other Java, .Net or PHP applications

Web-Service – Characteristics (3/3)

- The *State of a Web-Service* is one of the most commonly used terms. The State of a Web-Service can be confusing to understand:
  - A Web-Service, depending on its nature and architecture, can be either **Stateless** or **Stateful**.
  - A Stateful web-service is one that maintains some information between two consecutive requests from client to server.
    - *Example*: Let's say that an User enters username and password while invoking a web-service. A Stateful web-service may retain the username and password for the next call, so that the User does not have to enter it again.
  - A Stateless web-service is one that does not maintain any information between to consecutive requests from client to server.
    - *Example*: For calling a Stateless web-service, the username and password would need to be passed along with the request every time. The application would not store it anywhere.
  - An application built on Stateful web-services generally tends to be more wasteful of resources, as compared to a Stateless web-service.
    - This is because the application has to maintain the states of all open connections with its clients. More the number of clients that are connected to the application at the same time, more resources are spent to maintain the states of the open connections.
  - Application developers generally prefer to use Stateless web-services in their applications as they are lightweight and less taxing on the application servers.

Web-Service – Additional Information

- Web-Services are not to be confused with Java **Servlets**.

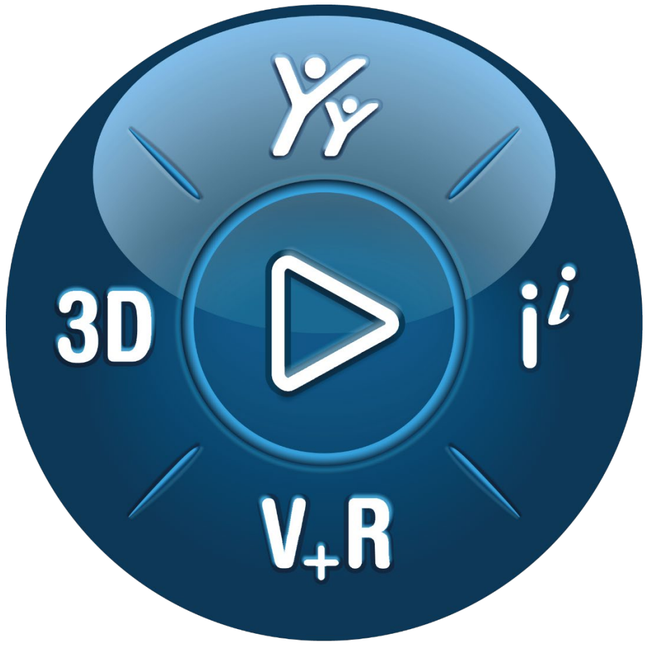
- A servlet is a Java programming language class that is used to extend the capabilities of servers and create web applications that responds to incoming requests.
  - Although servlets can respond to any type of request, they are commonly used to extend the applications hosted by web servers.

- The generic differences between Web-Services and Servlets are as follows:

Web-Services	Servlets
A Web-Service is a broader umbrella term, which may implement a servlet to deliver the service	A Servlet is a Java specific approach to that allows a Java application to respond to incoming requests
A Web-Service may use any open standard protocol, like HTTP or SOAP	A Servlet can operate only over HTTP protocol for exchanging data
Web-Services offer service discovery mechanisms like WSDL, WADL, etc. They provide a comprehensive list of available services, data exchanged in request and response, etc.	In case of a Servlet, the caller does not know what parameters are accepted by the Servlet application. The caller can only open an URL connection and pass parameters to the Servlet as part of the URL, without any knowledge of whether the parameters are actually being processed or if some parameters are missing.

In this course, we will not discuss about Servlets. Only Web-Services are in the scope of this learning module.

Types of Web-Services



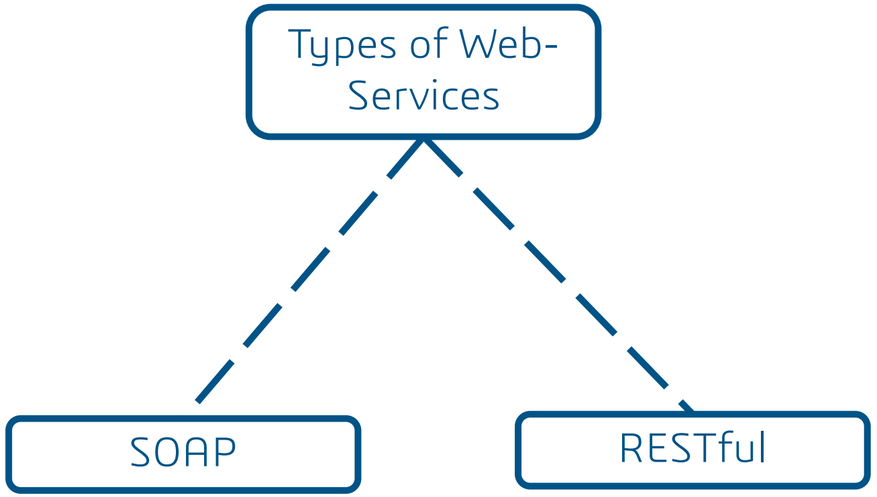
3DEXPERIENCE®

In this lesson, we shall know about the different kinds of web-services and their features. Here are the topics to be covered:

- *Introduction to Web-Services*
- Types of Web-Services
- HTTP Methods for Web-Services
- Building a Java Web-Service for 3DEXPERIENCE platform

Types of Web-Services

- Mainly, there are two types of Web-Services:
  - **SOAP** Web-Services
  - **REST**ful Web-Services



- In recent times, there has been much discussion about the **GraphQL** libraries which offers certain advantages over REST Web-Services.
  - However, GraphQL is still not as standardized as REST or SOAP Web-Services
  - We are not going to discuss about GraphQL in this course.

SOAP – Introduction

- **SOAP** stands for **S**imple **O**bject **A**ccess **P**rotocol.
  - **SOAP** is a W3C recommendation for communication between two applications.

- **SOAP** was developed as an intermediate language so that applications built on various programming languages could communicate easily with each other and avoid the extreme development effort.

- **SOAP** is an XML-based *protocol* for accessing Web-Services over HTTP.

  - Some legacy systems communicate over Simple Mail Transfer Protocol (SMTP) using SOAP.

  - HTTP has gained wider acceptance as it works well with today's internet infrastructure. Specifically, HTTP works well with network firewalls.

  - SOAP may also be used over HTTPS (which is the same protocol as HTTP at the application level, but uses an encrypted transport protocol underneath) with either simple or mutual authentication.

  - Use of XML as the exchange format allows the applications to send complex sets of data. Also, XML is a very standard data exchange format, which can be parsed by any programming language

- A **SOAP** web-service can be both Stateful and Stateless.

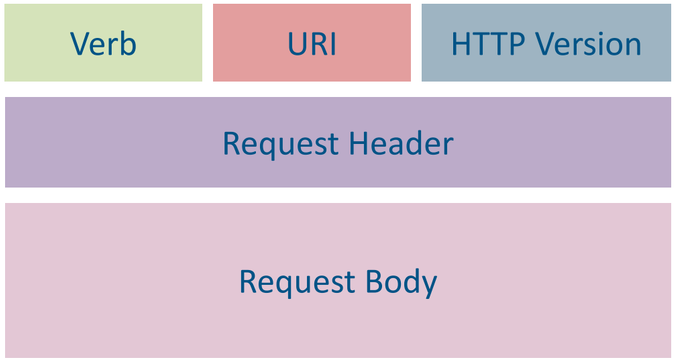
RESTful Web-Service – Introduction

- **RESTful Web-Services** is a lightweight, maintainable and scalable service that is built using the **REST** architecture.
- *REST Web-Services* allow the application to expose their APIs in a secure and **stateless** manner.
- *REST* stands for **RE**presentational **S**tate **T**ransfer.
- Some key characteristics of a RESTful service are as follows:
  - REST Web-Services operate over the HTTP protocol, using any of the normal HTTP verbs like GET, PUT, POST, DELETE, etc.
  - For a REST Web-Service, every component is a *resource*, that is accessed by a common Interface using HTTP standard methods.
  - REST Web-Services can work with all standard resources that are supported over HTTP, like text, JSON, XML, etc.
    - JSON is the most popular and widely using exchange format in case of REST Web-Services
  - Since a REST Web-Service is *stateless*, it is the entirely client's responsibility to maintain session information username and password (the application/server does not maintain it).
    - To overcome the shortcomings of a stateless web-service, the concept of a **cache** is often used to store information that is frequently requested for.
    - This also helps to reduce network traffic.
- Because of its lightweight, flexible and easy-to-develop nature, and also because of the advantages of being *Stateless*, REST web-services are gaining popularity over SOAP web-services.

REST Message (1/2)

- A REST web-service makes use of *HTTP* protocol for transferring message between Server and Client.

- Hence, a REST message is exactly the same as an HTTP message. An HTTP message has two categories:
  - **HTTP Request**: An HTTP request has five major components:
    - Verb – Indicates the HTTP methods such as GET, POST, DELETE, PUT, etc.
    - URI – Uniform Resource Identifier (URI) to identify the resource on the server.
    - HTTP Version – Indicates the HTTP version. For example, HTTP v1.1.
    - Request Header – Contains metadata for the HTTP Request message as key-value pairs. For example, client (or browser) type, format supported by the client, format of the message body, cache settings, etc.
    - Request Body -Message content or Resource representation.
      - In the adjacent example demonstrates a REST call, that transfers form data in the HTML body.



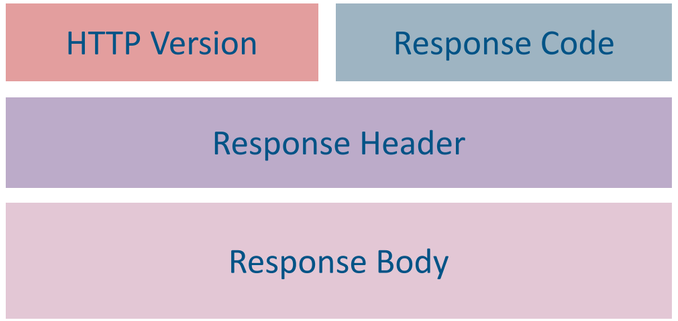

```
POST /cgi-bin/process.cgi HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; MSIE5.01; Windows NT)
Host: www.example.com
Content-Type: application/x-www-form-urlencoded
Content-Length: length
Accept-Language: en-us
Accept-Encoding: gzip, deflate
Connection: Keep-Alive

licenseID=string&content=string&paramsXML=string
```

REST Message (2/2)

❑ **HTTP Response**: An HTTP Response has four major components:

- **Response Code** – Indicates the Server status for the requested resource. For example, 404 means resource not found and 200 means response is ok.
- **HTTP Version** -Indicates the HTTP version. For example HTTP v1.1.
- **Response Header** – Contains metadata for the HTTP Response message as key-value pairs. For example, content length, content type, response date, server type, etc.
- **Response Body** – Response message content or Resource representation.



In the adjacent example, a successful web-service execution (indicated by return code **200**) has returned an HTML code in the response body.

```
HTTP/1.1 200 OK

Date: Mon, 24 May 2021 12:28:53 GMT
Server: Apache/2.2.14 (Win32)
Last-Modified: Wed, 22 Jul 2009 19:15:56 GMT
Content-Length: 88
Content-Type: text/html
Connection: Closed

<html>
    <body>
        <h1>Hello, World!</h1>
    </body>
</html>
```

SOAP vs REST

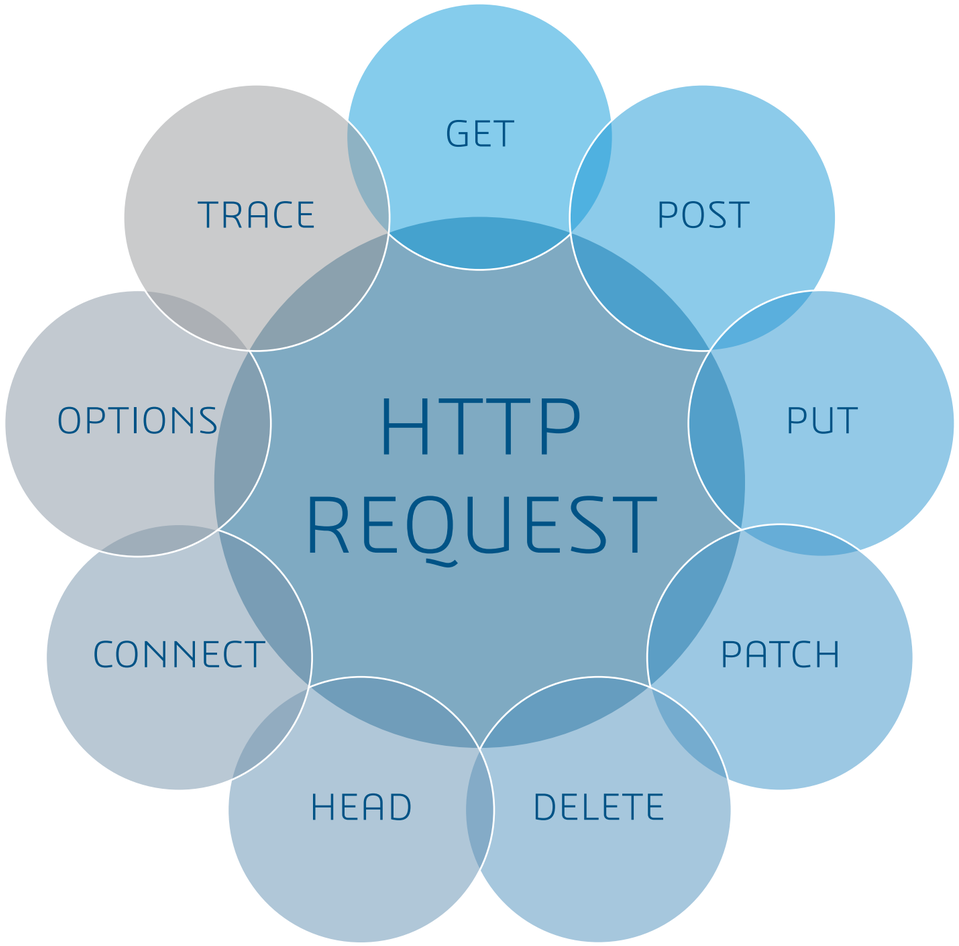
- Following are the differences between REST and SOAP web-services:

SOAP	REST
SOAP is a protocol.	REST is an architectural style.
SOAP cannot use REST, because SOAP is a protocol.	REST can use SOAP web-services because it is a concept and can use any protocol like HTTP, SOAP.
SOAP uses services interfaces (WSDL) to expose the business logic.	REST uses URI to expose business logic.
JAX-WS is the java API for SOAP web-services.	JAX-RS is the java API for RESTful web-services.
SOAP defines standards to be strictly followed.	REST does not define too much standards like SOAP.
SOAP requires more bandwidth and resources, as its message contains a lot of information in the form of tags.	REST requires less bandwidth and resources, as REST messages mostly consist of JSON data.
SOAP defines its own security.	RESTful web-services inherits security measures from the underlying transport layer.
SOAP permits XML data format only.	REST permits different data format such as Plain-text, HTML, XML, JSON etc.

Web-Service Discovery in 3DEXPERIENCE platform

- Although the 3DEXPERIENCE platform no longer supports SOAP web-services, WSDL or WIDL documents for pre-existing legacy services can be found at the following location in the 3DSpace deployment directory:
  - <TomEE Deployment Root>\WEB-INF\resources\widl
  - <TomEE Deployment Root>\WEB-INF\resources\wsdl
- It is very important to note that SOAP Web-Services are no longer supported in the 3DEXPERIENCE platform
  - New web-services developed using SOAP protocol will not be guaranteed by DS
- Only REST web-services are now supported in the 3DEXPERIENCE platform. REST web-services are discovered using their URI.
  - The URI of a Jakarta-RS web-service is configured by the use of **@Path** annotation. We shall discuss about these annotations in detail in a later chapter.

HTTP Methods for Web-Services

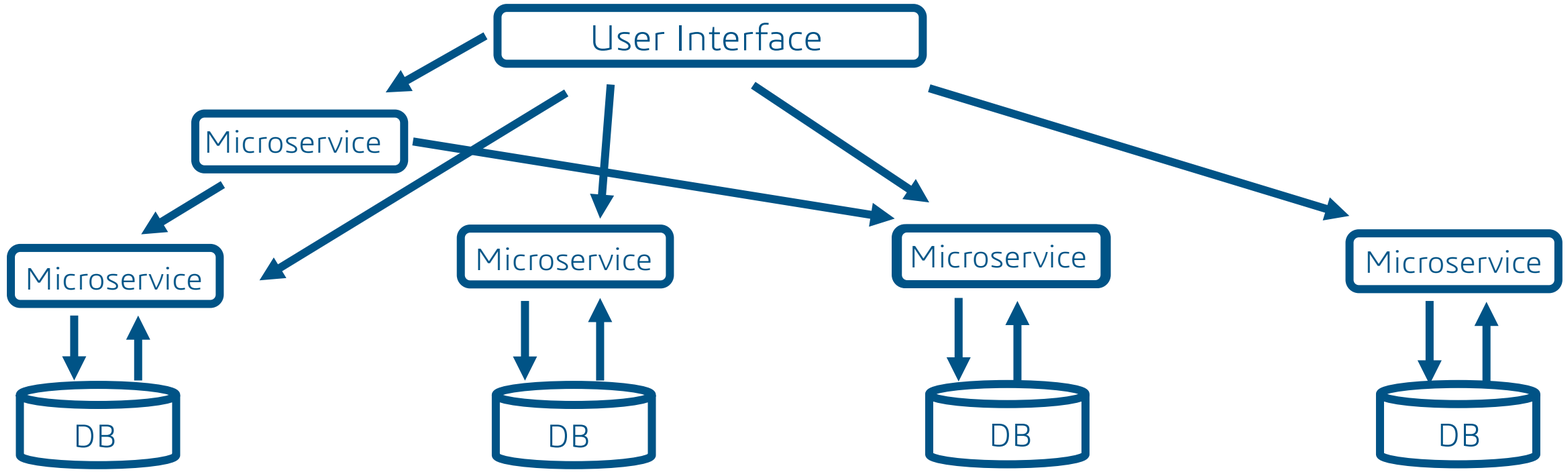


Here are the topics to be covered:

- ❑ *Introduction to Web-Services*
- ❑ *Types of Web-Services*
- ❑ HTTP Methods for Web-Services
- ❑ Building a Java Web-Service for **3D**EXPERIENCE platform

Microservice Architecture – Introduction

- Microservices is a service-oriented architecture pattern wherein applications are built as a collection of various smallest independent service units.
- It is a software engineering approach that focuses on decomposing an application into single-function modules with well-defined interfaces.
- Microservices can be independently deployed and operated by small teams that own the entire lifecycle of the service.
- All microservices should be loosely coupled with one another such that changes in one will not affect the other.
- Each service unit of the entire application should be the smallest and capable of delivering one specific business goal.
- Deployment of microservice application is very easy and less time-consuming.



HTTP Methods – Introduction

- Hyper Text Transfer Protocol (HTTP) works as a request-response protocol between a client and a server.
- In order to exchange data, HTTP makes use of Methods or Verbs. Some of the most widely used HTTP methods are listed below

Method Name	Description
GET	The GET method is used to retrieve information from the given server using a given URI.
POST	A POST request is used to send data to the server, for example, customer information, file upload, etc. using HTML forms.
HEAD	Same as GET, but transfers the status line and header section only. The response doesn't have a body.
PUT	PUT is used to send data to a server to create or update/replace a resource
PATCH	PATCH verb is used to update or modify an existing resource
DELETE	Removes all current representations of the target resource given by a URI
CONNECT	Establishes a tunnel to the server identified by a given URI
OPTIONS	Describes the communication options for the target resource
TRACE	Performs a message loop-back test along the path to the target resource

Although the PUT and PATCH verbs may look similar, they have their differences.

- the PUT method uses the request URI to supply a modified version of the requested resource which replaces the original version of the resource
- the PATCH method supplies a set of instructions to modify the existing resource. It does not replace the existing resource. The PATCH request only needs to contain the changes to the resource, not the complete resource.

HTTP Methods – Example

- **GET Method**: In a GET request, the data is primarily transferred as URL parameters (name/value pairs).
  - Example:

```
GET /test/demo_form.php?name1=value1&name2=value2 HTTP/1.1
```

- **POST Method**: The data sent to the server with POST is stored in the request body of the HTTP request.
  - Example:

```
POST /test/demo_form.php HTTP/1.1
Host: example.com

name1=value1&name2=value2
```

HTTP Methods – Path Vs Query Parameters

- **Path parameter** : It is a part of the URL and takes you to end-point/resources and give you the result of query from that resources.
  - It can not be null and if you don't append path parameter you will get 404 error.
  - A URL can include several parameters in the path.
  - Example : `GET /users/{id}`
    - In above example *id* is path parameter. Please note that curly braces around *id* are not part of the path parameter, it is denoted using curly braces.
    - So if we want to find user with id=1234 then our url will be `GET /users/1234`
- **Query parameter :** It is NOT a part of the URL and they are added to the url after the ? mark, as key and value it is filtering the result of query
  - It can be null and optional `GET /notes?offset=100&limit=50`
  - Different query parameter name=value pairs are separated by ampersands (&)
  - Example :
    - In above example *offset =100* and *limit =50* are query parameters in the form of key value pair

CRUD

- CRUD is an acronym that comes from the world of computer programming and refers to the four functions that are considered necessary to implement a persistent storage application: **C**reate, **R**ead, **U**pdate and **D**elete.

- In a REST environment, CRUD often corresponds to the HTTP methods POST, GET, PUT, and DELETE, respectively. These are the fundamental elements of a persistent storage system.

  - Create - To create resources in a REST environment, we most commonly use the HTTP POST method. POST creates a new resource of the specified resource type.

  - Read - To read resources in a REST environment, we use the GET method

  - Update - To update resources in a REST environment, we use the PUT method

  - Delete - To delete resources in a REST environment, we use the DELETE method

HTTP verb	CRUD
POST	Create
GET	Read
PUT	Update
PATCH	Update
DELETE	Delete

HTTP Methods – GET vs POST in context of Web Browser

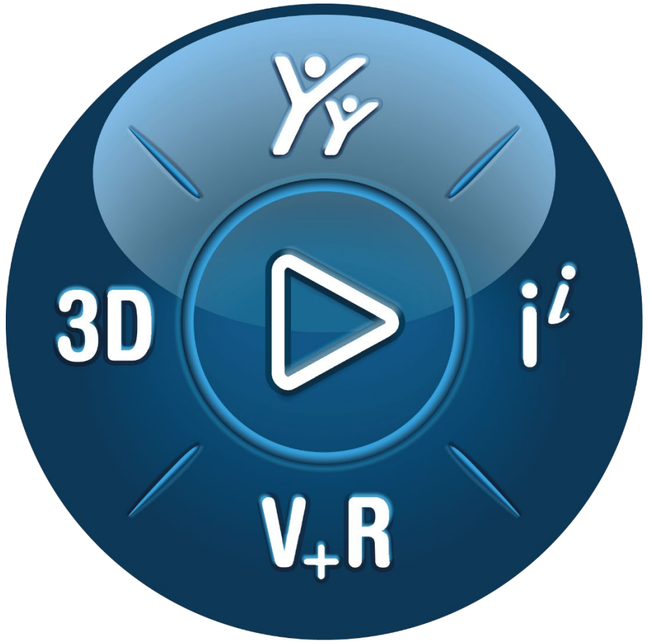
- The following table compares the two HTTP methods: **GET** and **POST** in context of Web Browser

GET	POST
In GET method, values are visible in URL	In Post method, values are NOT visible in URL
GET has a limitation on the length of the values, generally 255 characters	POST has no limitation on the length of the values, since they are submitted via the body of HTTP.
GET performs are better compared to POST because of the simple nature of appending the values in the URL.	It has lower performance as compared to GET method because of time spent in including POST values in the HTTP body.
This method supports only string data types.	This method supports different data types, such as string, numeric, binary, etc.
GET request is often cacheable	POST request is hardly cacheable

Building a Java Web-Service for 3DEXPERIENCE platform

In this lesson, we shall explore the following:

- ❑ Java Libraries required to create a standard **3D**EXPERIENCE platform web-service
- ❑ Sample Java web-service code



3DEXPERIENCE®

Here are the topics to be covered:

- ❑ *Introduction to Web-Services*
- ❑ *Types of Web-Services*
- ❑ *HTTP Methods for Web-Services*
- ❑ Building a Java Web-Service for 3DEXPERIENCE platform

3DEXPERIENCE platform Web-Service Development

❑ The following table compares the various available Web-Services technologies, with respect to the 3DEXPERIENCE platform.

			Strategic Direction	
	WIDL	WSDL/JPO	JAX-RS	JAX-WS
Type of service	Doc-based	SOAP/Doc-based	REST	SOAP/Doc-based
Statefulness	Both	Stateful	Stateless	Both
Authentication	Proprietary	No	Yes	Yes
Standard based	Proprietary	AXIS 1.4 *Obsolete since about 2008	Java EE 6 Web Profile	Java EE 6 Web Profile
Customizable	R&D Only **Requires ADELE compiler	No	Yes	No
Object visibility	IRPC only	ER only	ER only (R2015x/R2016x) IRPC/ER (>R2016x)	ER only
Hot Deploy	No	Yes	No	Not without heavy customization
Availability	Being phased out	Being phased out	Limited public endpoints	No plan

Jakarta-RS API – Introduction

- **Jakarta-RS (JAX-RS in the previous versions)** is a JAVA based programming language API and specification to provide support for creating RESTful Web-Services.

- **Jakarta-RS** uses annotations available from Java SE 5 to simplify the development of web-services and its subsequent deployment.

- From version 1.7 on, **Jakarta-RS** is an official part of Java EE 10. A notable feature of being an official part of Java EE is that no configuration is necessary to start using **Jakarta-RS**.

  - For environments with releases earlier than Java EE 6, a small entry in *web.xml* deployment descriptor file is required to deploy the Web-Service

- **Jakarta-RS** also provides support for creating clients for RESTful Web-Services.

JAX-RS API – Annotations

- Following are the most popular and common annotations that are used in a JAX-RS implementation

Annotation	Description
@ApplicationPath	The @ApplicationPath annotation is used to define the URL mapping for the application.
@Path	Identifies the *Relative* path of resource class/method, relative to @ApplicationPath
@PathParam	Binds the parameters passed to a method to values in the Path.
@QueryParam	Binds the parameters passed to a method to values in the request URL parameter
@Consumes	Specifies the MIME types of the HTTP request that the resource can consume
@Produces	Specifies the MIME types of the HTTP response that the web-service returns back to the client
@FormParam	Binds the parameter passed to the method to a form value
@GET	Designates a method for the HTTP GET request
@POST	Designates a method for the HTTP POST request
@PUT	Designates a method for the HTTP PUT request
@DELETE	Designates a method for the HTTP DELETE request
@HEAD	Designates a method for the HTTP HEAD request
@OPTIONS	Designates a method for the HTTP OPTIONS request
@HeaderParam	Represents the parameters of the header
@CookieParam	Represents the parameters of the cookie
@Context	Used to inject information, like Security Context, into an instance field or directly into the resource method as a parameter

Implementing REST Web-Service in 3DSpace (1/2)

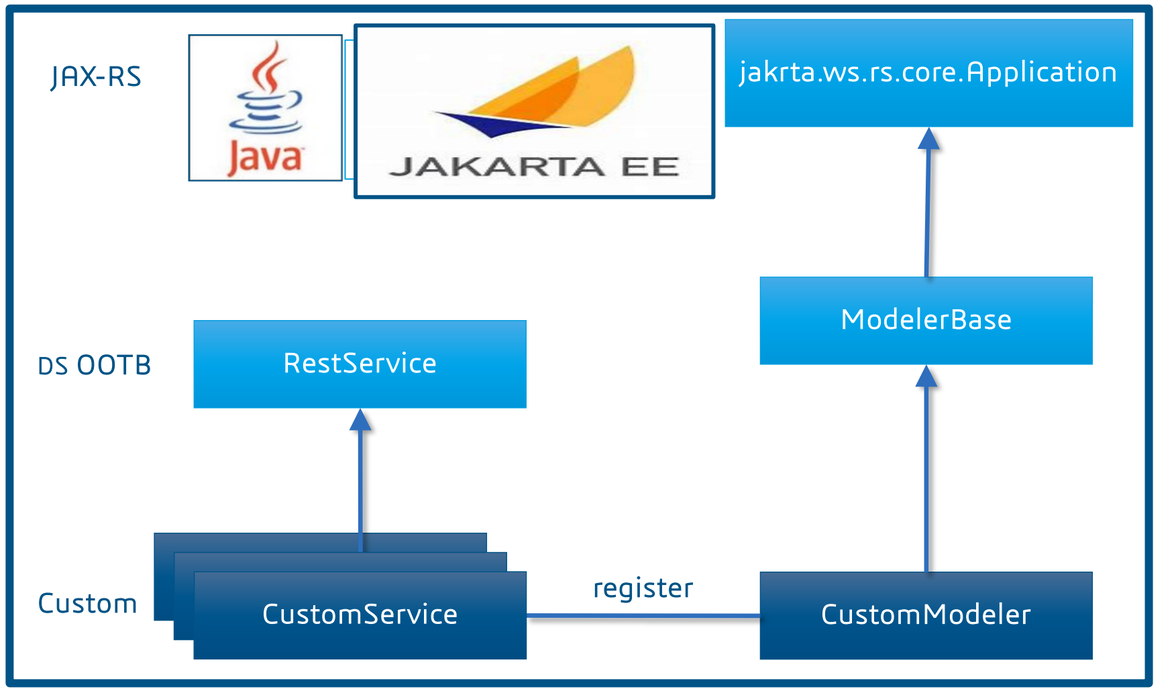
- The architecture of a **3D**EXPERIENCE platform web-service can be explained with the following diagram.

- The main Application class (*CustoModeler*), which would act as the entry point into the application, needs to conform to the following rules:
  - It applies the **@ApplicationPath** annotation
  - It extends the following class: com.dassault_systemes.platform.restServices.ModelerBase

    The **ModelerBase** class extends the javax.ws.rs.core.Application which is needed for the @ApplicationPath annotation.

    The **ModelerBase** class contains the following important methods:

    - **init()**: Used by the servlet container to initialize the internal servlet implementation.
    - **getServices()**: This is an abstract method which is overridden in the main Application class to return the class(es) implementing the resources



Implementing REST Web-Service in 3DSpace (2/2)

- By overriding the **getServices()** method, the main Application class returns the service class(es) to the servlet container.
  - In the example in the previous page, the Service class is named as *MyService*.
- Below are some important points that are to be noted regarding the service class(es):
  - The service class applies the **@Path** annotation either at the class level or the method level
  - The service class extends the com.dassault_systemes.platform.restServices.RestService
    - The class **RestService** implements the method **getAuthenticatedContext()** which is required to obtain a valid Matrix context that is needed for transactions.
  - The service class applies annotations like @GET, @POST, etc., as required, against corresponding methods to designate resources for the HTTP request verbs that the service is supposed to respond to.

REST Web-Service in 3DSpace – Example (1/8)

- We shall try to understand the implementation of REST web-service in 3DSpace with the example of a simple **Calculator** application.

- The **Calculator** application is implemented using a web-service, and it has the following modules:
  - The **Add** module, which adds two numbers and returns the result
  - The **Subtract** module, which finds the difference between two numbers
  - The **Multiply** module, which multiplies two numbers and returns the result
  - The **Divide** module, which Divides one number by another, and returns the result

- The **Calculator** web-service would accept input and return the corresponding results in **JSON** format.

- In the following pages, we shall see the code that implements the above web-service.

REST Web-Service in 3DSpace – Example (2/8)

- The main Application class is as follows:

```
package com.webservices;
import com.dassault_systemes.platform.restServices.ModelerBase;

import jakarta.ws.rs.ApplicationPath;
import jakarta.ws.rs.Path;

@ApplicationPath("/DSISCalculator")
public class DSISCalculator extends ModelerBase {
public Class<?> [] getServices () {

return new Class[] {DSISAdd.class, DSISSubtract.class, DSISMultiply.class, DSISDivide.class};
}
}
```

- Observe the following about this application class **DSISCalculator**:
  - It uses the **@ApplicationPath** annotation
  - It extends the **ModelerBase** parent class
  - It overrides the **getServices()** method to return the service classes that implement the web-service resources

REST Web-Service in 3DSpace – Example (3/8)

- The services class **DSISAdd** is as follows:

```
package com.webservices;
import com.dassault_systemes.platform.restServices. RestService;

import jakarta.json.Json;
import jakarta.json.JsonObjectBuilder;
import jakarta.ws.rs.GET;
import jakarta.ws.rs.Path;
import jakarta.ws.rs.Path;
import jakarta.ws.rs.Produces;
import jakarta.ws.rs.QueryParam;
import jakarta.ws.rs.core.Response;
@Path("/add")
@Produces ({"application/json"})
public class DSISAdd extends RestService {
@GET
public Response addFunction (@jakarta.ws.rs.core.Context HttpServletRequest request, @QueryParam ("num1") String num1, @QueryParam ("num2") String num2) { Strir
float val1,val2,result;
JsonObjectBuilder output = Json.createObjectBuilder();
try {
val1=Float.parseFloat(num1);
val2 = Float.parseFloat (num2);

result=val1 + val2;
output.add("msg", "ok");
output.add("result", result);
} catch (Exception e) {

System.out.println("Error: "+e);
return Response.status (401) .entity("Error") .build();
}
String outputArray = output.build().toString();
return Response.status (200) .entity (outputArray) .build();

}
}
```

- Observe the following:
  - The class uses the **@Path** annotation to define the service path, and extends the **RestService** parent class
  - The class defines methods to respond to the GET HTTP request, using **@GET** annotation.

REST Web-Service in 3DSpace – Example (4/8)

- Similarly, we have the service class **DSISSubtract**

```
package com.webservices;
import com.dassault_systemes.platform.restServices. RestService;

import jakarta.json.Json;
import jakarta.json.JsonObjectBuilder;
import jakarta.ws.rs.GET;
import jakarta.ws.rs.Path;
import jakarta.ws.rs.Path;
import jakarta.ws.rs.Produces;
import jakarta.ws.rs.QueryParam;
import jakarta.ws.rs.core.Response;
@Path("/subtract")
@Produces ({"application/json"})
public class DSISSubtract extends RestService {
@GET
public Response addFunction (@jakarta.ws.rs.core.Context HttpServletRequest request, @QueryParam ("num1") String num1, @QueryParam ("num2") String num2) { Strir
float val1,val2,result;

JsonObjectBuilder output = Json.createObjectBuilder();
try {
val1=Float.parseFloat(num1);
val2 = Float.parseFloat (num2);

result=val1 - val2;
output.add("msg", "ok");
output.add("result", result);
} catch (Exception e) {

System.out.println("Error: "+e);
return Response.status (401) .entity("Error") .build();
}
String outputArray = output.build().toString();
return Response.status (200) .entity (outputArray) .build();

}
}
```

REST Web-Service in 3DSpace – Example (5/8)

- Similarly, we have the service class **DSISMultiply**

```
package com.webservices;
import com.dassault_systemes.platform.restServices. RestService;

import jakarta.json.Json;
import jakarta.json.JsonObjectBuilder;
import jakarta.ws.rs.GET;
import jakarta.ws.rs.Path;
import jakarta.ws.rs.Path;
import jakarta.ws.rs.Produces;
import jakarta.ws.rs.QueryParam;
import jakarta.ws.rs.core.Response;
@Path("/multiply")
@Produces ({"application/json"})
public class DSISMultiply extends RestService {
@GET
public Response addFunction (@jakarta.ws.rs.core.Context HttpServletRequest request, @QueryParam ("num1") String num1, @QueryParam ("num2") String num2) { Stri
float val1,val2,result;

JsonObjectBuilder output = Json.createObjectBuilder();
try {
val1=Float.parseFloat(num1);
val2 = Float.parseFloat (num2);

result=val1 * val2;
output.add("msg", "ok");
output.add("result", result);
} catch (Exception e) {

System.out.println("Error: "+e);
return Response.status (401) .entity("Error") .build();
}
String outputArray = output.build().toString();
return Response.status (200) .entity (outputArray) .build();


}
}
```

REST Web-Service in 3DSpace – Example (6/8)

- Similarly, we have the service class **DSISDivide**

```
package com.webservices;
import com.dassault_systemes.platform.restServices.RestService;
import jakarta.json.Json;
import jakarta.json.JsonObjectBuilder;
import jakarta.ws.rs.GET;
import jakarta.ws.rs.Path;
import jakarta.ws.rs.Path;
import jakarta.ws.rs.Produces;
import jakarta.ws.rs.QueryParam;
import jakarta.ws.rs.core.Response;
@Path("/divide")
@Produces ({"application/json"})
public class DSISDivide extends RestService {
@GET
public Response divideFunction (@jakarta.ws.rs.core.Context HttpServletRequest request, @QueryParam ("num1") String num1, @Quer
float val1, val2;
String result = new String();
JsonObjectBuilder output = Json.createObjectBuilder();
try {
          if((val1>0) && (val2==0)) {
              result = Float.toString(Float.POSITIVE_INFINITY);
          } else if ((val1<0) && (val2==0)) {
              result = Float.toString(Float.NEGATIVE_INFINITY);
          } else if ((val1==0)&&(val2==0)) {
              result = "UNDEFINED";
          } else {
              result = Float.toString(val1/val2);
          }
          output.add("msg", "ok");
          output.add("result", result);
       } catch (Exception e) {
          System.out.println("Error: "+e);
          return Response.status(401).entity("Error").build();
       }
       String outputArray = output.build().toString();
       return Response.status(200).entity(outputArray).build();
   }
}
```

REST Web-Service in 3DSpace – Example (7/8)

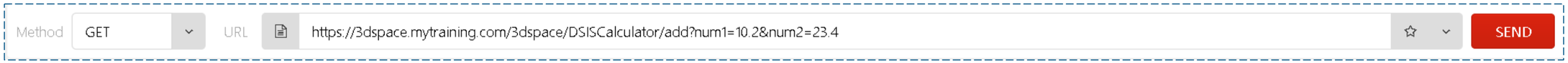
- Let us assume the following about the deployment of the developed web-service:
  - The Web-Service is deployed to 3DSpace
  - The root URL of the 3DSpace application is https://3dspace.mytraining.com/3dspace

- We wish to add two numbers **10.2** and **23.4**

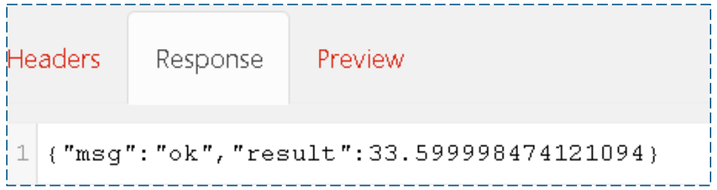
- After studying the code in the previous pages, we know that the value of **@ApplicationPath** for main application class is "**/DSISCalculator**". Also, the value of **@Path** for the service class that performs the additions is "**/add**".

- Hence, we can add the above two numbers by calling the following URL with the **GET** HTTP verb:

  https://3dspace.mytraining.com/3dspace/DSISCalculator/add?num1=10.2&num2=23.4



- The response is a JSON object, which is as follows:



For this example, the **RESTClient** plugin for Firefox browser was used to test the Web-Service calls. There are other tools as well, which we shall discuss in the following pages.

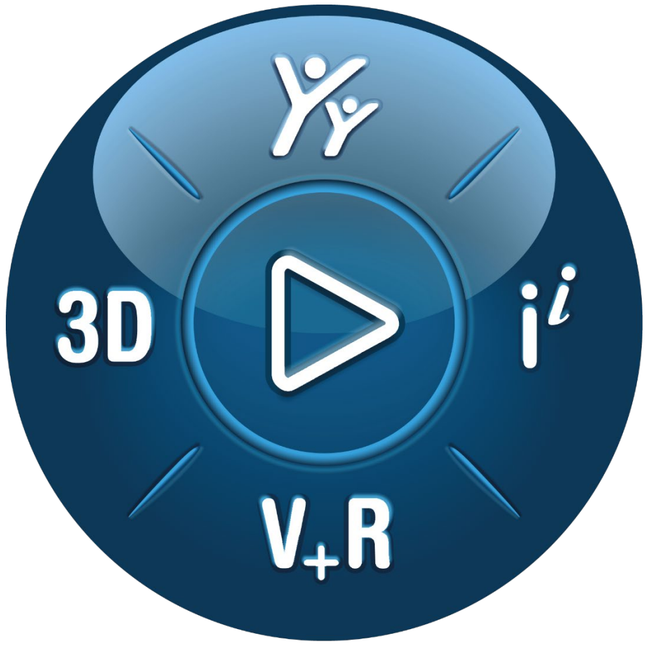
REST Web-Service in 3DSpace – Example (8/8)

- **Important Note**: It is not mandatory for the service classes to extend com.dassault_systemes.platform.restServices.RestService.
  - The **RestService** class implements the **getAuthenticatedContext()** which is required to obtain a valid Matrix context that is needed for transactions.
  - Unless we need to make a transaction with the **3D**EXPERIENCE platform database, call an SCT class or call a JPO, where a valid session context needs to be passed, it is not required to extend the **RestService** class from the service class.
- For example, we can have the following class definition for the **DSISSubtract** class:

```
@Path("/subtract")
@Produces({"application/json"})
public class DSISSubtract {
    @GET
    @POST
    public Response subtractFunction(@QueryParam("num1") String num1, @QueryParam("num2") String num2) {
```

- The above definition would still work as we are not making any transaction or calling a JPO, and hence we do not need a security context for the session.

Consuming 3DEXPERIENCE platform Web-Services



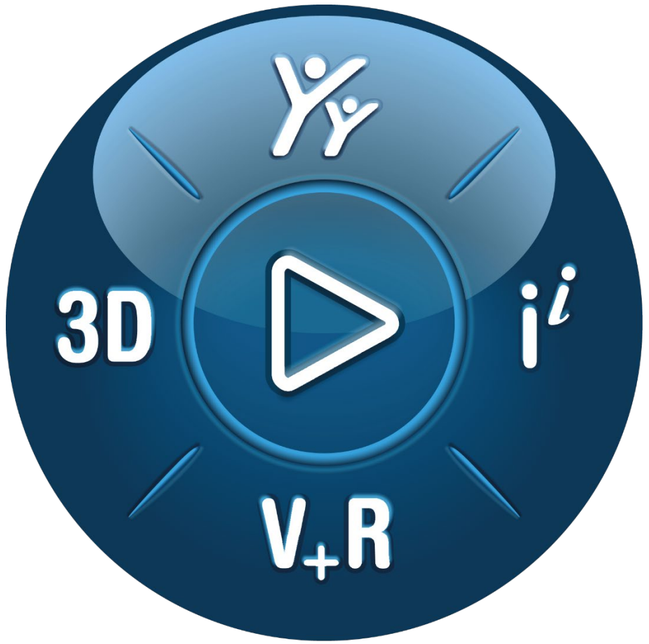
3DEXPERIENCE®

In this lesson we are going to know about the following topics:

- ❑ Various way in which 3DEXPERIENCE platform web-services are consumed
- ❑ Documentation about 3DEXPERIENCE platform web-services
- ❑ Tools to test Web-Services
- ❑ Demonstration of how to read Developer Assistance documentation to test a 3DEXPERIENCE platform web-service, using a popular testing tool

3 Hours

Consuming Web-Services in **3D**EXPERIENCE platform



3DEXPERIENCE®

In this chapter, we shall be briefly introduced to how Web-Services are consumed in **3D**EXPERIENCE platform.

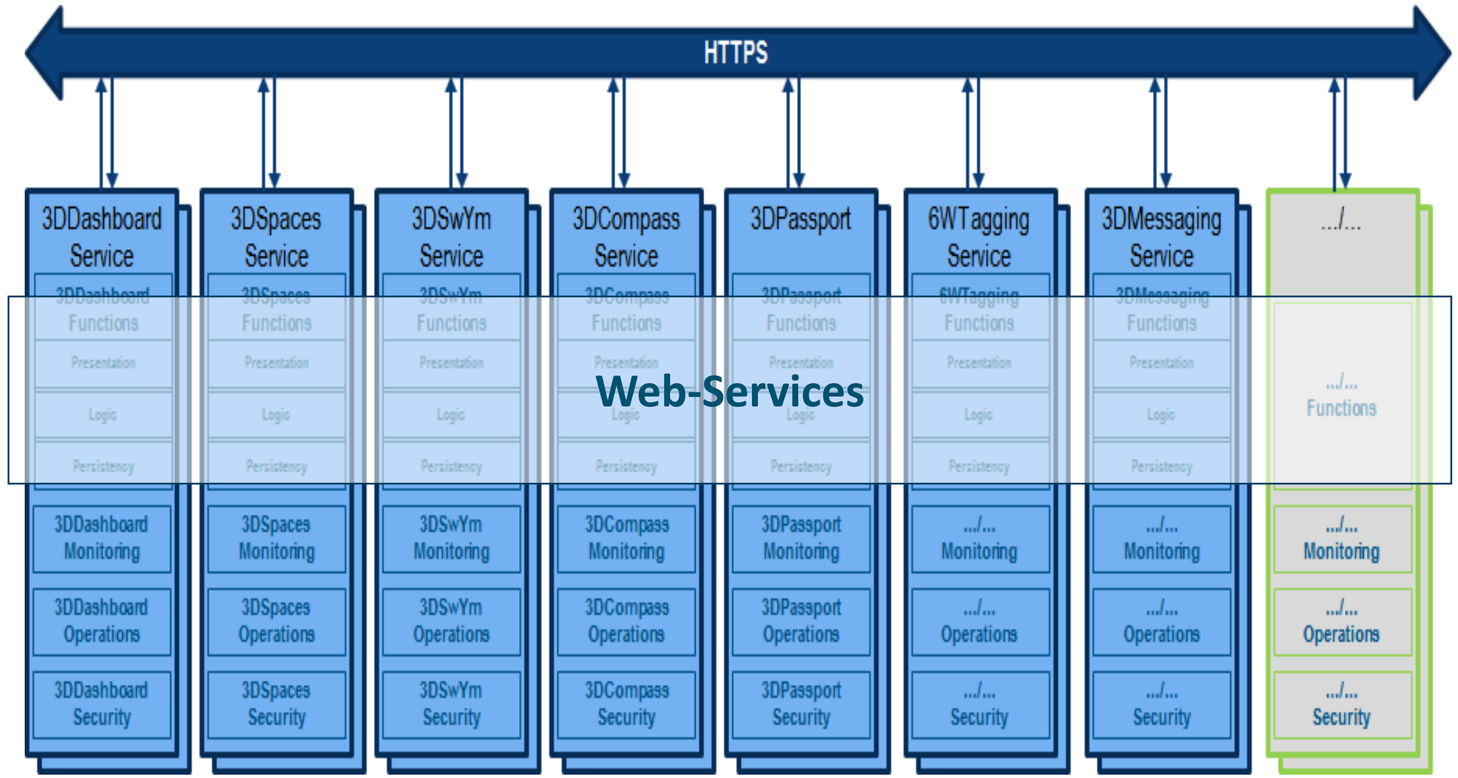
Here are the topics to be covered:

- Tools for Testing Web-Services
- Consuming Web-Services in 3DEXPERIENCE platform
- Documentation for **3D**EXPERIENCE platform Web-Services
- Demonstration with Engineering Web-Services

Consuming Web-Services in **3D**EXPERIENCE platform (1/4)

- As already described earlier, the **3D**EXPERIENCE platform consists of many different service, like 3DPassport, 3DSpace, 3DSpaceIndex, etc.
  - External, customer specific services can also be configured with the **3D**EXPERIENCE platform.

- All these services interact with each other using Web-Services.



Consuming Web-Services in **3D**EXPERIENCE platform (2/4)

- There are four contexts to consume **3D**EXPERIENCE platform web-services:
  - From a dashboard application (custom widget), one can reach **3D**EXPERIENCE services. The **3D**EXPERIENCE authentication (username/password) has to be performed manually by the concerned person.
  - From any non-DS applications (web or batch) you can reach **3D**EXPERIENCE services using your personal credential. Before consuming any DS web-services, the **3D**EXPERIENCE authentication must be executed.
  - From any non DS applications (web or batch) you can reach **3D**EXPERIENCE services on behalf of someone. The capability is possible through *service agent*, and is available for on-premises or private cloud only.
  - From iPaaS (Integration Platform As A Service) you can consume **3D**EXPERIENCE web-services on behalf of someone using an *openness agent*. This capability is available for cloud only.

Consuming Web-Services in **3D**EXPERIENCE platform (3/4)

On-Behalf Authentication for 3DEXPERIENCE platform:

- **3D**EXPERIENCE platform authentication is managed by 3DPassport. It consists of checking the *password* for the user who is trying to access the platform. For automation processes, it is recommended to use **On-Behalf** authentication, where the password of the platform user is not distributed.

- On-Behalf Authentication can be achieved with the following:
  - **Service Agent**:
    - Available for On-Premises and Private Cloud
    - Using the **3DPassport Control Center** app platform, administrators can declare a *Service Agent*
    - A batch can perform an "on-behalf" CAS authentication by passing *Service Agent* credentials (batch-service name and passkey) and the username of a Person. It leads to authenticate the person to the 3DEXPERIENCE while never providing his/her password.
    - For security reason, only platform member people may be "on-behalf" person
    - More about Service Agents can be found in Developer Assistance at this link
  - **Openness Agent**:
    - Available for Public Cloud only
    - The Administrator of a Tenant, with Enterprise Integration Architect (PFI) role, can use the **Agent Management** app to work with Openness Agents
    - The Openness Agent is an object managed by the 3DEXPERIENCE service named Credential Lifecycle Management (CLM)
    - To create a new Openness Agent, an Agent name, and the username of a **3D**EXPERIENCE platform user are required. Using these, the CLM returns an internal name (agent ID) and a password (agent password).
    - Using the credentials as generated by the CLM, a machine/batch can listen to messages from the **3D**EXPERIENCE platform, or call **3D**EXPERIENCE platform web-services
    - More about Openness Agent can be found in Developer Assistance at this link.

Consuming Web-Services in 3DEXPERIENCE platform (4/4)

Centralized Authentication Service (CAS) for 3DEXPERIENCE platform:

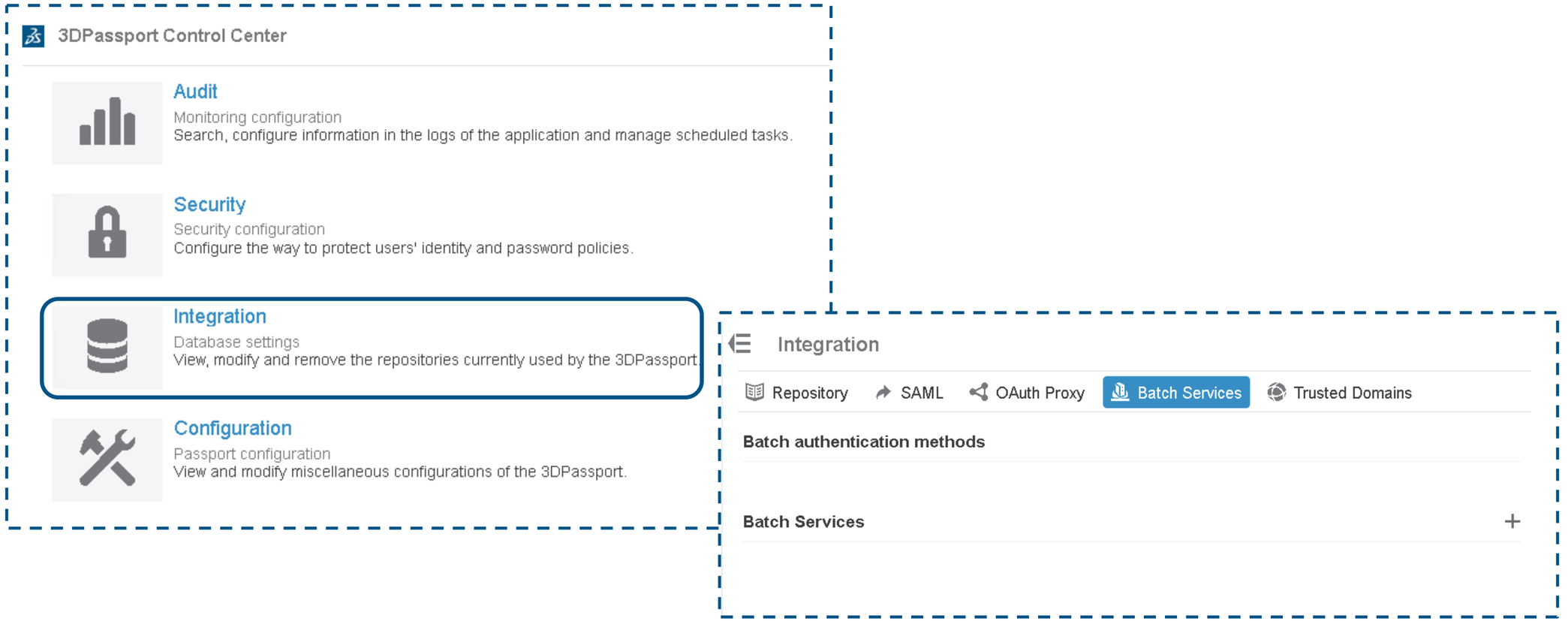
- The 3DEXPERIENCE platform relies its authentication on a centralized authentication service system, shortly named CAS.
- The 3DPassport service is the service managing and controlling by this way the 3DEXPERIENCE authentication.
- The CAS protocol ensures that once you are authenticated no further authentication will be requested for accessing any authenticated services of the platform. It is the Single Sign On (SSO) principle.
- In a same way, when there is a logout from any service, it automatically implies a logout of all other services. It is the Single Logout Out (SLO) principle.
- The authentication consists in checking the credentials of the person trying to access the 3DEXPERIENCE. This person must already have a 3DPassport account.
- Such account is created by the person after having being invited to the 3DEXPERIENCE by an administrator. The person is identified by a username (also named login name).
- For the authentication, either the username or the email address (the one used for the invitation) may be used. At the first connection, 3DPassport confirms the password. The username and password are the person's credentials.
- Once the authentication is validated by 3DPassport , each first access to a 3DEXPERIENCE service triggers a backend conversation between the specific service and the 3DPassport to check that successful authentication has previously happened. Cookies are exchanged between the service and the client to cache authentication status. Cookies expire after a limited amount of time and re-authentication is required.

Transient Ticket– Demonstration – Introduction

- In this demonstration, we shall generate the transient ticket and perform CAS Authentication using web service :
  - We shall use **Postman** as the tool to call the web-services, and analyze their output
  - We shall refer to Developer Assistance to get information about the web-services that would be run, like, required parameters, expected output, etc.
- Following are the **prerequisites** for this demonstration:
  - If an On-Premises installation is used, then ensure the following:
    - **3D**EXPERIENCE platform is properly setup and configured
    - All services are up and running
    - 3DSpace Index service is running properly to regularly index the 3DSpace Database

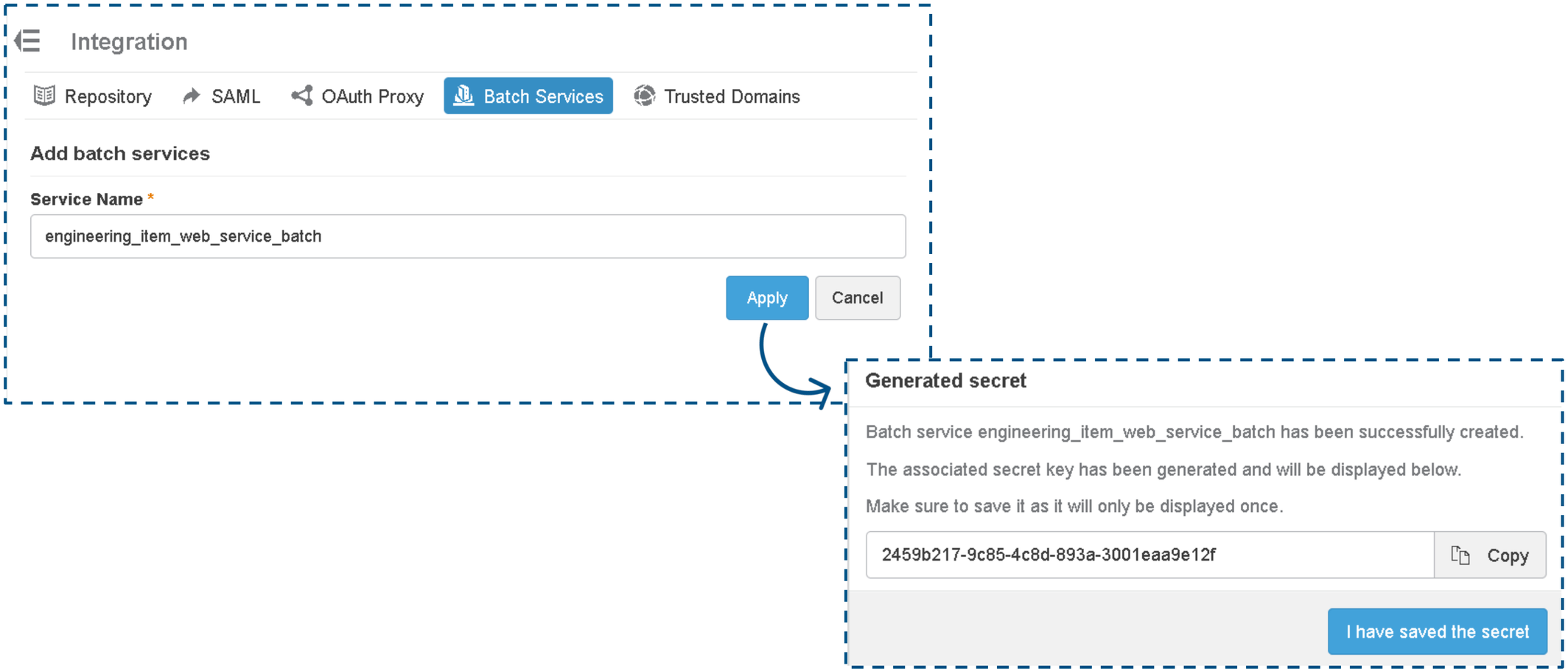
Generating Secret Key (1/2)

- Login with Administrator role and launch **3DPassport Control Center** widget from compass.
- Select **Integration** module and go to **Batch Services** tab.
- Click on add button in order to generate batch ticket name and secret key.



Generating Secret Key (2/2)

- Enter the Service name as “engineering_item_web_service_batch” and click on Apply.
- Copy the Secret key generated.

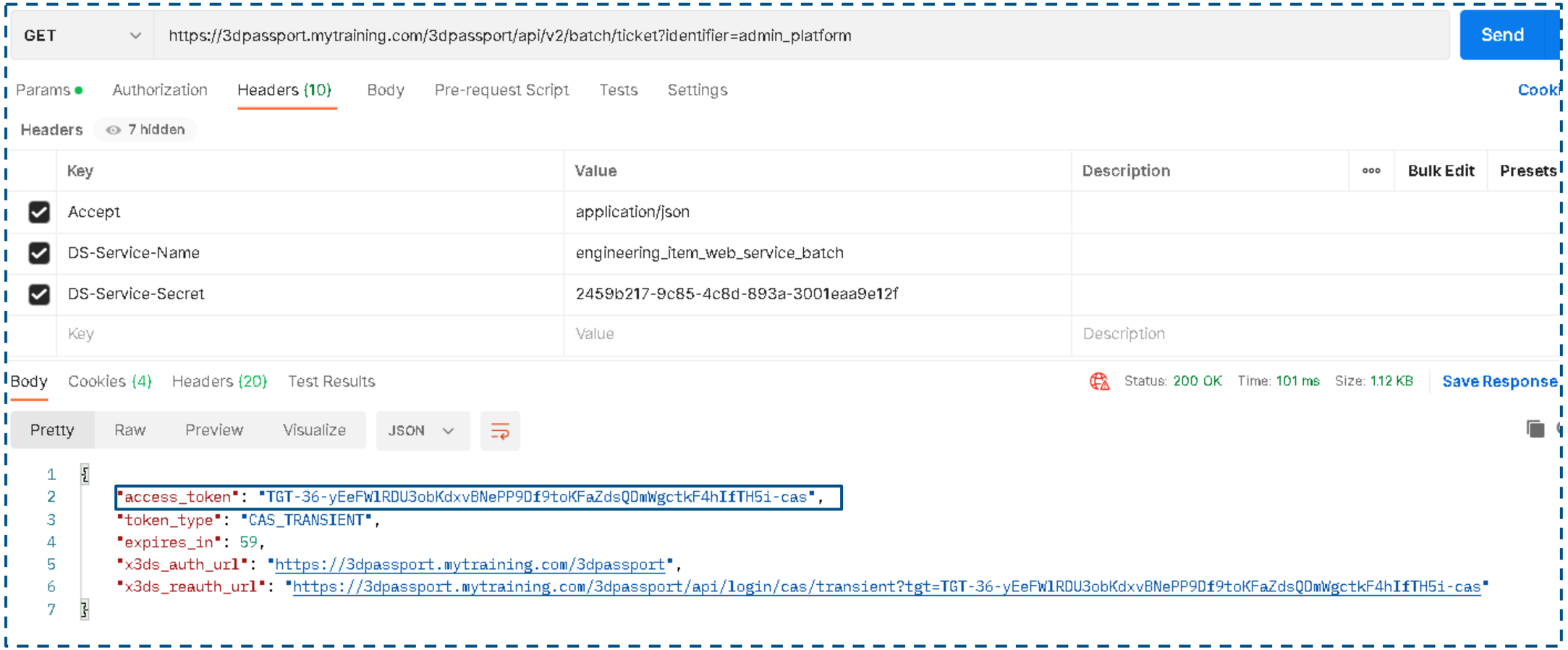


Consuming Secret Key (1/4)

- From the postman create a new request to create Transient Token.
- As per Developer Assistance to generate the Transient Token the following are applicable:
  - **Web-Service URI**: /api/v2/batch/ticket
  - **HTTP Verb**: GET
  - **Parameters**:

Parameter Name	Parameter Type	Mandatory	Description
identifier	Query	Yes	Example: admin_platform
service	Query	No	An UTF8 encoded URL.
Accept	Header	Yes	application_json
DS-Service-Name	Header	Yes	The named of the batch service declared in the 3DPassport Control Center application. Example :
DS-Service-Secret	Header	Yes	The secret key associated with the batch service . Example :

Consuming Secret Key (2/4)

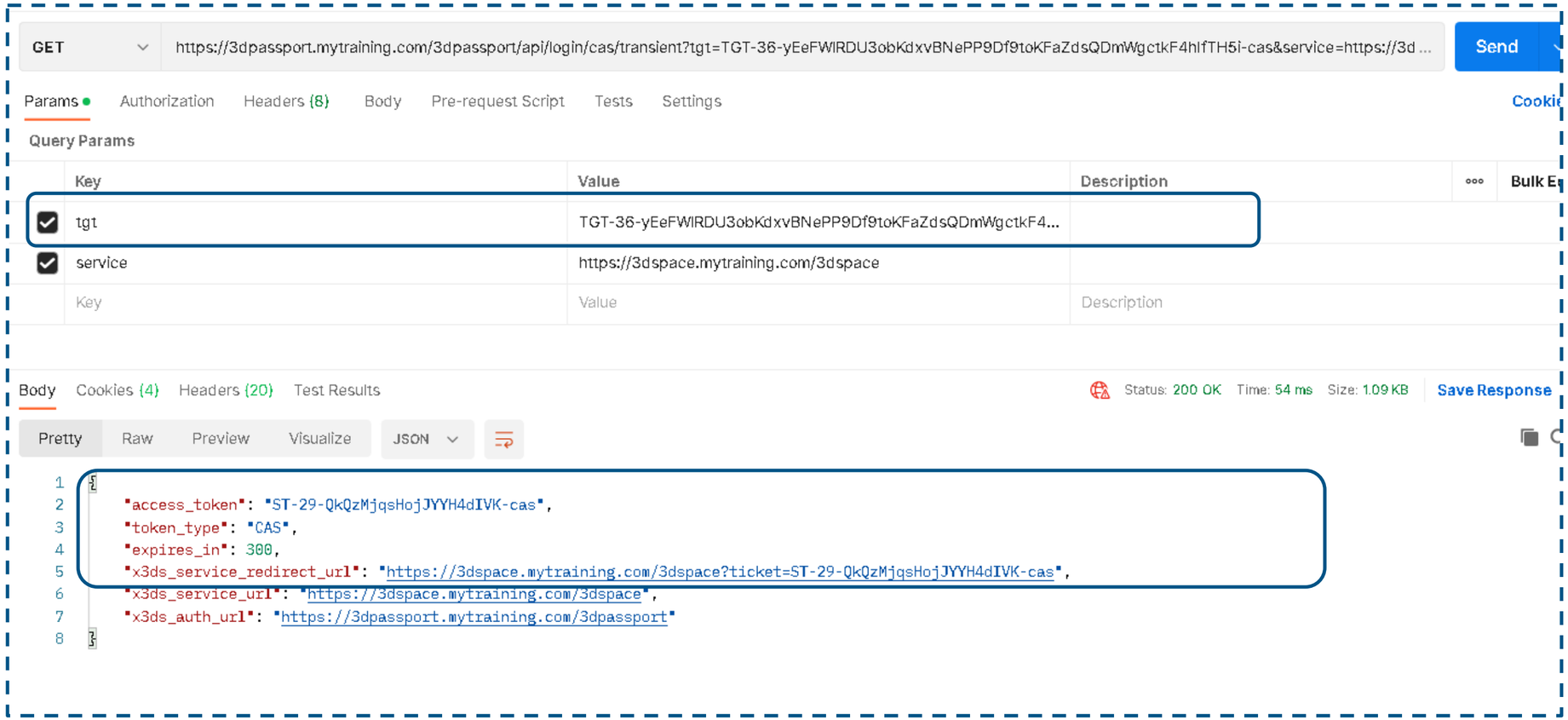


Consuming Secret Key (3/4)

- From the postman create a new request for CAS Authentication with Transient Token.
- As per Developer Assistance to go for CAS Authentication the following are applicable:
  - **Web-Service URI**: api/login/cas/transient
  - **HTTP Verb**: GET
  - **Parameters**:

Parameter Name	Parameter Type	Mandatory	Description
tgt	Query	Yes	The transient token. Example : TGT-36-yEeFWlRDU3obKdxvBNePP9Df9toKFaZdsQDmWgctkF4hIfTH5i-ca
service	Query	No	An UTF8 encoded URL. Example : https://3dspace.mytraining.com/3dpace
Accept	Header	Yes	Application-json

Consuming Secret Key (4/4)



Once the CAS Authentication is done you can call 3DSpace web service

Consuming Web-Services in 3DEXPERIENCE platform – Widgets

- Web-Services are widely used by 3DDashboard Widgets to interact with other platform services like 3DSpace, etc.

- For calling 3DEXPERIENCE platform web-services, 3DDashboard Widgets make use of the **DS/WAFData/WAFData** JavaScript API. Below is a JavaScript code snippet to demonstrate a web-service call.

```
WAFData.authenticatedRequest(urlWAF, {
    method: methodWAF,
    data: dataWAF,
    headers: headerWAF,
    type: 'json',
    onComplete: function (dataResp) {
        if(dataResp.msg==="OK"){
            myWidget.dataFull=JSON.parse(dataResp.data);
            myWidget.displayData(myWidget.dataFull);
        }else{
            widget.body.innerHTML += "<p>Error in WebService Response</p>";
            widget.body.innerHTML += "<p>"+JSON.stringify(dataResp)+"</p>";
        }
    },
    onFailure: function(error){
        widget.body.innerHTML += "<p>Call Failure</p>";
        widget.body.innerHTML += "<p>"+JSON.stringify(error)+"</p>";
    }
});
```

- Please refer to the *3DEXPERIENCE Widget Development Fundamentals* course for more technical details about web-services by 3DDashboard widgets.

Web-Services On-Cloud vs. On-Premise

- When it comes to deploying and consuming web-services, developers must keep the following in mind:
  - For an **On-Premise** installation, users would have access to the Application Servers at the backend. Hence, for On-Premise installation, it implies that:
    - Users can consume OOTB web-services
    - Users can also deploy customized web-services to implement customized business logic
  - For an **On-Cloud** installation, users do not have access to the backend Application Servers. Hence, for On-Cloud, we can safely conclude that:
    - Users can consume **only** OOTB web-services
    - Users cannot deploy custom web-services

OOTB Web-Services – Versioning

- As the 3DEXPERIENCE platform matures over time, the OOTB web-services may also be gradually updated to enhance their functionality.
- Hence, a proper version control is necessary for services offered by 3DEXPERIENCE platform OOTB.
- The version number of an OOTB web-service can be spotted on it URI, as highlighted in the below image:

- If an new version of a web-service is introduced, it would co-exist with the older one(s). This would ensure continuity for customers who have developed custom clients/batches to consume the older versions of the web-services.
  - The older versions may eventually get deprecated, in favor of the newer version(s).

Documentation for 3DEXPERIENCE platform Web-Services

DEVELOPER ASSISTANCE

Access information on how to use the V5, V6, and 3DEXPERIENCE development toolkits

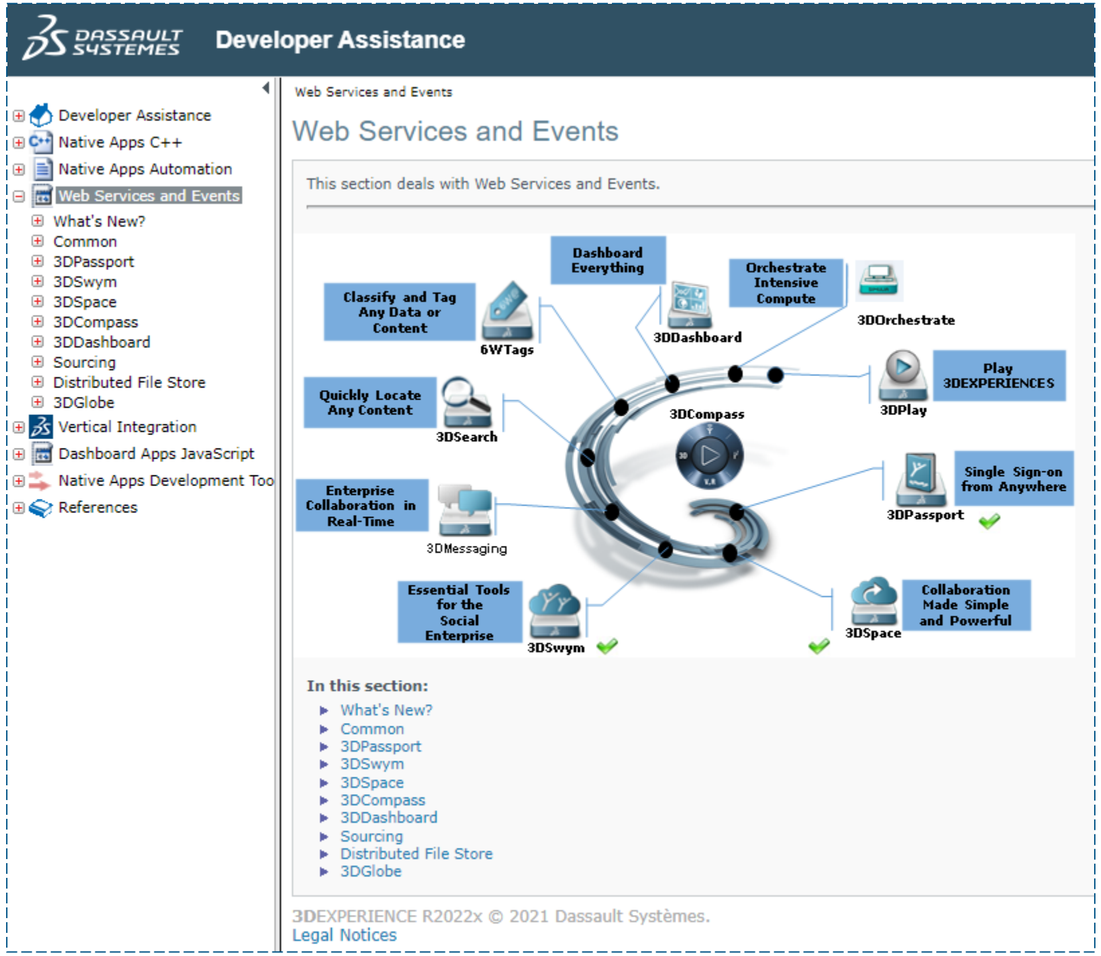
Here are the topics to be covered:

- ❑ *Consuming Web-Services in **3D**EXPERIENCE platform*
- ❑ Documentation for 3DEXPERIENCE platform Web-Services
- ❑ Tools for Testing Web-Services
- ❑ Demonstration with Engineering Web-Services

Web-Service Documentation

- Detailed documentation about **3D**EXPERIENCE platform web-services can be found on **Developer Assistance** at below link:

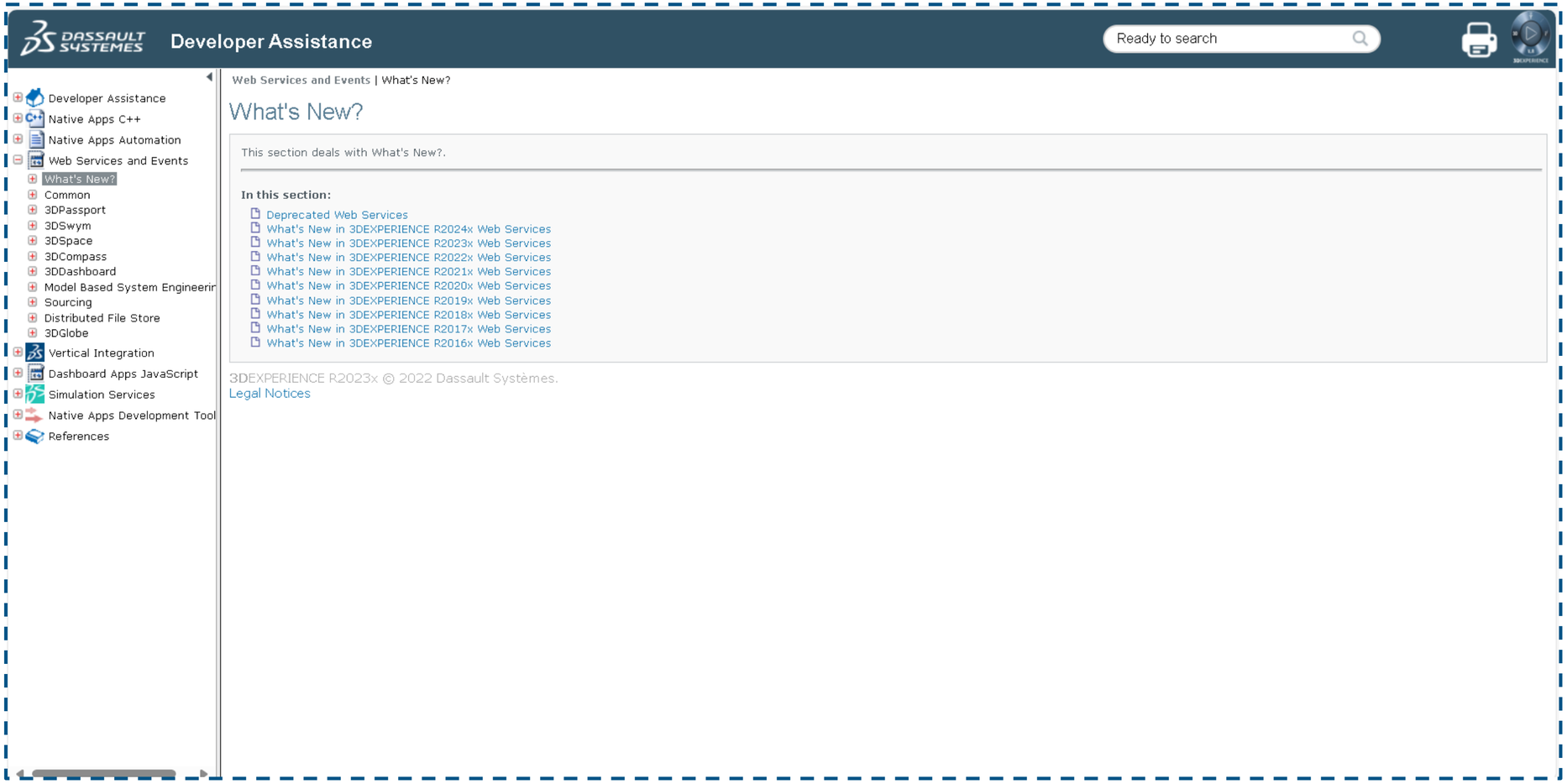
  https://media.3ds.com/support/documentation/developer/R2024x/en/DSDoc.htm?show=CAAiamREST/CAATciamRESTToc.htm



Web-Service Documentation – What's New

- You can refer to What's New section following the below link :

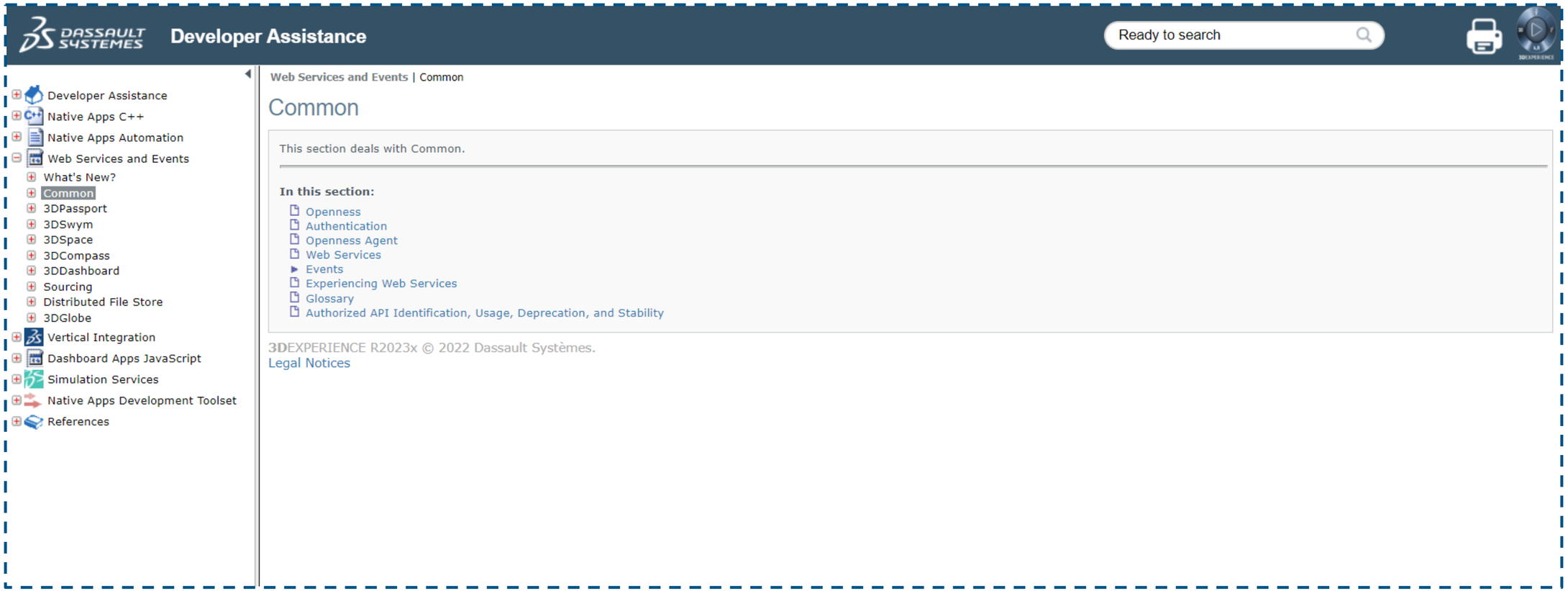
https://media.3ds.com/support/documentation/developer/R2024x/en/DSDoc.htm?show=CAAiamRESTNews/CAAiamRESTNewsToc.htm



Web-Service Documentation – Common

- You can refer to What's New section following the below link :

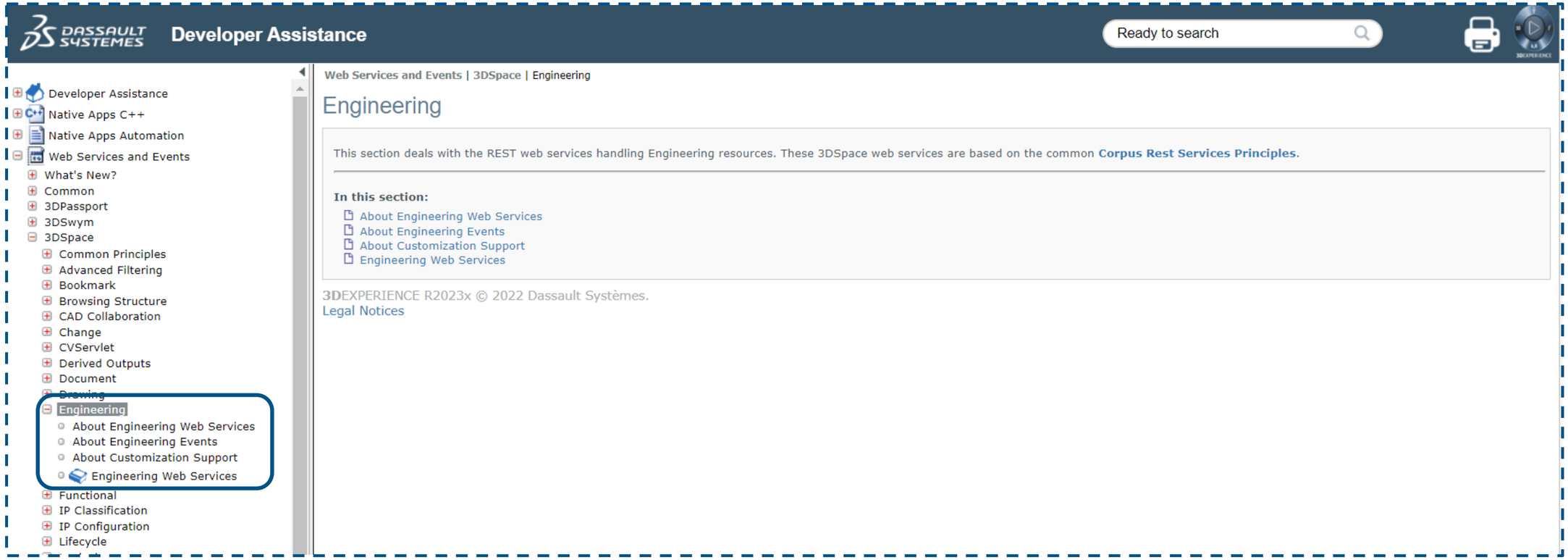
https://media.3ds.com/support/documentation/developer/R2024x/en/DSDoc.htm?show=CAAiamRESTCommon/CAAiamRESTCommonToc.htm



Web-Service Documentation – Web Service Family (1/5)

- This section details about the REST Web Service to manage the objects, their URL path, the Engineering Events Supported and the supported customization.

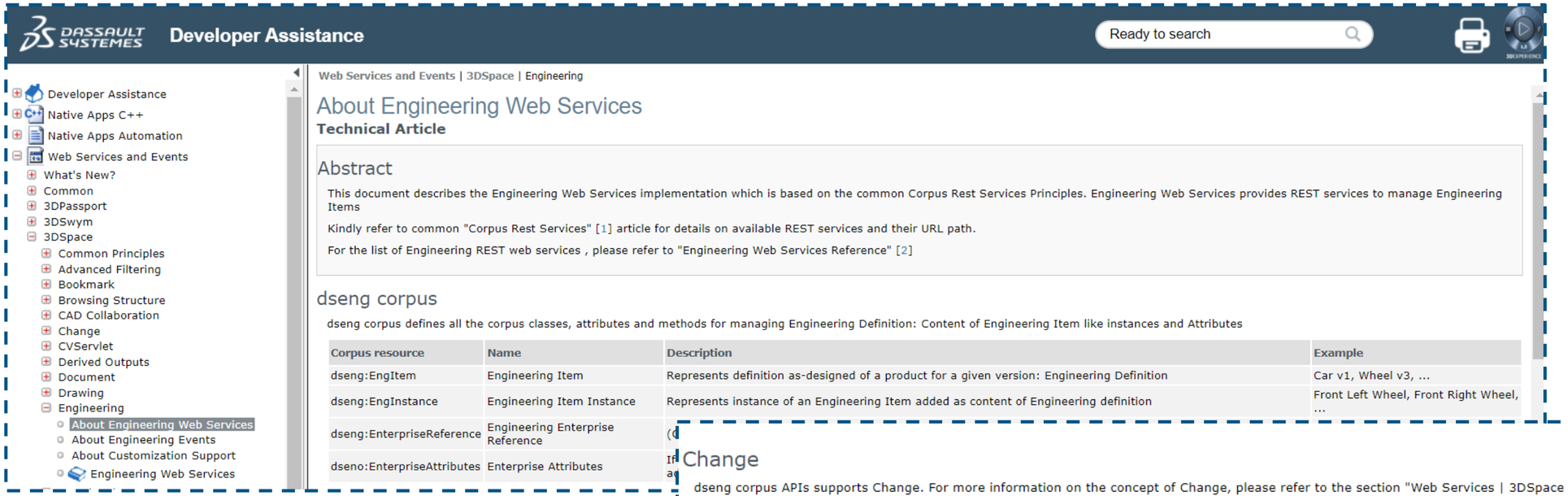
- These 3DSpace web services are based on the common **Corpus Rest Services Principles**.

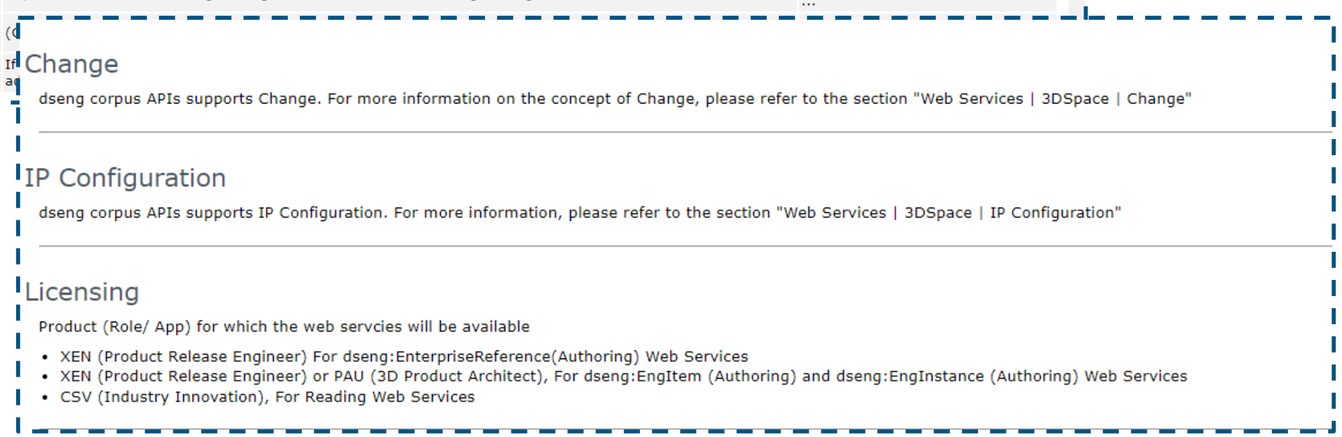


Web-Service Documentation – Web Service Family (2/5)

- This section describes the Engineering Web Services implementation which is based on the common Corpus Rest Services Principles.

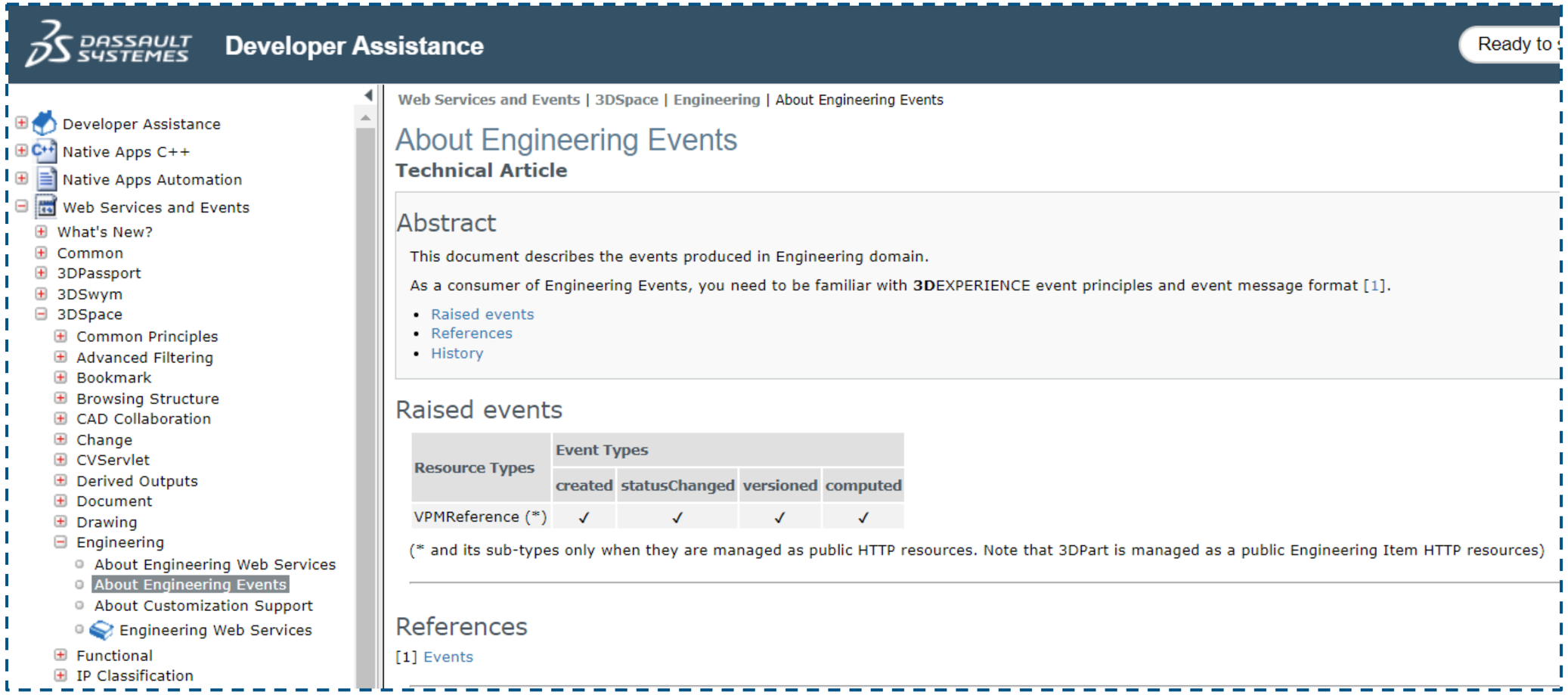
- From this section you can also refer to Licenses/Roles for which the web services will be available





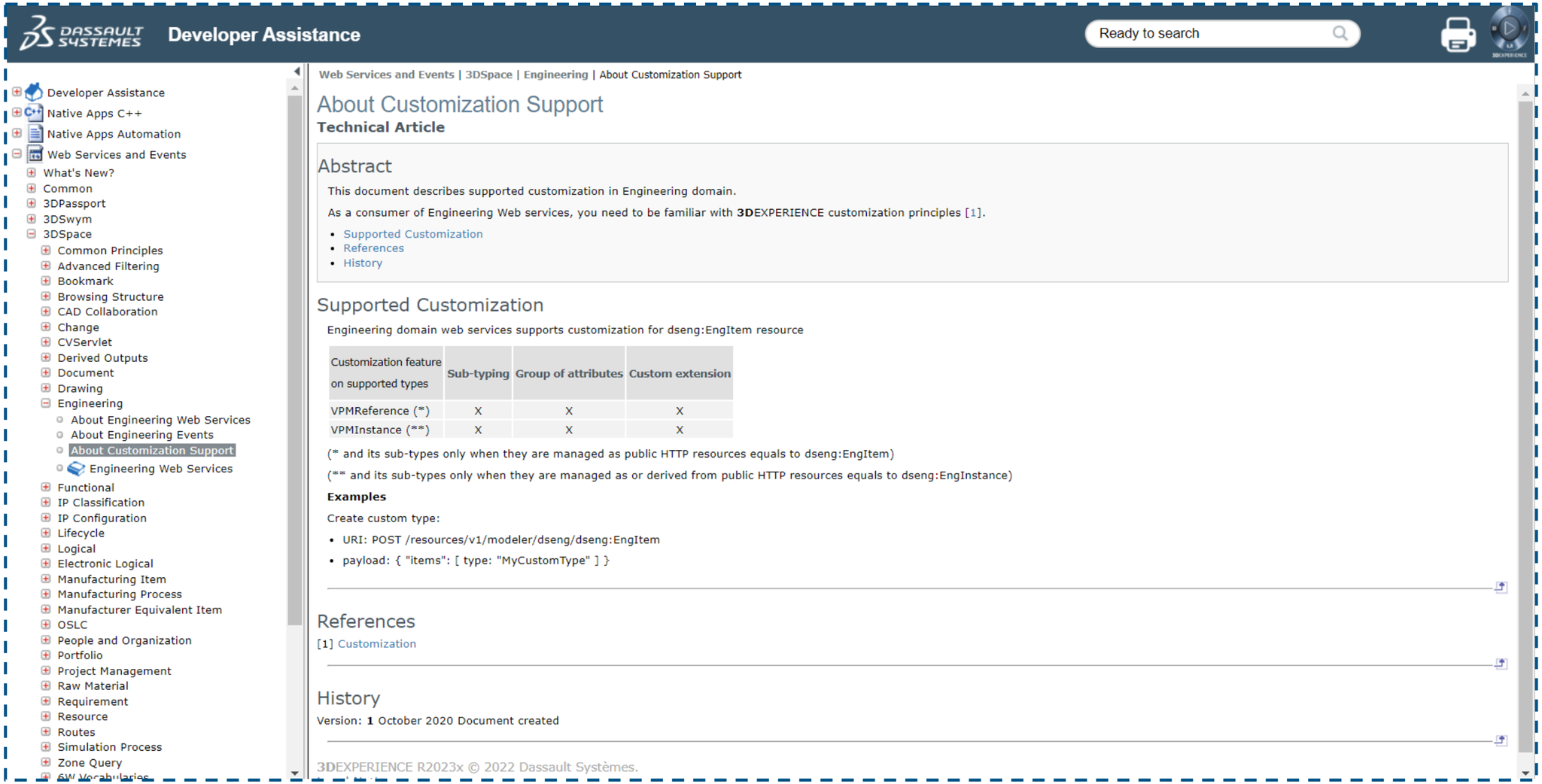
Web-Service Documentation – Web Service Family (3/5)

- This section describes about the supported events.



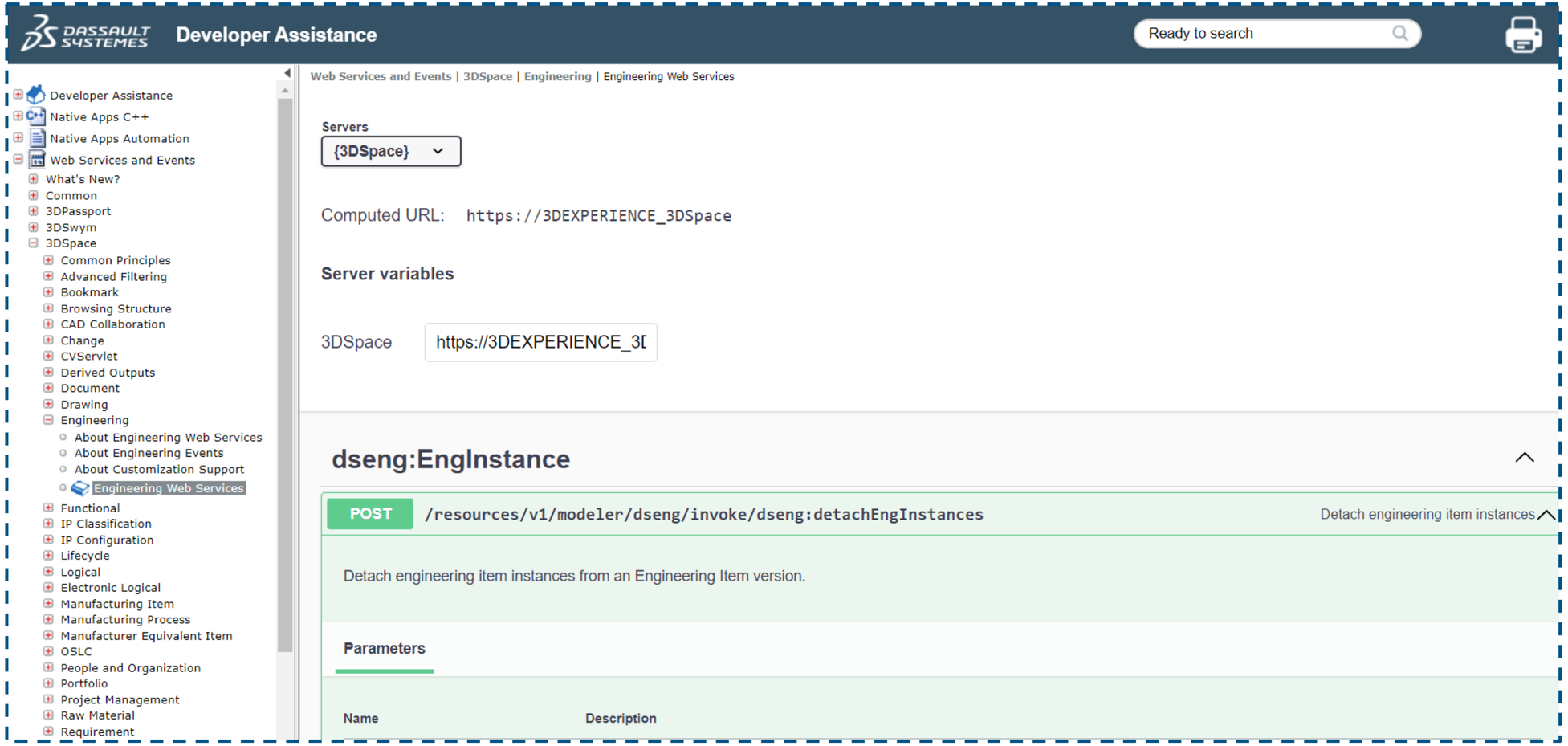
Web-Service Documentation – Web Service Family (4/5)

- This section describes about the supported customization



Web-Service Documentation – Web Service Family (5/5)

- This section details about the REST Web Service URL, the response code and different parameters which you can pass while calling the web service.



Headers

- Request headers are usually information about the action to perform. They do not contain information about the resource itself. This part is being passed by the message body.

- A header is a couple consisting of a header field name (not case sensitive) and a header field value (may be case sensitive).

Header Name	Used For	Header Value
Accept	Specify media types which are acceptable for the response. If the response is not binary (picture, video,..), the **3D**EXPERIENCE guidelines recommend JSON format. If the API proposes other kind of response formats (XML, HTML, text), it is specified in the API documentation. The expected response MIME type should be at least ensured for the successful response. In case of failure, refer to the API documentation. In call cases, to be rigorous, before processing the response body, check the Content-Type header in the response message.	Some examples: • application/json • image/png • application/octet-stream
Content-Type	Specify the media type of the request body.	Some examples: • application/json • application/xml
SecurityContext	An applicative header used in the context of **3D**EXPERIENCE services based on ENOVIA backend. 3DSpace being one example.	<role>.<organization>.<collabspace>
ENO_CSRF_TOKEN	The CSRF token for 3DSpace web service. For other **3D**EXPERIENCE services, refer to their documentation.	A value returned by another web service.

- The response headers are the same kind of headers as for the request.

Header Name	Used For	Header Value
Content-Type	Specify the media type of the response body.	Some examples: • application/json • application/xml
Content-Length	The count of chars in the response message body.	An integer.
Cache-Control	The cache control policy of the web services.	

- For more information refer Developer Assistance.

Mask

- Mask Infra generate JSON output based on mask string query parameter

- The response output format will vary depending on the mask parameter, When no mask is passed , default mask would be applied. Please find below the sample for different output format based on mask parameter value for the same URL

With Mask

```
GET {{3DSPACE_CAS_URL}}/resources/v1/modeler/dseng/dseng:EngItem/search?$mask=dsmveng:EngItemMask.Details
```


```
{
"owner": "Admin Platform",
"created": 04/08/2024 10:03:56 AM",
"description": "Engineering Item created for demonstration", "title": "Demo Engineering Item",
"type": "VPMReference",
"cestamp":
"42720E0800001A5C6436A00C00000962",
"revision": "A",
"organization": "Company Name",
"name": "prd-22903816-00000050",
modified: 04/08/2024 12:11:56 PM",
"id": "42720E0800001A5C6436820C00059407", "state": "PRIVATE",
"collabspace": "CDETraining",
"dseng: EnterpriseReference": {
"partNumber": "11992958"
}
```


Without Mask

```
GET {{3DSPACE_CAS_URL}}/resources/v1/modeler/dseng/dseng:EngItem/search
```


```
{
"owner": "Admin Platform",
"created": 04/08/2024 10:03:56 AM",
"description": "Engineering Item created for demonstration", "title": "Demo Engineering Item",
"type": "VPMReference",
"cestamp":
"42720E0800001A5C6436A00C00000962",
"revision": "A",
"organization": "Company Name",
"name": "prd-22903816-00000050",
"modified": 04/08/2024 12:11:56 PM",
"id": "42720E0800001A5C6436820C00059407",
"state": "PRIVATE",
'collabspace": "CDETraining"
}
```


- For more information refer Developer Assistance.

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.
  - $fields=all: Includes all available elements declared by the rest service even if turned off by the rest service by default.
  - $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.

- For more information refer Developer Assistance

CESTAMP(1/2)

- The **cestamp** (concurrent engineering stamp) is a mechanism that helps web service clients perform consistent data updates.

- A "fresh" cestamp needs to be passed in the request for a successful update of the data. Every time that the **3D**EXPERIENCE updates one or more attributes of an item, the associated cestamp attribute is automatically refreshed.

- Every time you updated the object properties the updated or fresh cestamp needs to be passed. While updating the properties of object the server compares the cestamp value which you have passed with the updated cestamp attribute value

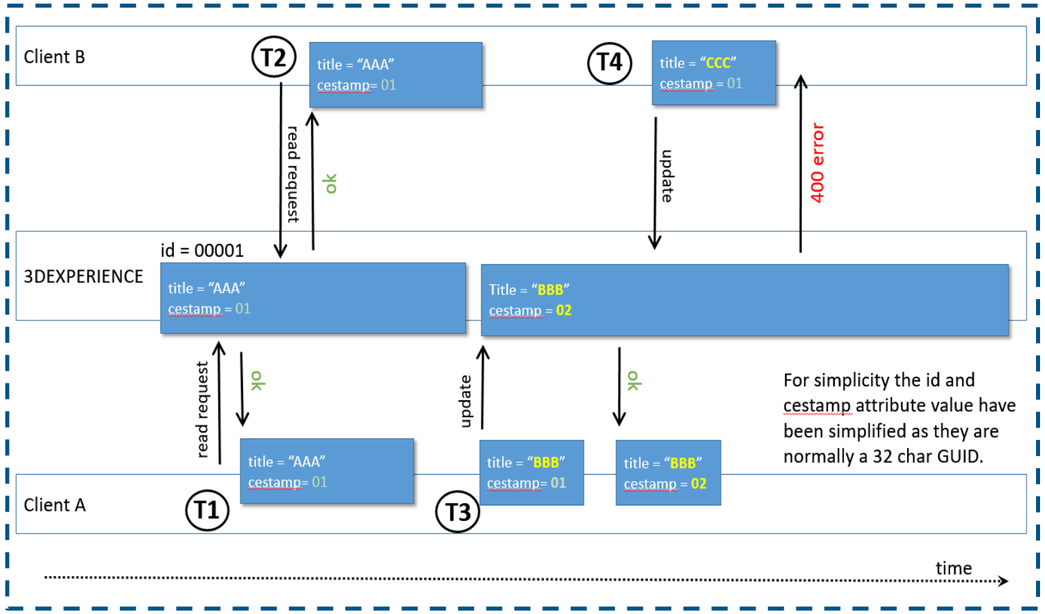
- If the cestamp value doesn't matches with the value you have passed then the object will not get updated and you will get 404 error.

```
"status": 400,
"message": "Your session object is now deprecated. Please update it with latest database information."
```

- Note that this mechanism is independent of the ENO_CSRF_TOKEN which is normally also required on most updates.

CESTAMP(2/2)

- Lets understand how we can use CESTAMP for updating the title of an object :
- In this scenario we have 2 web service clients; A and B. Time flows from left to right.
- At time T1, client A requests information for item id = 00001 (e.g. search/get). The response retrieves Title = “AAA” and cestamp=“01”
- Later on at time T2, client B requests the same information. The response retrieves Title = “AAA”, cestamp=“01”
- At time T3, client A updates the Title of the item to “BBB” by passing on the payload Title=“BBB” and cestamp=“01”.
- At this point the server compares the item cestamp with the one in the payload. Given that they are equal the server updates the information and generates a new cestamp.
- The response to T3 request retrieves the new Title (“BBB”) and new cestamp (“02”).
- At time T4, client B tries to update the Title of the item to “CCC” however it passes on the payload the old cestamp = “01”.
- The server compares the cestamps and, given that they are different, returns an error. The item title is therefore not updated.

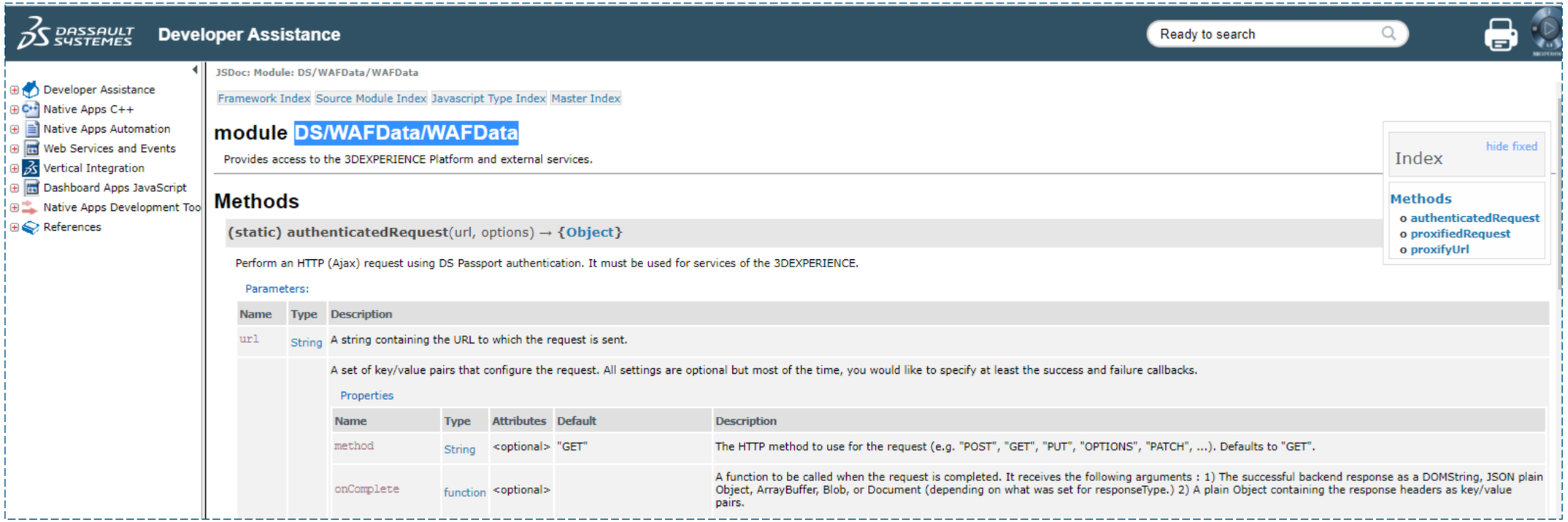



```
"status": 400,
"message": "Your session object is now deprecated. Please update it with latest database information."
```

Web-Service Documentation – WAFData

- More information regarding the DS/WAFData/WAFData JavaScript module can be found at the below path:

  https://media.3ds.com/support/documentation/developer/R2022x/en/DSDoc.htm?show=../generated/js/WebAppsFoundations-WAFData/module-DS_WAFData_WAFData.htm

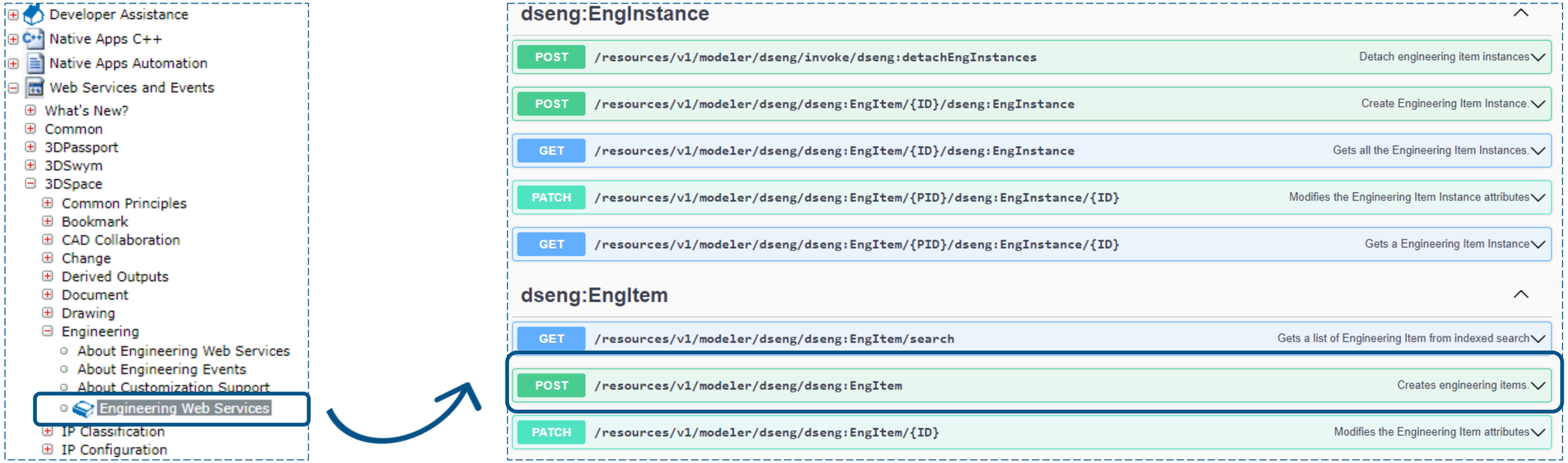


Reading Developer Assistance – Example

- Developer Assistance documentation for **3D**EXPERIENCE platform web-services are organized in an easy to search format, based on their owning services.

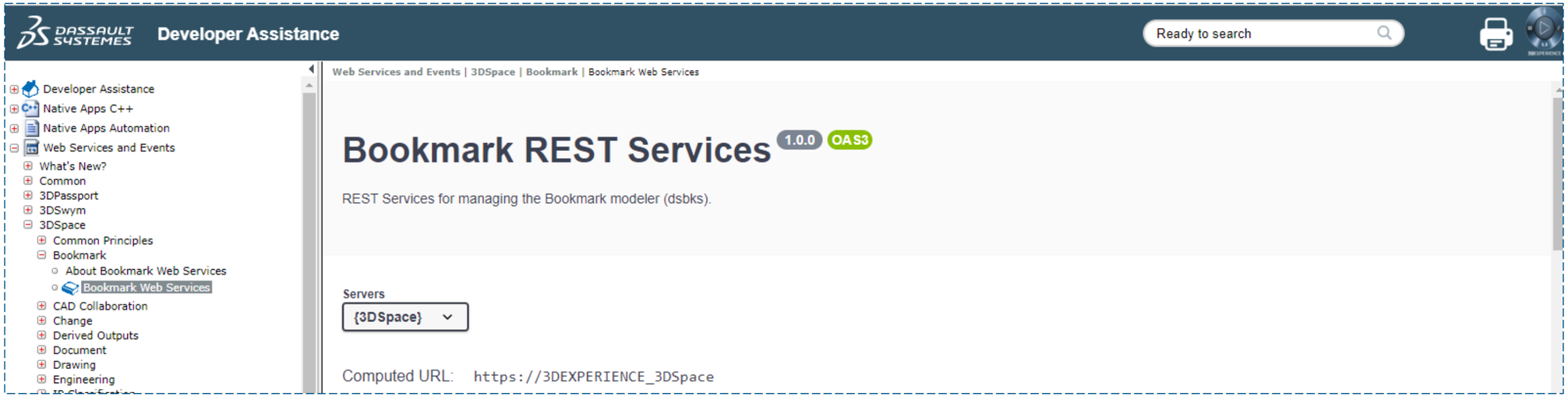
- In the following example, we shall try to find the documentation for web-service to create an *Engineering Item*.
  - Engineering Items come under the scope of 3DSpace services. Hence the relevant information would be found with the following navigation:

    Web Services and Events → 3DSpace → Engineering → Engineering Web Services
  - We can then explore the contents of *Engineering Web Services* to find information about web-service to **Create engineering items**.



Reading Developer Assistance – OAS3 (1/2)

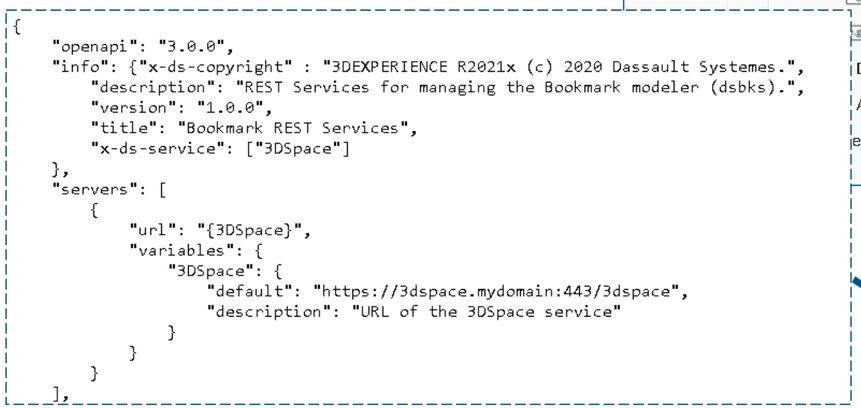
- The Developer Assistance documentation for **3D**EXPERIENCE platform web-services follows the OAS3 standards.
  - This can be observed at the beginning of every page documenting a particular category of Web-Services, as shown in the below example:

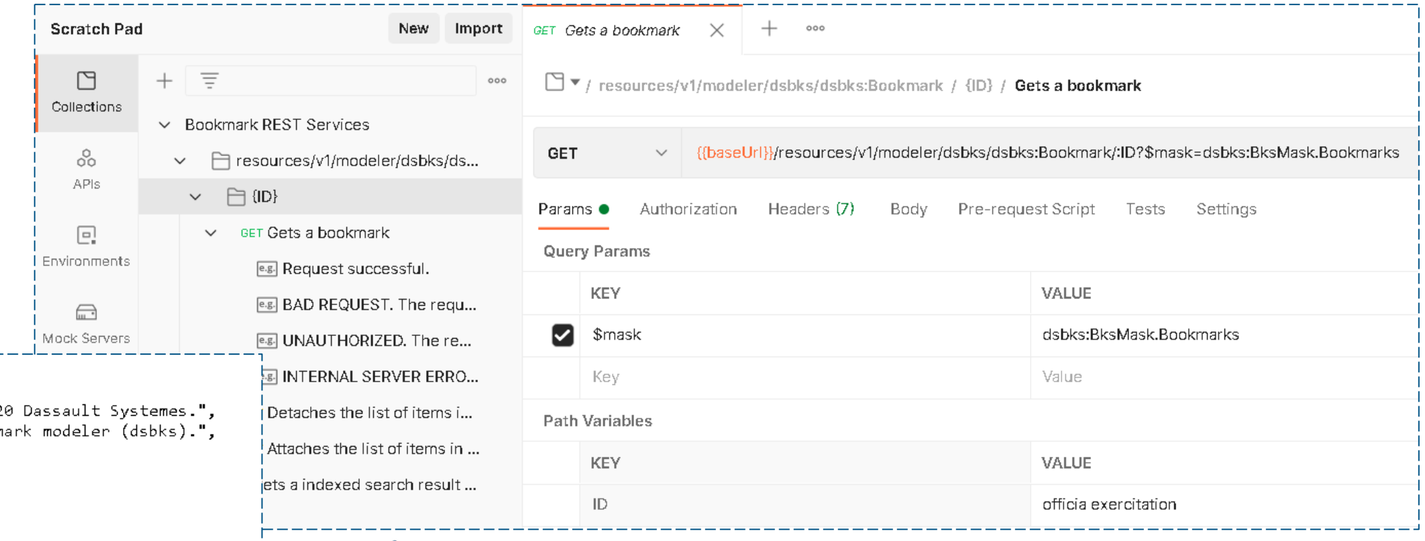


- The **OpenAPI Specification** (OAS) defines a standard, language-agnostic interface to RESTful APIs which allows both humans and computers to discover and understand the capabilities of the service without access to source code, documentation, or through network traffic inspection.
  - When properly defined, a consumer can understand and interact with the remote service with a minimal amount of implementation logic.

Reading Developer Assistance – OAS3 (2/2)

- Since Developer Assistance documentation for **3D**EXPERIENCE platform web-services implements OAS3, such documentation can be downloaded in JSON format.
  - The downloaded JSON file can then be uploaded in tools like *Postman* for easy and quick testing of the web-services.
- At the end of every Developer Assistance page documenting **3D**EXPERIENCE platform web-services, there is a link that allows us to download the documentation in JSON format.





Tools for Testing Web-Services

In this chapter, we will be introduced to some of the most popular tools that are used to test Web-Services

Here are the topics to be covered:

- *Consuming Web-Services in 3DEXPERIENCE platform*
- *Documentation for 3DEXPERIENCE platform Web-Services*
- Tools for Testing Web-Services
- Demonstration with Engineering Web-Services

Test Web-Services

- Testing of a Web-Service involves validating the service against use-cases to check the functionality, reliability, performance and security aspects of the application.

- There are many tools available in the market for testing web-services. Some of the most notable ones are as below:

  - RESTClient

  - Postman

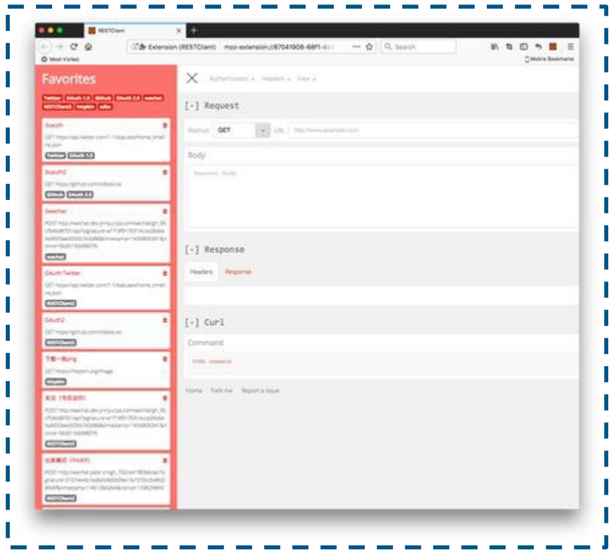
- It is to be noted that the above testing tools are external, industry standard tools.

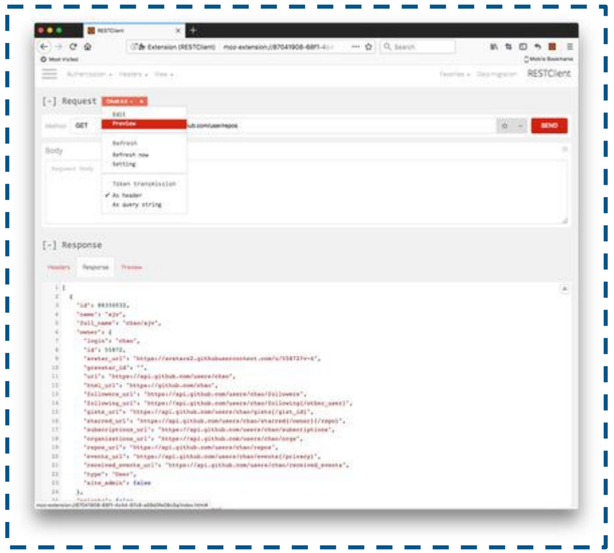
  - Since these tools are external to **3D**EXPERIENCE platform, we need to provide inputs to these tools for authentication for web-services which is needed for testing the developed APIs.

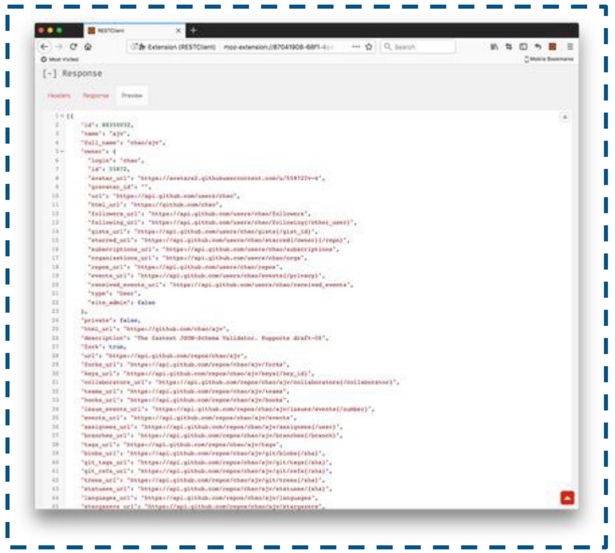
Test Web-Services – RESTClient

- **RESTClient** is a browser plugin that is designed specifically for Firefox browser.

- This tool is used to construct custom HTTP requests (custom method with resources URI and HTTP request body), and test the same directly against the server.

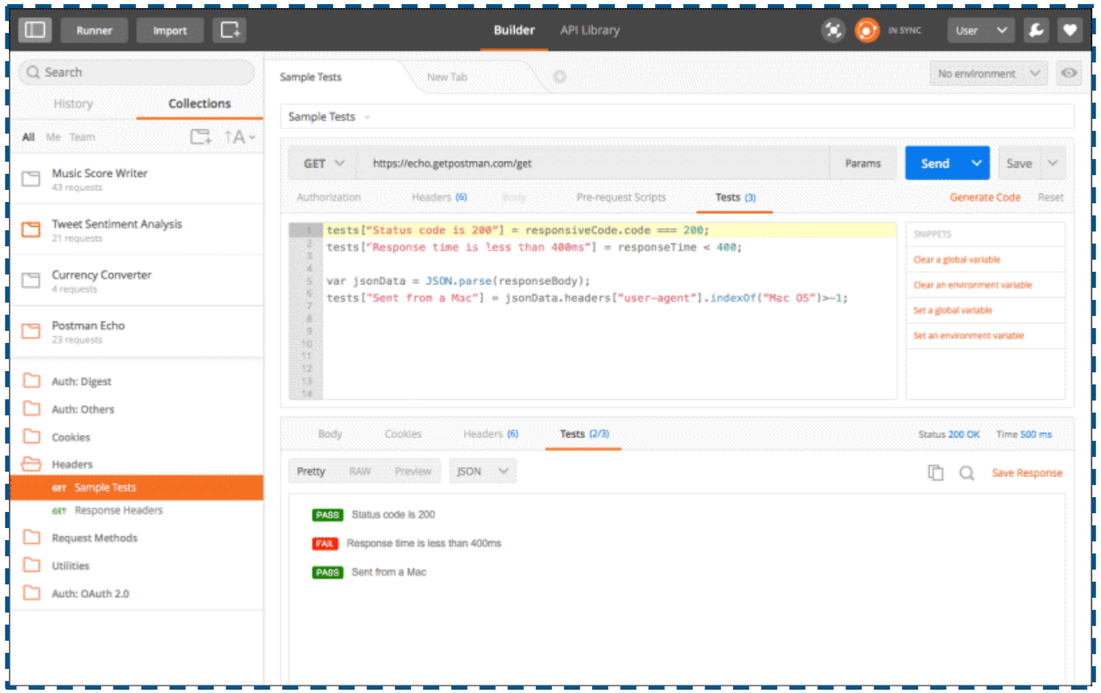




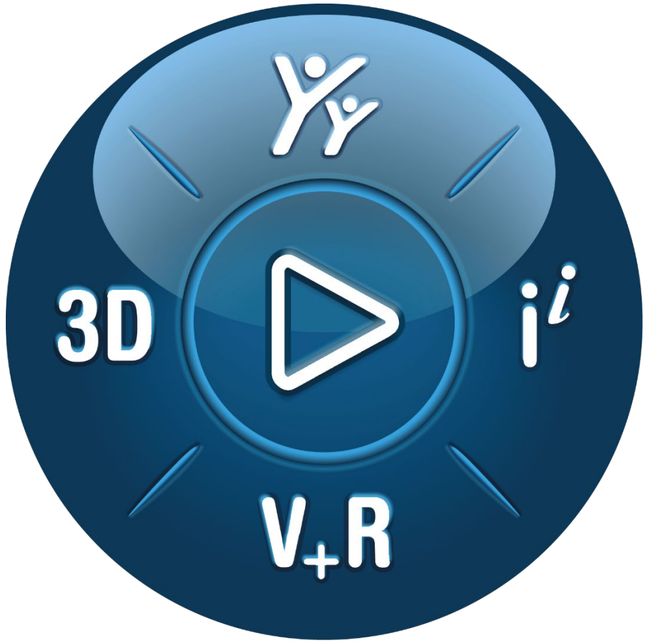


Test Web-Services – Postman

- **Postman** is a collaboration platform for API development.
- *Postman* allows you to perform the following:
  - Quickly test REST, SOAP or GraphQL requests directly against the server
  - Perform automation testing of developed API, as part of Continuous Integration (CI) of code.



Demonstration with Engineering Web-Services



3DEXPERIENCE®

In this lesson we shall demonstrate how to consume web-services, while using standard testing tools.
Here are the topics to be covered:

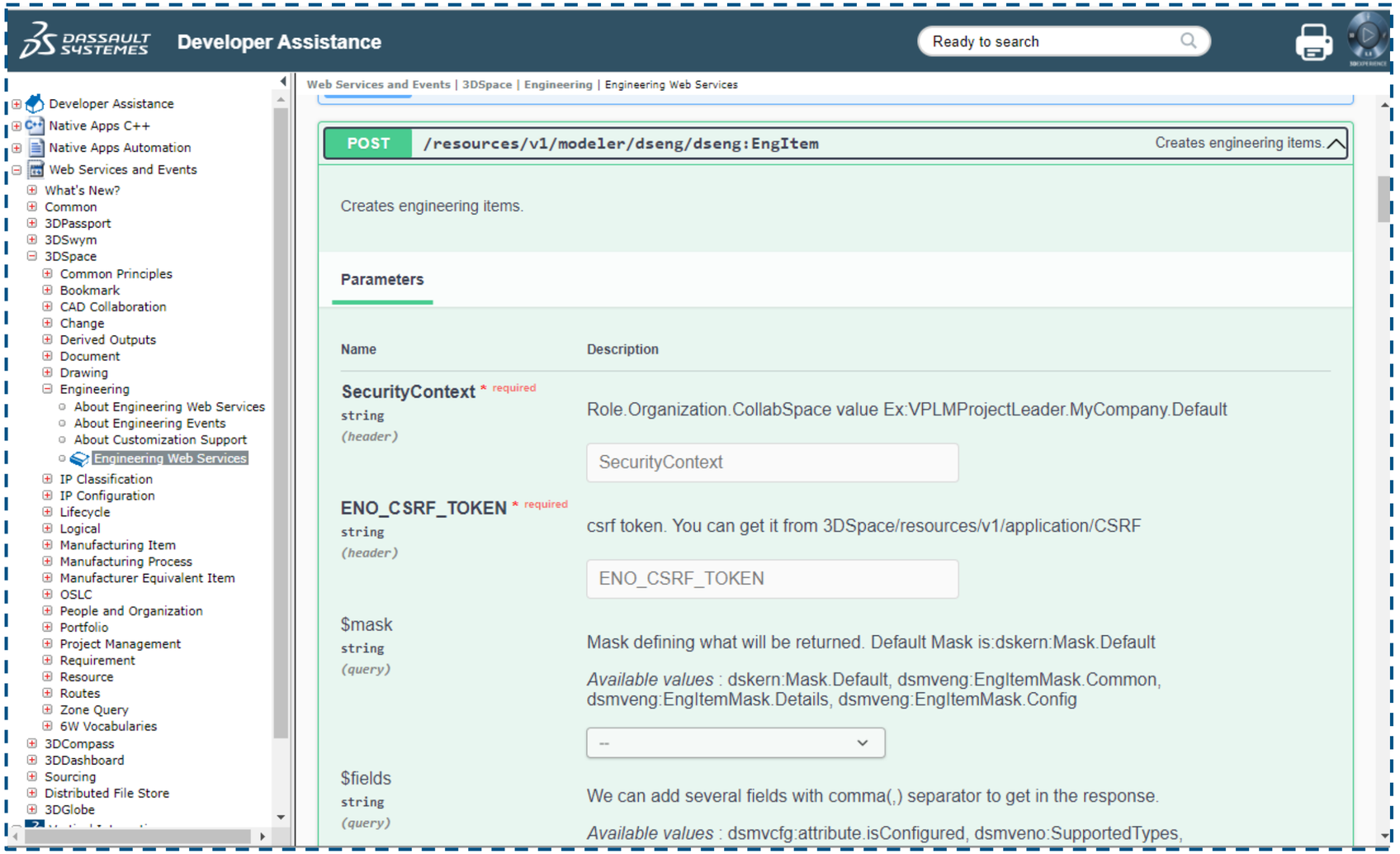
- ❑ *Consuming Web-Services in 3DEXPERIENCE platform*
- ❑ *Documentation for 3DEXPERIENCE platform Web-Services*
- ❑ *Tools for Testing Web-Services*
- ❑ Demonstration with Engineering Web-Services

Create Engineering Item – Demonstration – Introduction

- In this demonstration, we shall create an Engineering Item (type **VPMReference**) using Web-Service:
  - We shall use **Visual Studio Code**/**Postman** as the tool to call the web-services, and analyze their output
  - We shall refer to Developer Assistance to get information about the web-services that would be run, like, required parameters, expected output, etc.
- Following are the **prerequisites** for this demonstration:
  - User has valid **Product Release Engineer** role (**XEN** license)
  - If an On-Premises installation is used, then ensure the following:
    - **3D**EXPERIENCE platform is properly setup and configured
    - All services are up and running
    - 3DSpace Index service is running properly to regularly index the 3DSpace Database

Create Engineering Item – Demonstration – Developer Assistance (1/7)

- Let us take a look at the *Developer Assistance* documentation on web-services for creating *Engineering Items*. Below is a screen-capture of the same.



Create Engineering Item – Demonstration – Developer Assistance (2/7)

- As per Developer Assistance on creating Engineering Item using Web-Services, the following are applicable:
  - **Web-Service URI**: /resources/v1/modeler/dseng/dseng:EngItem
  - **HTTP Verb**: POST
  - **Parameters**:

Parameter Name	Parameter Type	Mandatory	Description
SecurityContext	Header	Yes	Example: VPLMProjectLeader.MyCompany.Default
ENO_CSRF_TOKEN	Header	Yes	Can be obtained from <3DSpace_URL>/resources/v1/application/CSRF
$mask	Query	No	Mask defining the data that would be returned. Possible values are: • dskern:Mask.Default (Default value) • dsmveng:EngItemMask.Common • dsmveng:EngItemMask.Details • dsmveng:EngItemMask.Config
$fields	Query	No	We can add several fields with comma(,) separator to get in the response. Possible values are: • dsmvcfg:attribute.isConfigured • dsmveno:SupportedTypes • dsmveno:CustomerAttributes

Create Engineering Item – Demonstration – Developer Assistance (3/7)

- Request Body: Request body should contain details of Engineering Item to be created, in the below format:

```
{
  "items": [
    {
      "attributes": {
        "title": "Engineering Item Title ",
        "description": " Engineering Item Description ",
        "dseng:EnterpriseReference": {
          "partNumber": "<Part Number Value>"
        },
        "dseno:EnterpriseAttributes": {
          "Attribute DB Name": "<Attribute DB Value>"
        }
      }
    }
  ]
}
```

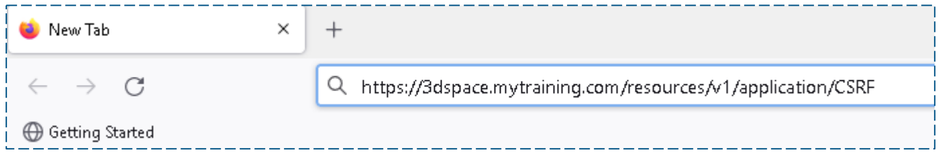
Create Engineering Item – Demonstration – Developer Assistance (4/7)

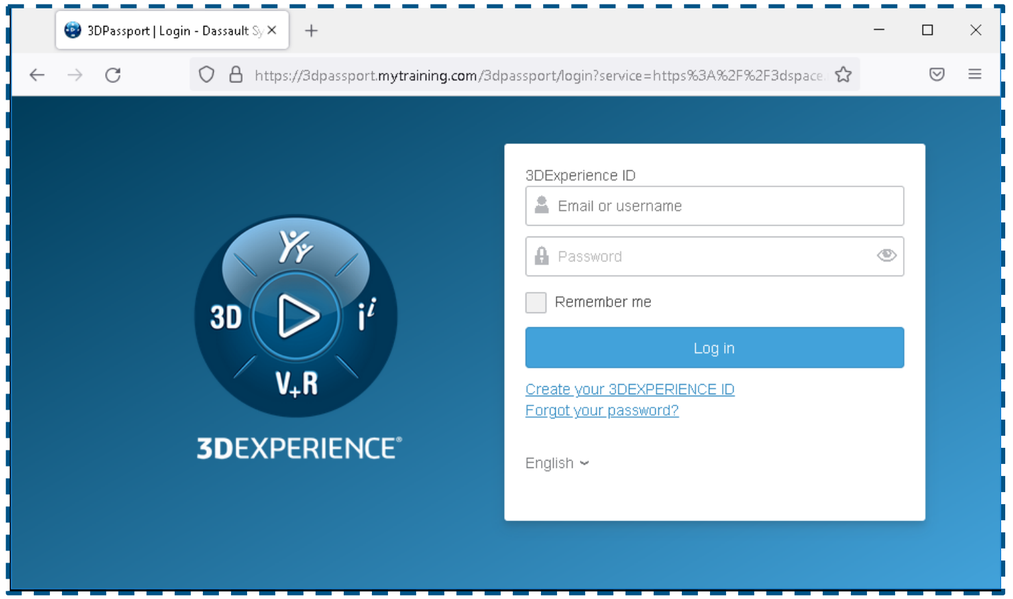
- If we analyze the parameters, we would see that we need to first obtain ENO_CSRF_TOKEN.

- As per Developer Assistance, ENO_CSRF_TOKEN can be obtained by calling the URL:

  `<3DSpace_URL>/resources/v1/application/CSRF`

- However, if we try to simply call the above URL from a browser, we would see that the call would redirect to 3DPassport Login page, as shown below:





The URL for 3DSpace service used in demonstration is:

https://3dspace.mytraining.com/3dspace

- This means that we should perform *CAS Authentication* first, before we can fetch CSRF Token for our web-service.

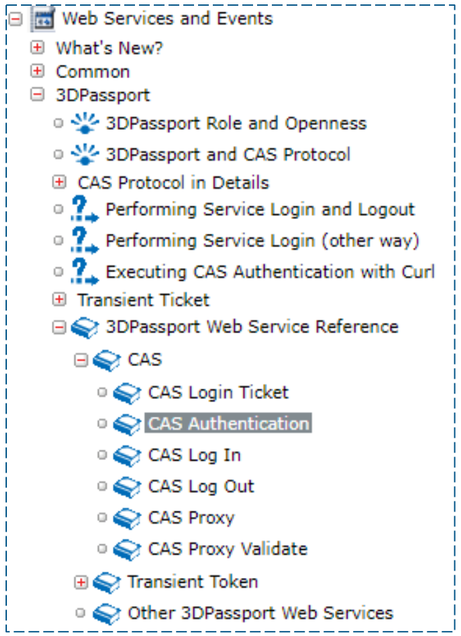
Create Engineering Item – Demonstration – Developer Assistance (5/7)

- Let us take a look at the *Developer Assistance* documentation for **3DPassport CAS Authentication** Web-Service:
  - **URL**: `<3DPassport_URL>/login`
  - **HTTP Verb**: POST
  - **Parameters**:

Parameter Name	Parameter Type	Mandatory	Description
service	Query	No	The URL of the service to be redirected to after successful authentication
Content-Type	Header	Yes	The POST body message MIME format. Value: application/x-www-form-urlencoded;charset=UTF-8

- **Message Body**: `lt=<loginticket>&username=<username>&password=<password>`

Key	Mandatory	Description
username	Yes	The username or email (UTF8 en encoded) of the user to authenticate
password	Yes	The password (UTF8 en encoded) of the user to authenticate
lt	Yes	The login ticket
rememberMe	No	"no" to keep the CASTGC cookie 2H (OnPremises) or 24H (OnCloud) instead of 1 week.

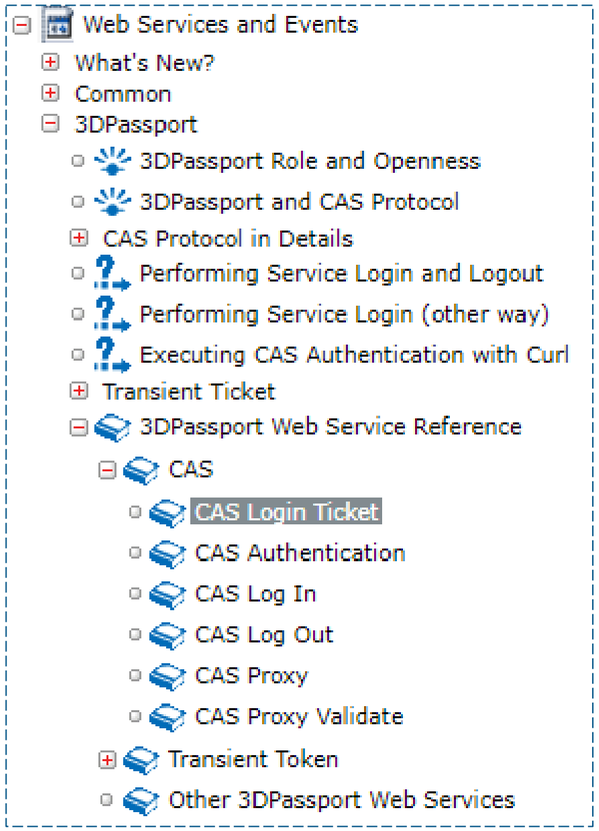


One of the mandatory parameters is "lt" or Login Ticket, which can be obtained from a separate 3DPassport web-service that returns login-tickets. This is described in the following section.

Create Engineering Item – Demonstration – Developer Assistance (6/7)

- Let us take a look at the *Developer Assistance* documentation for **3DPassport CAS Login Ticket** Web-Service:
  - **URL**: <3DPassport_URL>/login
  - **HTTP Verb**: GET
  - **Parameters**:

Parameter Name	Parameter Type	Mandatory	Description
action	Query	Yes	Value: get_auth_params
Accept	Header	Yes	The response MIME format. Value: application/json



Create Engineering Item – Demonstration – Developer Assistance (7/7)

- In the following exercise, we shall call OOTB (Out-Of-The-Box) 3DEXPERIENCE platform Web-Services, as previously mentioned, to create Engineering Items.
- A similar procedure may be adopted to use other Web-Services as well.
- Before executing any 3DEXPERIENCE platform web-service, the developer must first refer to Developer Assistance to thoroughly understand all the prerequisites and dependencies.

DEVELOPER ASSISTANCE

Access information on how to use the V5, V6,
and 3DEXPERIENCE development toolkits

Exercise: Create Engineering Item using Web-Service

- In this exercise, we shall demonstrate the manual creation of an *Engineering Item* and update EIN on *Engineering Item* by Web-Services.

- The following are the features of this exercise:
  - We shall use POSTMAN application for this demonstration
  - The web-services would be called based on information obtained from Developer Assistance documentation.

1.5 hours

Create Engineering Item – Demonstration – Steps (1/12)

Setup Postman Environment:

- ❑ Open Postman. Navigate as follows:

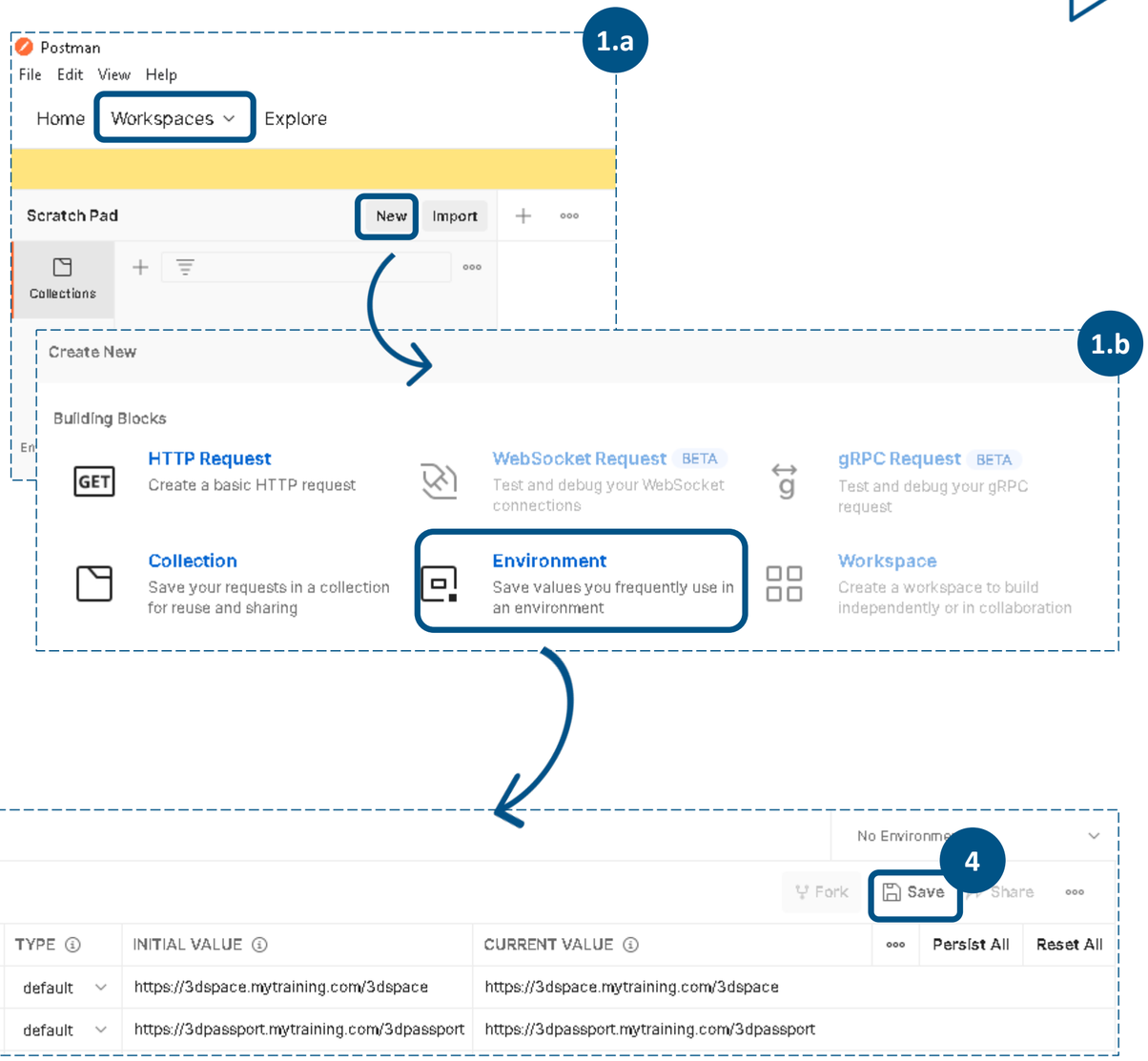
  Workspaces tab → New → Environment

- ❑ Rename the environment to **PATH_VARIABLES**

- ❑ Create two environment variables as mentioned below:

Variable Name	Initial Value
3DSPACE_CAS_URL	Your configured 3DSpace URL
3DPASSPORT_URL	Your configured 3DPassport URL

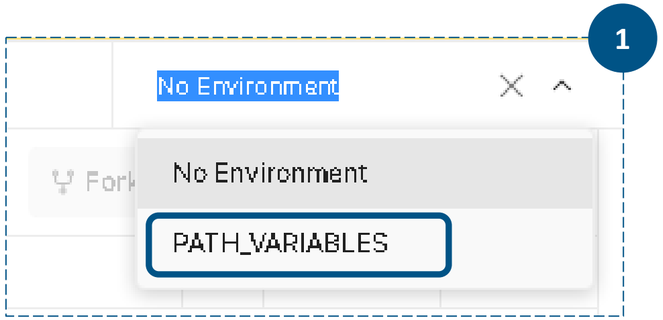
- ❑ Click on **Save** button.

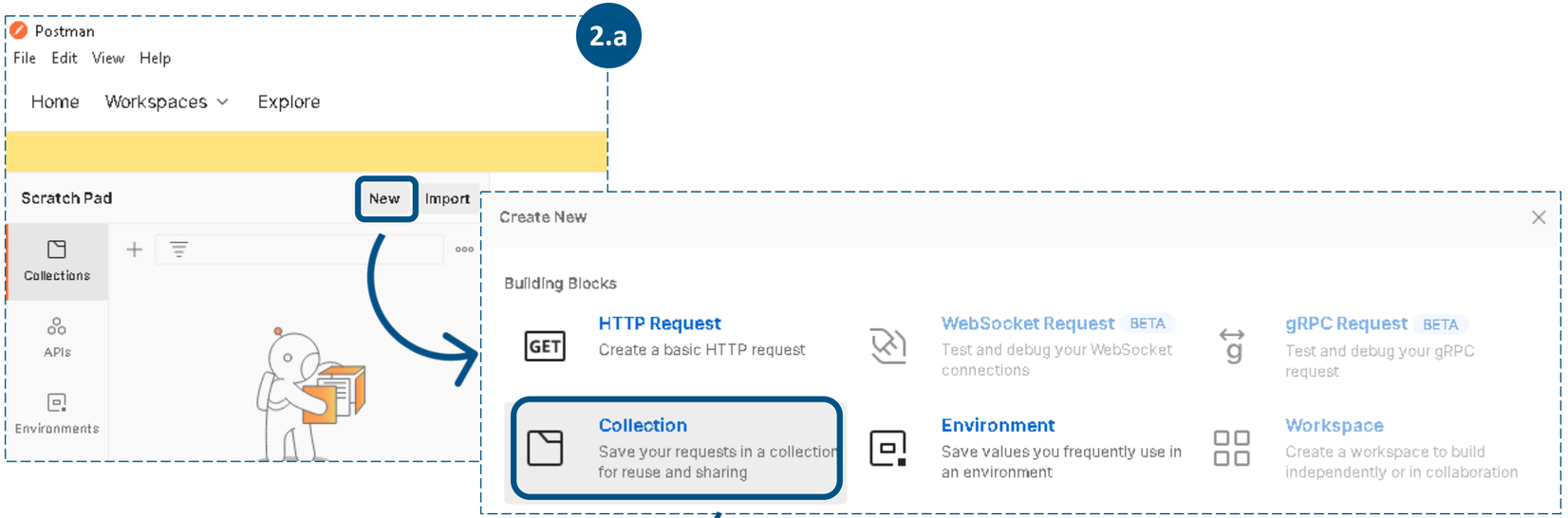


Create Engineering Item – Demonstration – Steps (2/12)

Create New Collection:

- From the list of defined environments, choose **PATH_VARIABLES**.
- Navigate as follows: New → Collection
- Rename the newly created collection to **Web-Service Demo**





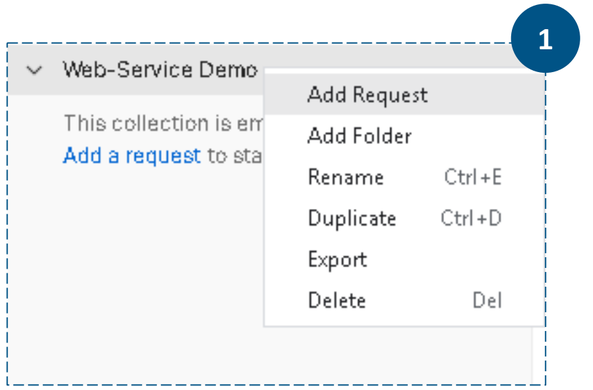
Create Engineering Item – Demonstration – Steps (3/12)

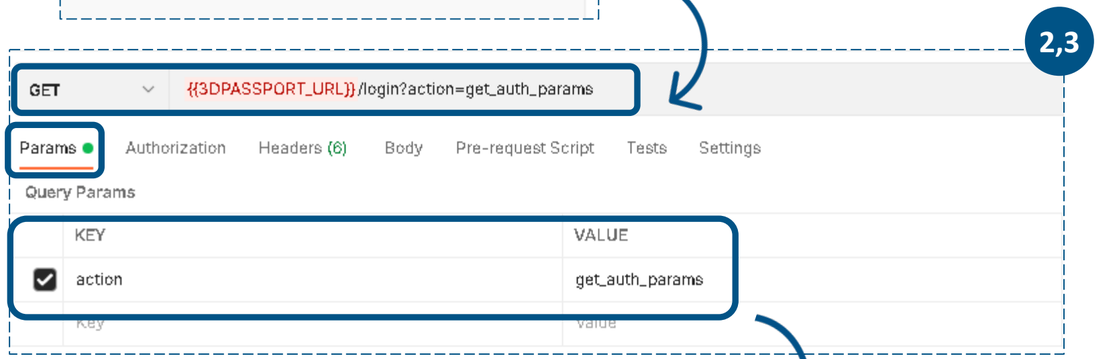
Create Request for CAS Login Ticket:

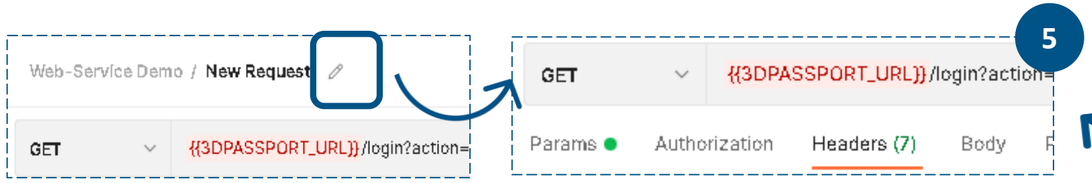
After analyzing the *Developer Assistance* documentation as mentioned in the previous pages, it can be logically concluded that the first web-service that needs to be called is the one to fetch *CAS Login Ticket*.

- ❑ Under **Collections** tab, right-click on **Web-Service Demo**, and select **Add Request** option.
- ❑ Select the HTTP verb as **GET**, and *Enter request URL*, as mentioned below:

  `{{3DPASSPORT_URL}}/login`
- ❑ Under **Params** section, enter Query parameter, as shown in screenshot.
- ❑ Under **Headers** sections, enter parameters as shown in screenshot.
- ❑ Rename the request to **CAS LOGIN TICKET**, as shown below, and **Save** the changes.







Create Engineering Item – Demonstration – Steps (4/12)

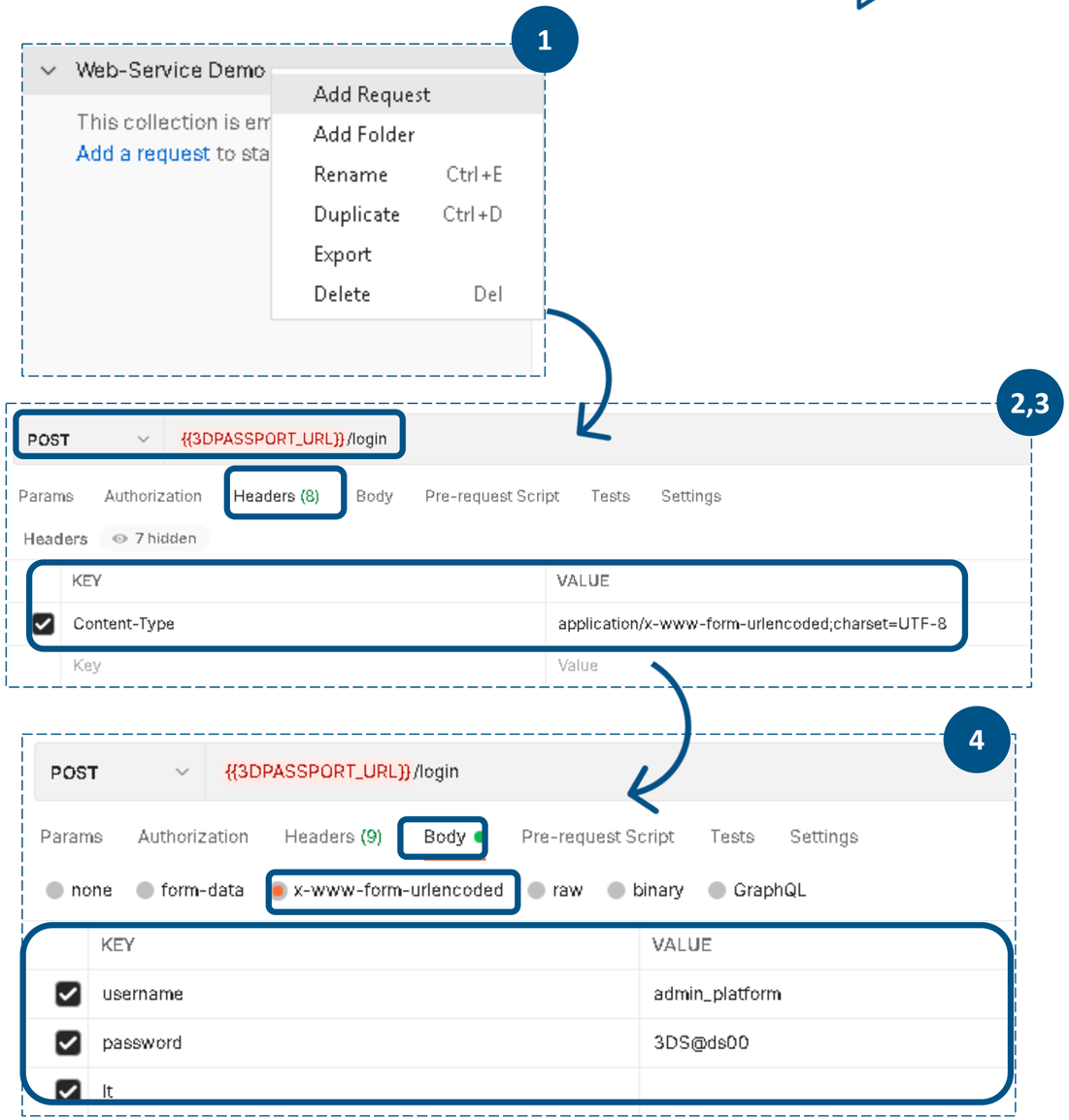
Create Request for CAS Authentication:

1. Under **Collections** tab, right-click on **Web-Service Demo**, and select **Add Request** option.
2. Select the HTTP verb as **POST**, and *Enter request URL*, as mentioned below:

   `{{3DPASSPORT_URL}}/login`
3. Under **Headers** section, enter parameter, as shown in screenshot.
4. Under **Body** section, choose option **x-www-form-urlencoded**. Enter the following key-value pairs:

Key	Value
username	User Name of user having Product Release Engineer role
password	Password of above user
lt	*Empty*. We will copy Login Ticket from output of CAS Login Ticket web-service, during execution.

5. Rename the request to **CAS AUTHENTICATION**, as shown below, and **Save** the changes.



Create Engineering Item – Demonstration – Steps (5/12)

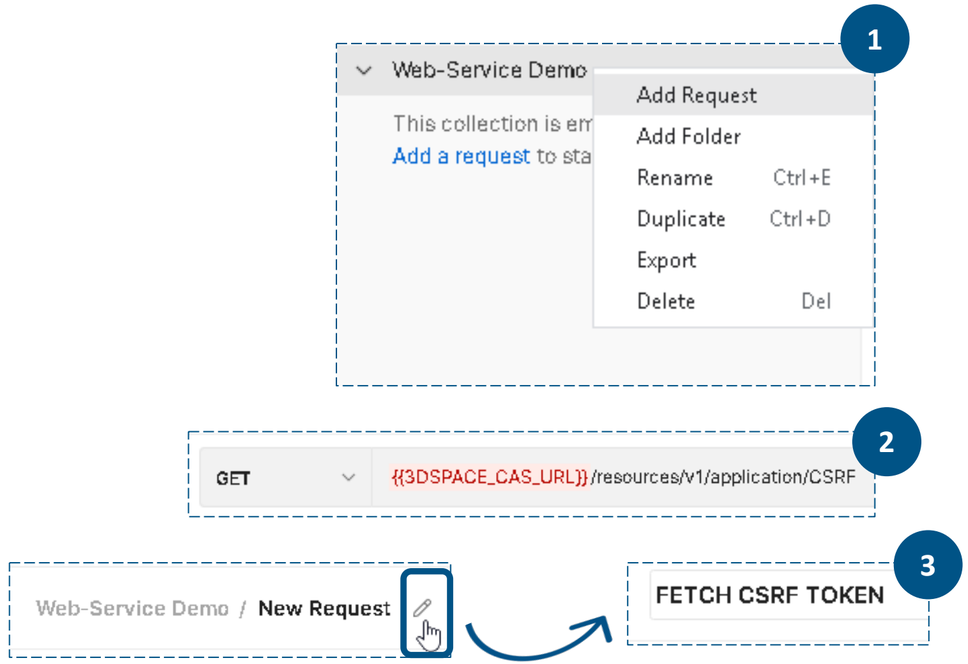
Create Request for fetching CSRF Token:

1. Under **Collections** tab, right-click on **Web-Service Demo**, and select **Add Request** option.

2. Select HTTP verb as **GET**, and *Enter request URL*, as mentioned below:

   ```
   {{3DSPACE_CAS_URL}}/resources/v1/application/CSRF
   ```

3. Rename the request to **FETCH CSRF TOKEN**, as shown, and **Save** the changes.



Create Engineering Item – Demonstration – Steps (6/12)

Create Request for creating Engineering Item:

1. Under **Collections** tab, right-click on **Web-Service Demo**, and select **Add Request** option.

2. Select HTTP verb as **POST**, and *Enter request URL*, as mentioned below:

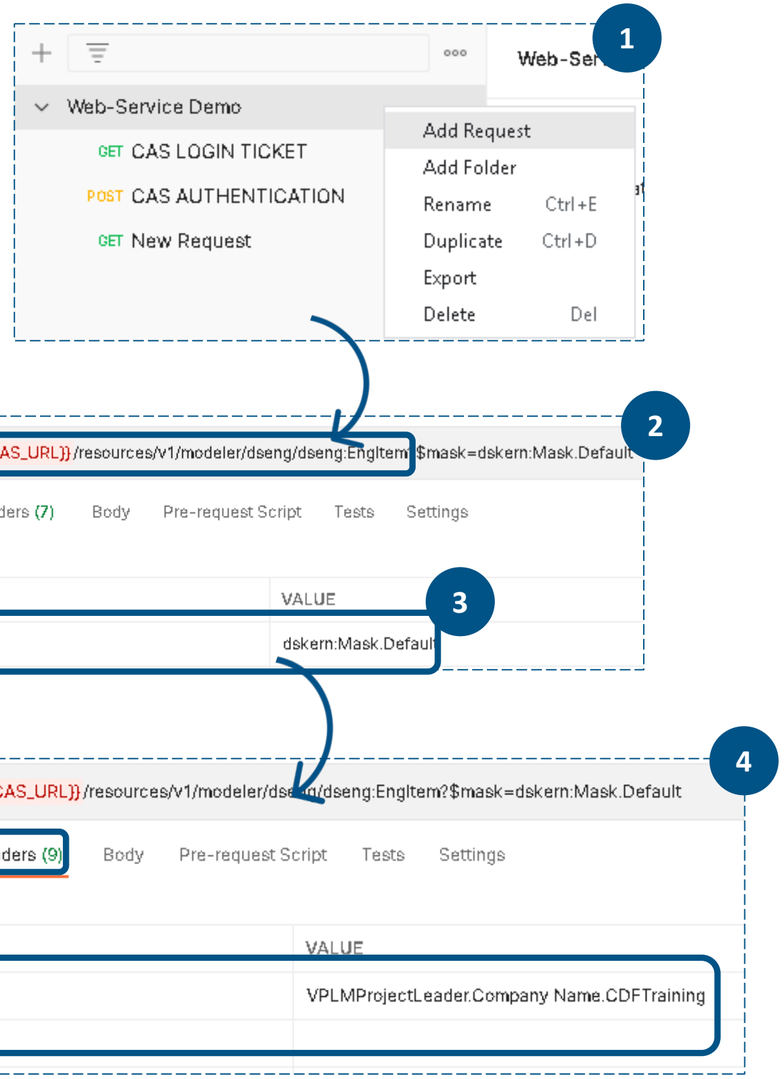
   `{{3DSPACE_CAS_URL}}/resources/v1/modeler/dseng/dseng:EngItem`

3. Under **Params** section, enter Query parameter, as mentioned below.

Key	Value
$mask	One of the allowed values as per Developer Assistance. In this case, value is **dskern:Mask.Default**

4. Under **Headers** section, enter parameters as shown in screenshot.

Key	Value
SecurityContext	Valid security context required to create Engineering Item
ENO_CSRF_TOKEN	*Empty*. We will copy ENO_CSRF_TOKEN from output of *FETCH CSRF TOKEN* web-service, during execution.



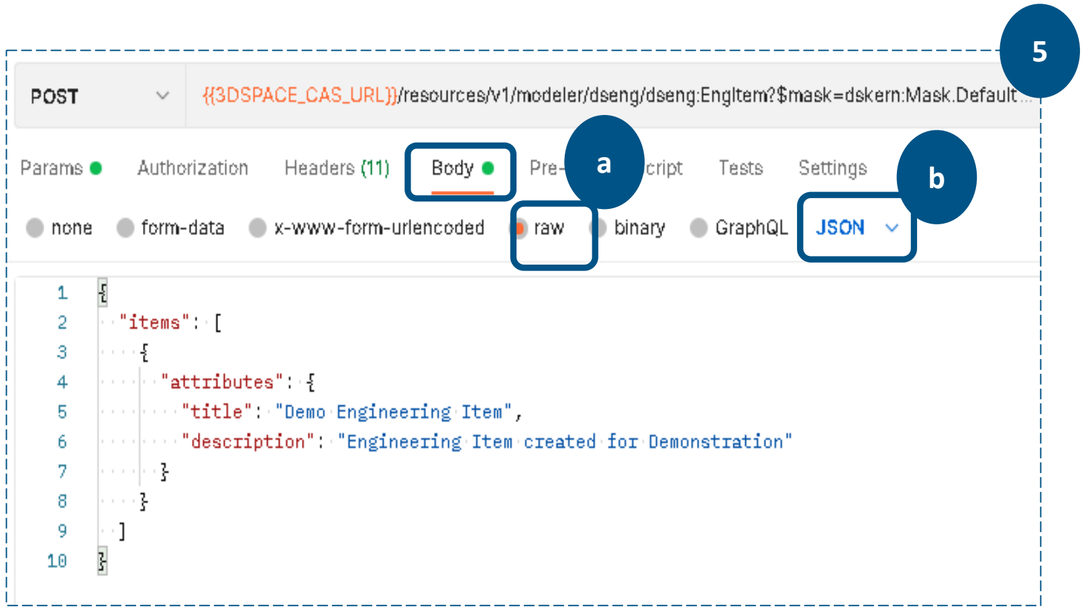
Create Engineering Item – Demonstration – Steps (7/12)

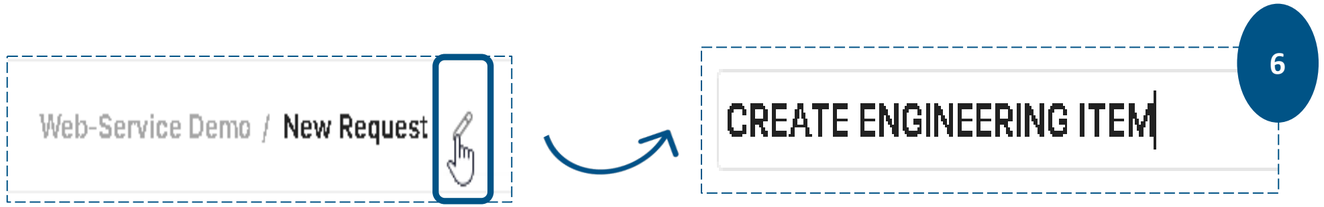
Create Request for creating Engineering Item (continued…):

5. Under **Body** section, enter the request body containing details of the Engineering Item to be created.

    a) Select the data-type as **raw**, and
    b) Select format as **JSON**

6. Rename the request to **CREATE ENGINEERING ITEM**, and **Save** the changes.





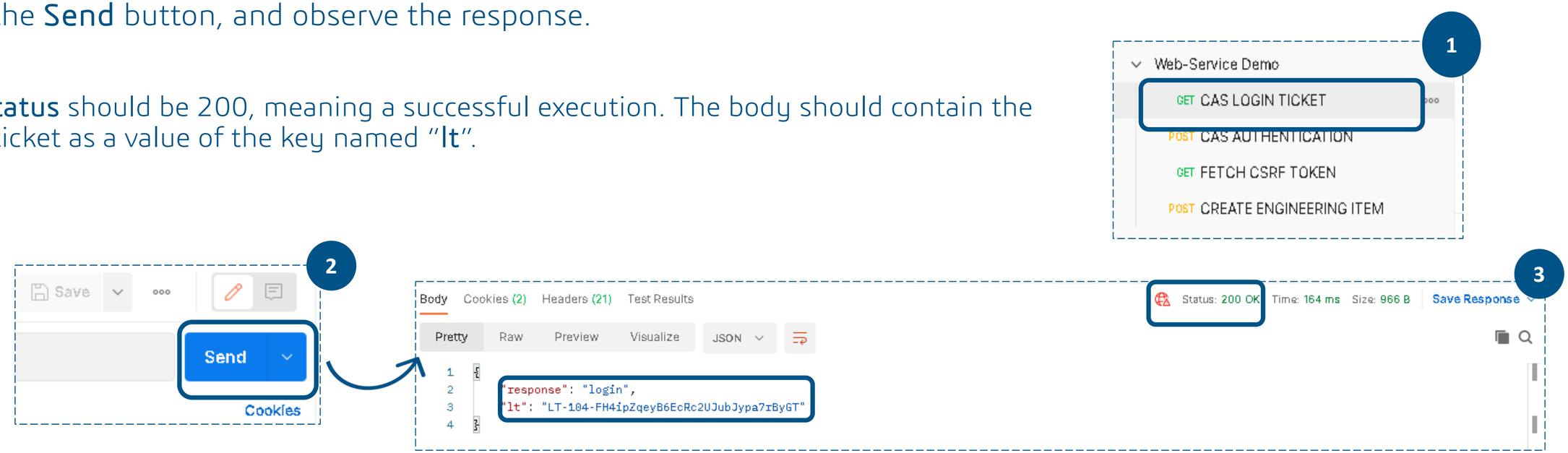
Create Engineering Item – Demonstration – Steps (8/12)

Execute Request for CAS Login Ticket:

From the list of Web-Service requests created in *Collections* tab, open **CAS LOGIN TICKET**.

Click on the **Send** button, and observe the response.

1. The **Status** should be 200, meaning a successful execution. The body should contain the login ticket as a value of the key named "**lt**".

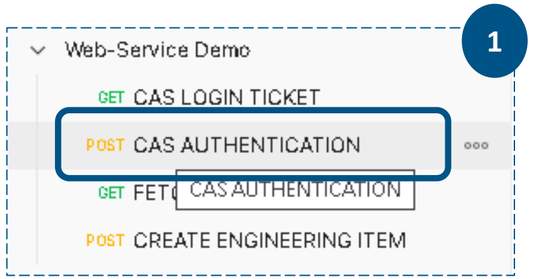


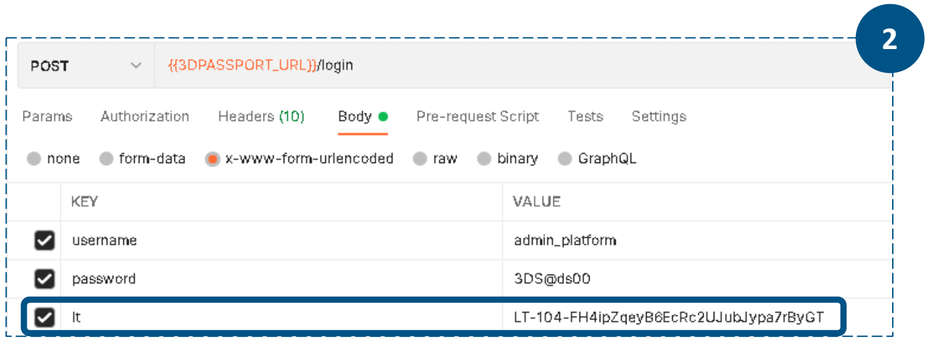
2. Copy the value of the "**lt**" key. This is the login ticket, and would be required for the **CAS Authentication** web-service.

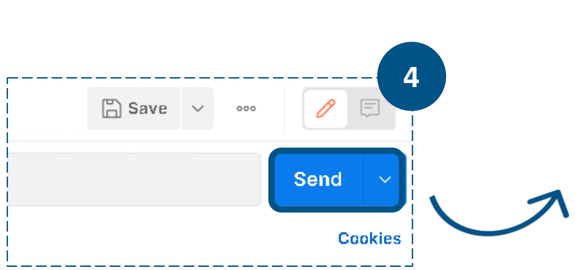
Create Engineering Item – Demonstration – Steps (9/12)

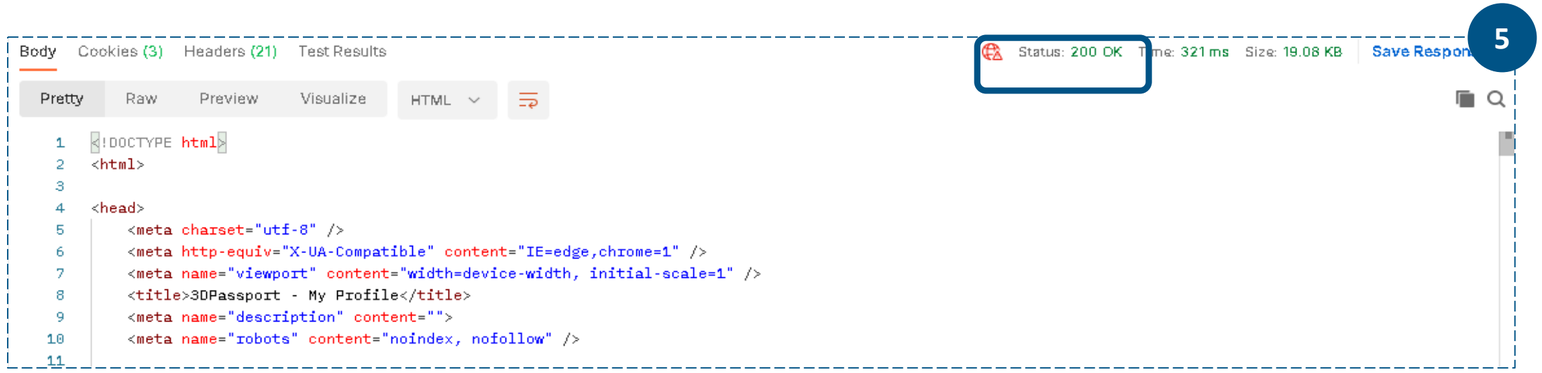
Execute Request for CAS Authentication:

1. From the list of Web-Service requests created in *Collections* tab, open **CAS AUTHENTICATION**.
2. Under **Body** section, paste the *Login Ticket*, as copied from the output of the previously executed **CAS LOGIN TICKET** web-service, against the key named "**lt**".
3. **Save** the changes.
4. Click on the **Send** button, and observe the response.
5. The **Status** should be 200, meaning a successful authentication of Username and Password.





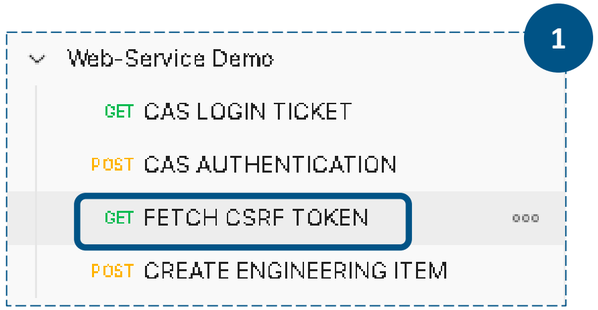




Create Engineering Item – Demonstration – Steps (10/12)

Execute Request for fetching CSRF Token:

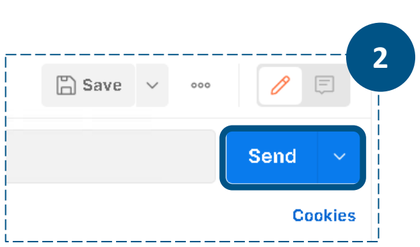
1. From the list of Web-Service requests created in *Collections* tab, open **FETCH CSRF TOKEN**.



2. Click on the **Send** button, and observe the response.

3. The **Status** should be 200, meaning a successful fetch of CSRF token.

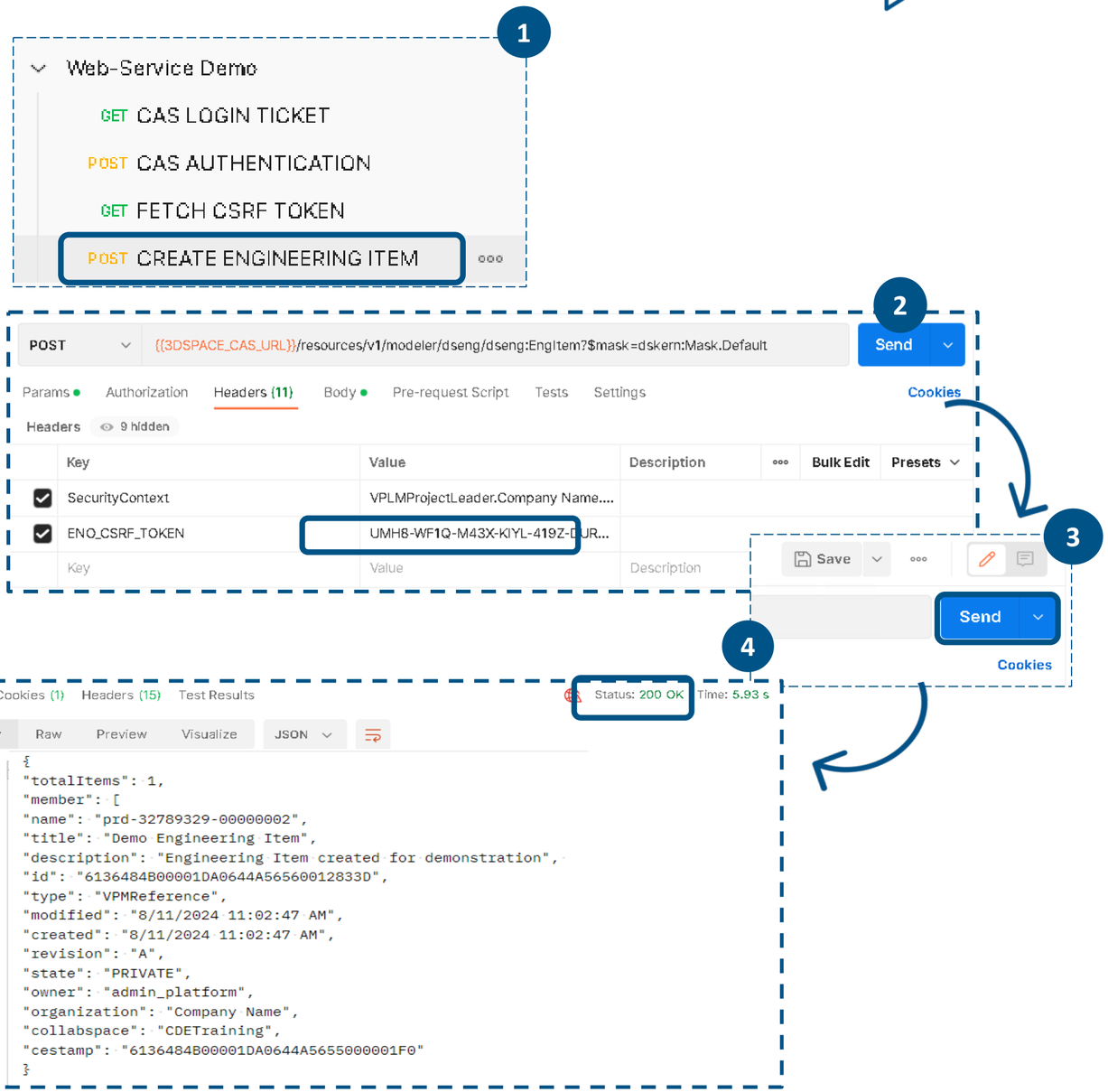
   Copy the **value** of the **ENO_CSRF_TOKEN**, as returned by the web-service. This would be used to create Engineering Item.



Create Engineering Item – Demonstration – Steps (11/12)

Execute Request to Create Engineering Item:

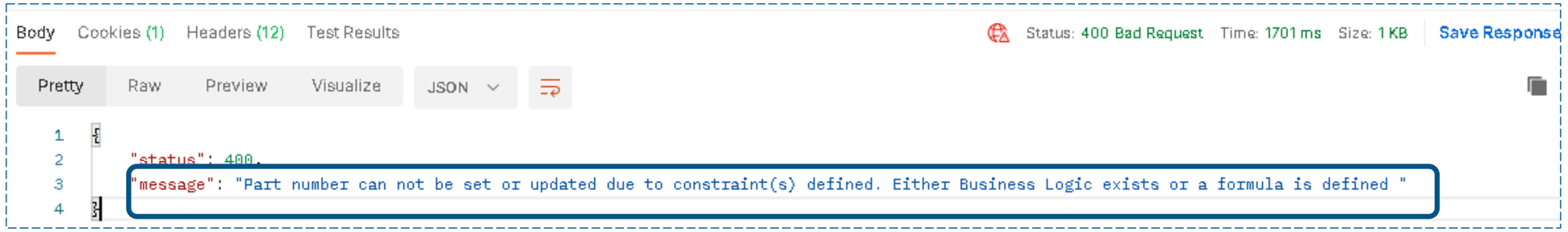
1. From the *Collections* tab, open the **CREATE ENGINEERING ITEM** web-service.
2. Under **Headers** section, paste the **value** of the **ENO_CSRF_TOKEN**, as copied from the output of the previously executed **FETCH CSRF TOKEN** web-service, against the key named **ENO_CSRF_TOKEN**.
3. **Save** the changes. Click on the **Send** button, and observe the response.
4. The **Status** should be **200**, meaning a successful creation of Engineering Item. The web-service should return the details of the created Engineering Item.



Create Engineering Item – Demonstration – Steps (12/12)

Execute Request to Create Engineering Item (continued…):

- Some customer systems may have the **PLMComputePartNumber** Business Rule implemented. This Business Rule automatically assigns Enterprise Item Number to a newly created Engineering Item.
  - If the *PLMComputePartNumber* Business Rule is defined in the customer's tenant, and the customer attempts to create an Engineering Item using web-services while passing the Part Number as a parameter, then the *Business Rule* would cause a conflict.
  - Under above circumstances, while calling the web-services, the below error may be encountered because of the conflict.



  - In this case, the **partNumber** parameter should be removed from the request body of the web-service call. This would allow the Business Rule to set the Enterprise Item Number automatically.

Create Enterprise Reference (Engineering Item Number) – Demonstration – Steps (1/4)

Create Request for creating Enterprise Reference (Engineering Item Number) :

1. Under **Collections** tab, right-click on **Web-Service Demo**, and select **Add Request** option.
2. Select HTTP verb as **POST**, and *Enter request URL*, as mentioned below:

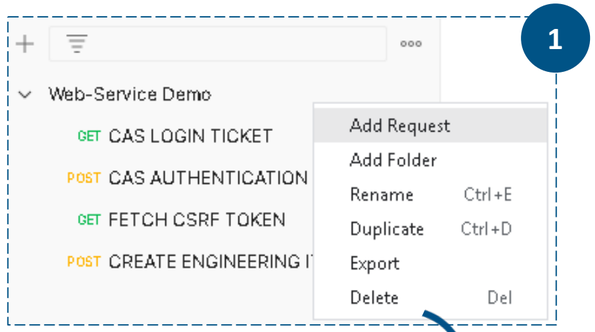
   `{{3DSPACE_CAS_URL}}/resources/v1/modeler/dseng/dseng:EngItem/{ID}/dseng:EnterpriseReference`

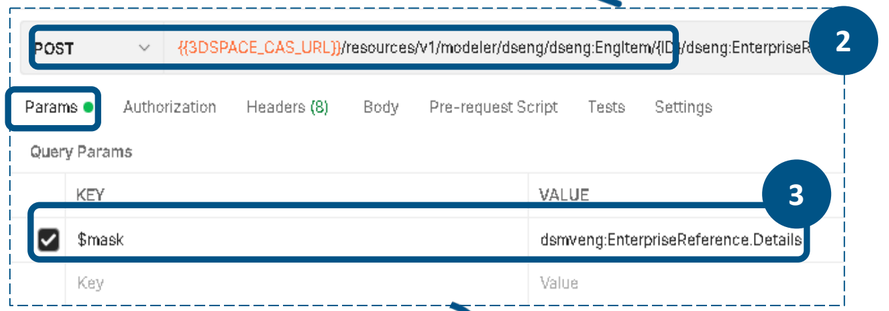
3. Under **Params** section, enter Query parameter, as mentioned below.

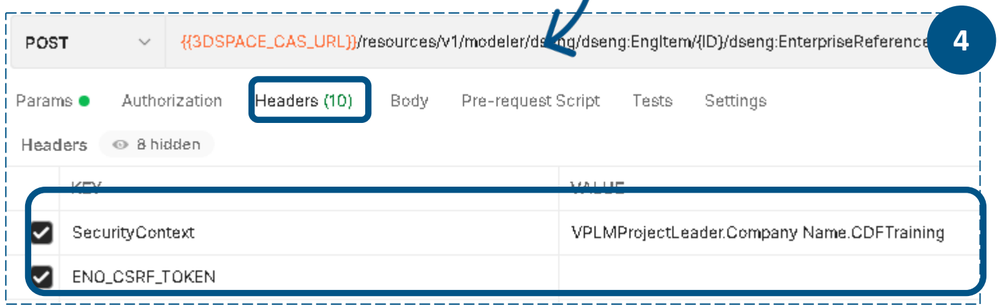
Key	Value
$mask	One of the allowed values as per Developer Assistance. In this case, value is **dsmveng:EnterpriseReference.Details**

4. Under **Headers** section, enter parameters as shown in screenshot.

Key	Value
SecurityContext	Valid security context required to create Engineering Item
ENO_CSRF_TOKEN	*Empty*. We will copy ENO_CSRF_TOKEN from output of *FETCH CSRF TOKEN* web-service, during execution.







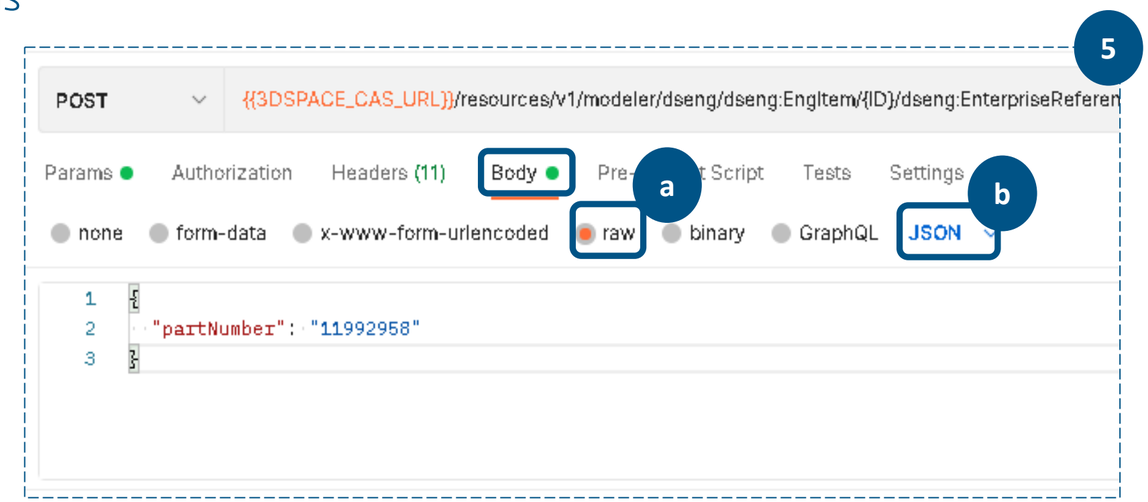
Create Enterprise Reference (Engineering Item Number) – Demonstration – Steps (2/4)

Create Request for creating Enterprise Reference (continued…):

5. Under **Body** section, enter the request body containing details of the Engineering Item to be created.

   a. Select the data-type as **raw**, and
   b. Select format as **JSON**

6. Rename the request to **CREATE ENTERPRISE REFERENCE**, and **Save** the changes.

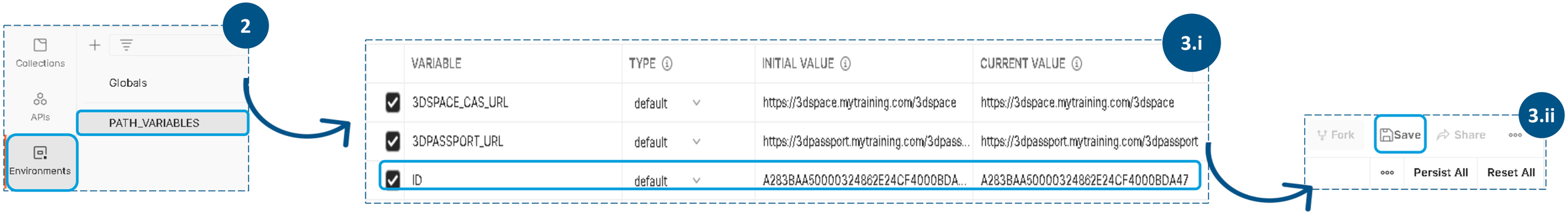


Create Enterprise Reference (Engineering Item Number) – Demonstration – Steps (3/4)

Copy Name of Created Engineering Item to Environment :

1. Analyze the response of the previously executed **CREATE ENGINEERING ITEM** web-service. If the return code was **200**, then the response would contain a key named **id**. Copy the value of this key from the response.

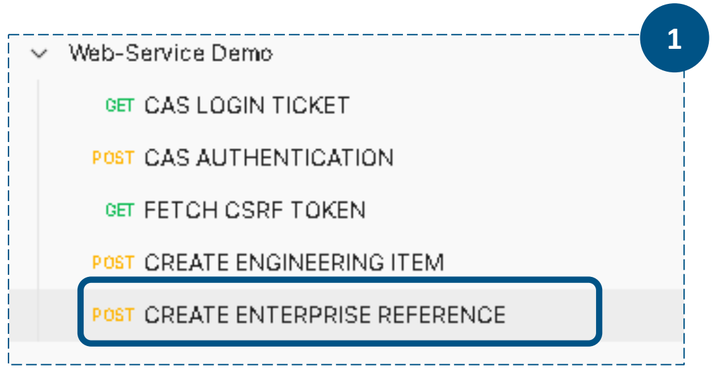
2. Go to **Environments** tab, and select previously created variable set **PATH_VARIABLES**.
3. Set both **INITIAL VALUE** and **CURRENT VALUE** of **ID** variable with the **id** as copied in step 1. **Save** the changes.

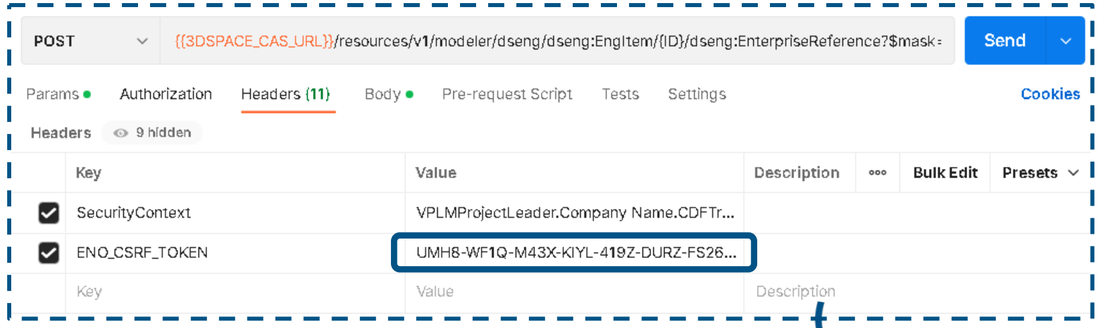


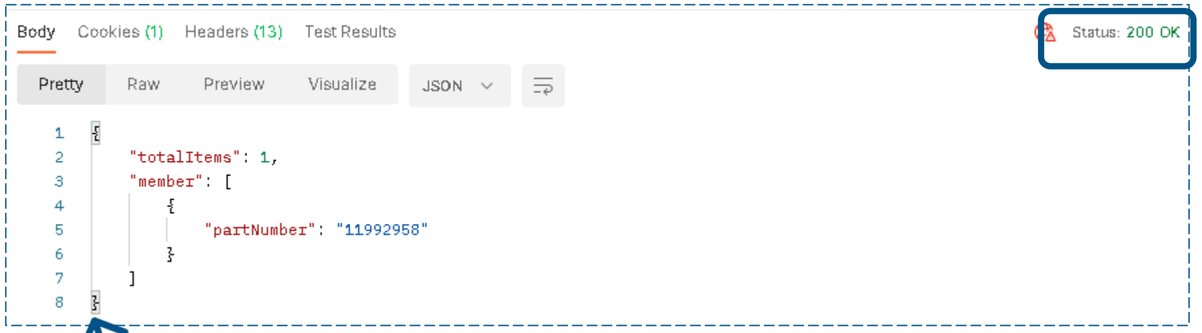
Create Enterprise Reference (Engineering Item Number) – Demonstration – Steps (4/4)

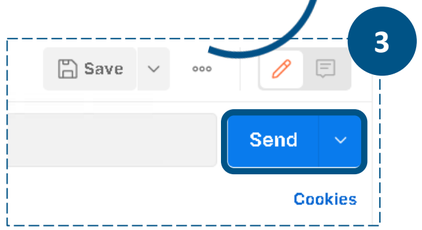
Execute Request to create Enterprise Reference :

1. From the *Collections* tab, open the **CREATE ENTERPRISE REFERENCE** web-service.
2. Under **Headers** section, paste the **value** of the **ENO_CSRF_TOKEN**, as copied from the output of the previously executed **FETCH CSRF TOKEN** web-service, against the key named **ENO_CSRF_TOKEN**.
3. **Save** the changes. Click on the **Send** button, and observe the response.
4. The **Status** should be **200**, meaning a successful creation of Enterprise Reference. The web-service should return the details of the created Enterprise Reference.





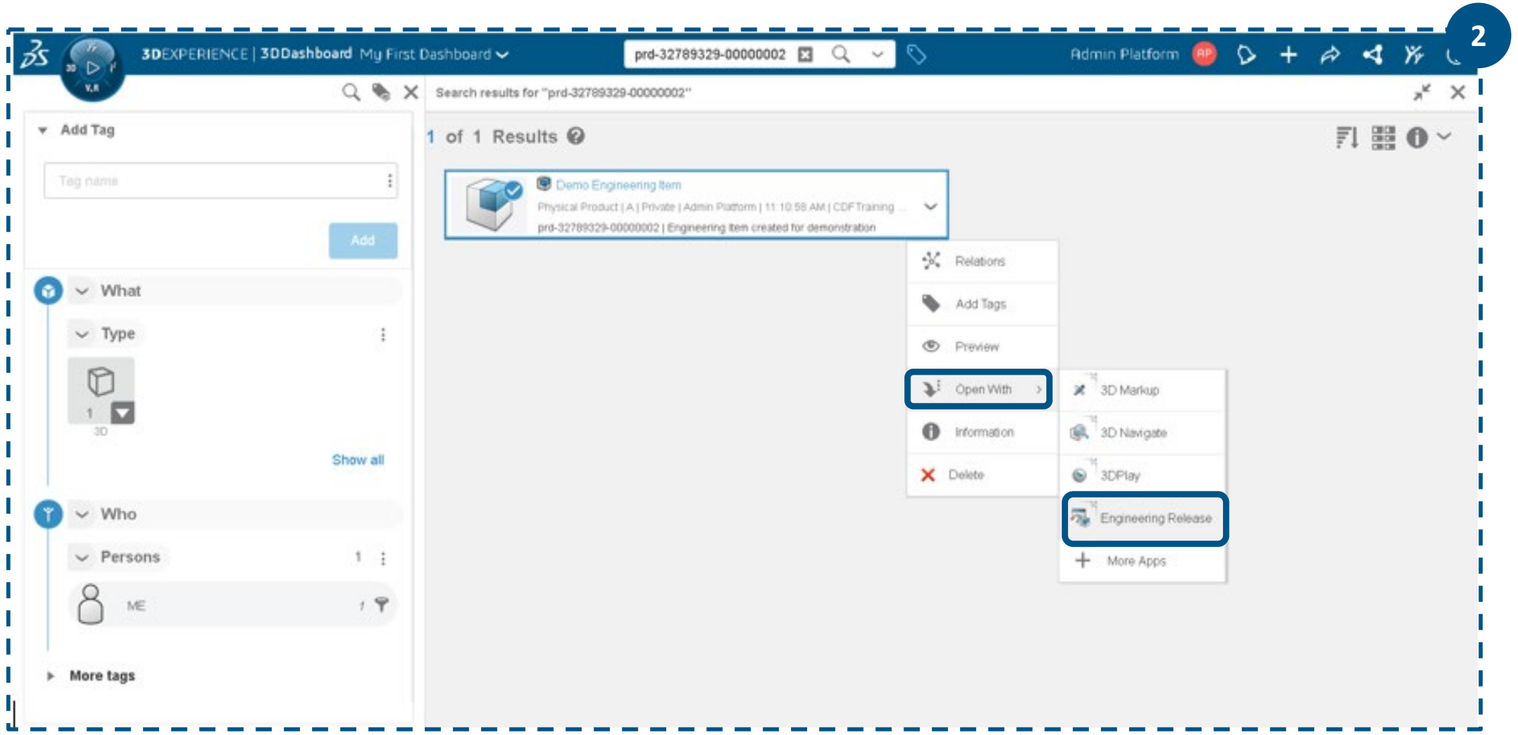




Verify Engineering Item and Enterprise Reference (*Engineering Item Number*) (1/2)

Verify the creation of Engineering Item and Enterprise Reference(*Engineering Item Number*) using the Engineering Release 3DDashboard App

1. Log in to **3D**EXPERIENCE platform with the same User whose context was used to create the Engineering Item using Web-Services.
2. Search for the created *Engineering Item*, using the name as returned by the last web-service.
   a. Open the object using *Engineering Release* app.

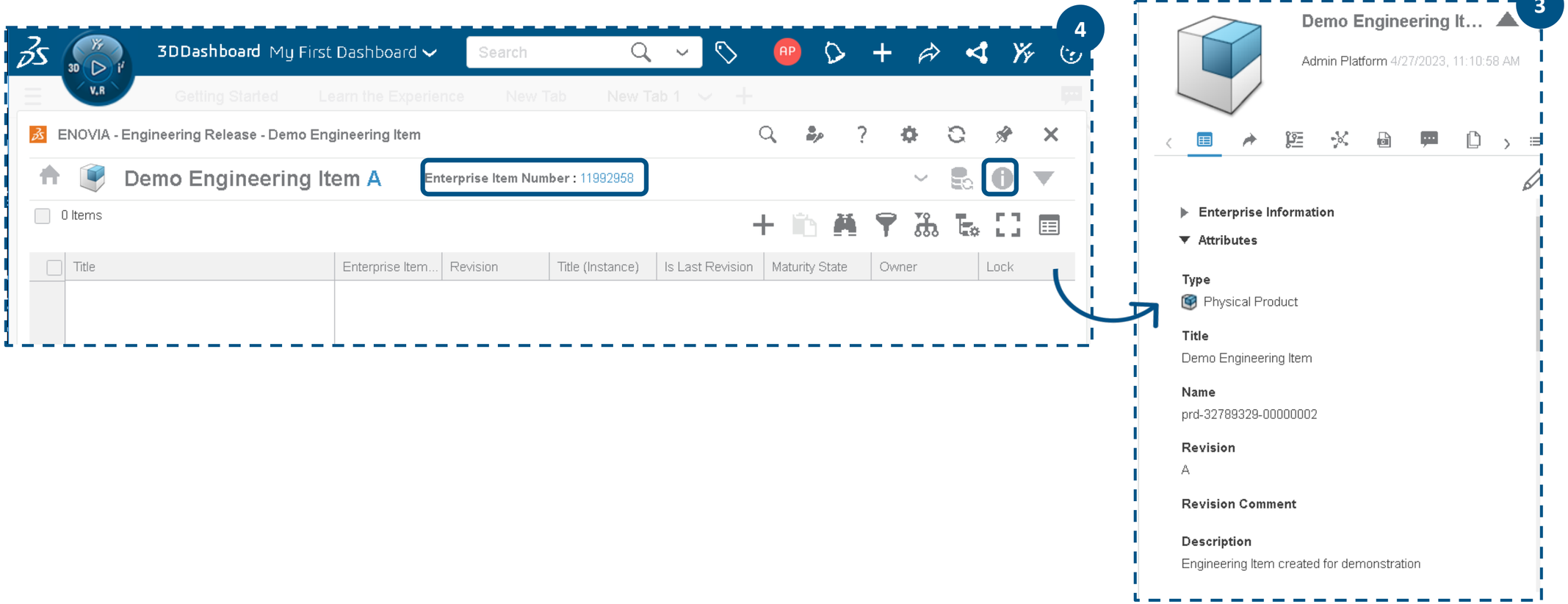


To be able to search the Engineering Item object, it needs to be indexed first. Please allow sufficient time for indexation to happen, before searching for object.

Verify Engineering Item and Enterprise Reference (*Engineering Item Number*) (2/2)

Verify the creation of Engineering Item and Enterprise Reference(*Engineering Item Number*) using the Engineering Release 3DDashboard App (continued...)

3. After the object loads in *Engineering Release* app, click on **Information** icon to see more details about the Engineering Item.
4. Also verify the **Enterprise Reference(*Engineering Item Number*)** is updated on newly created Engineering Item.



Create Engineering Item – Automation

- In the previous example, we saw how to call OOTB web-services manually, using POSTMAN. However, in a realistic scenario, the sequential web-service calls would need to automated.

- The automation of web-service calls and their chaining is achieved through scripts. The script shown below is a portion of a larger program, built using .NET/C#, and it serves to demonstrate an automated web-service call.

```
public async static Task InitTest(TestContext testContext)
        {
            platformSettings = PlatformSettingsProvider.CreateSettings();
            credential = PlatformSettingsProvider.GetCredentials();
            authenticatedClient = new PassportAuthenticationService(platformSettings).CreateAuthenticatedClient(credential);
            engineeringServices = new EngineeringServices(authenticatedClient, platformSettings);

            //Use anonymous class
            var request = new {
                items = new[]{
                    new {
                        attributes = new {
                            title=  "parent",
                            description = "parentDescription"
                        }
                    },
                       new {
                        attributes = new {
                            title=  "child1",
                            description = "child1Description"
                        }
                    },
                          new {
                        attributes = new {
                            title=  "child2",
                            description = "child2Description"
                        }
                    }
                }
            };

            var json = JsonDocument.Parse(JsonSerializer.Serialize(request));
            var uri = new Uri($@"/resources/v1/modeler/dseng/dseng:EngItem", UriKind.Relative);
            var responseCreate = await engineeringServices.SendAsync(HttpMethod.Post, uri, json.RootElement);

            var resultsArray = responseCreate.GetProperty("member").EnumerateArray().ToList();
```

This code is just for reference. Actual implementations may differ, based on requirements.

Exercise: Fetch Engineering Item using Web-Service

1. In this exercise, we shall demonstrate the how to fetch an *Engineering Item using* Web-Services.

2. The following are the features of this exercise:

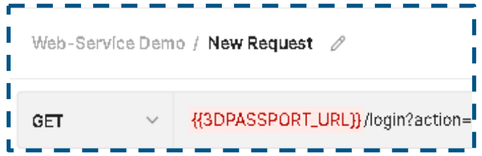
    a. We shall use POSTMAN application for this demonstration
    b. The web-services would be called based on information obtained from Developer Assistance documentation.

Fetch Engineering Item (1/2)

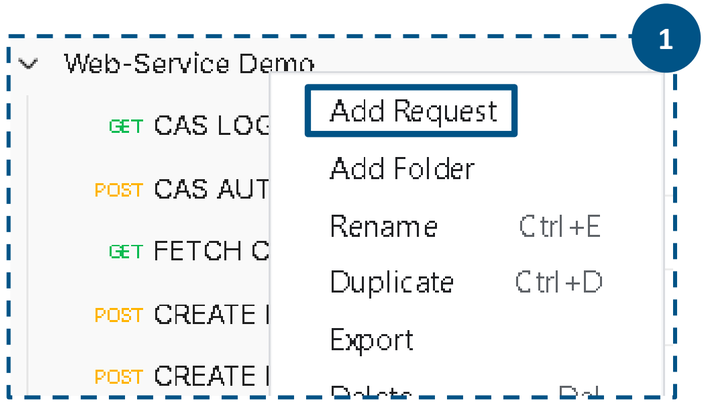
1. Under Collections tab, right-click on Web-Service Demo, and select Add Request option.
2. Select the HTTP verb as **GET**, and *Enter request URL*, as mentioned below:
   {{3DSPACE}}/resources/v1/modeler/dseng/dseng:EngItem/search
3. Under **Params** section, enter Query parameter, as mentioned below.

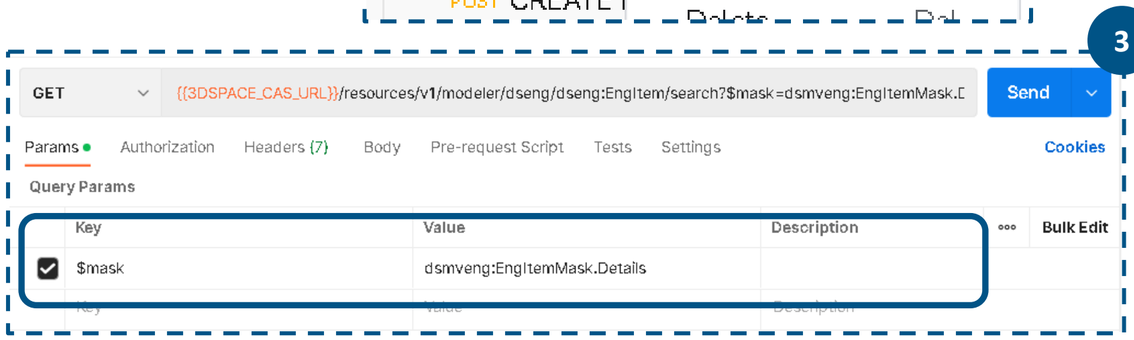
Key	Value
$mask	One of the allowed values as per Developer Assistance. In this case, value is dsmveng:EngItemMask.Details

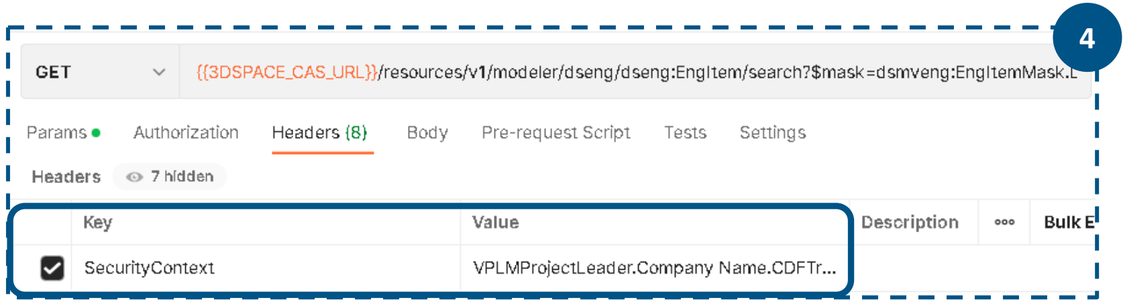
4. Under **Headers** section, enter parameter, as shown in screenshot.
5. Rename the request to **FETCH ENGINEERING ITEM**, as shown below, and **Save** the changes.



FETCH ENGINEERING ITEM

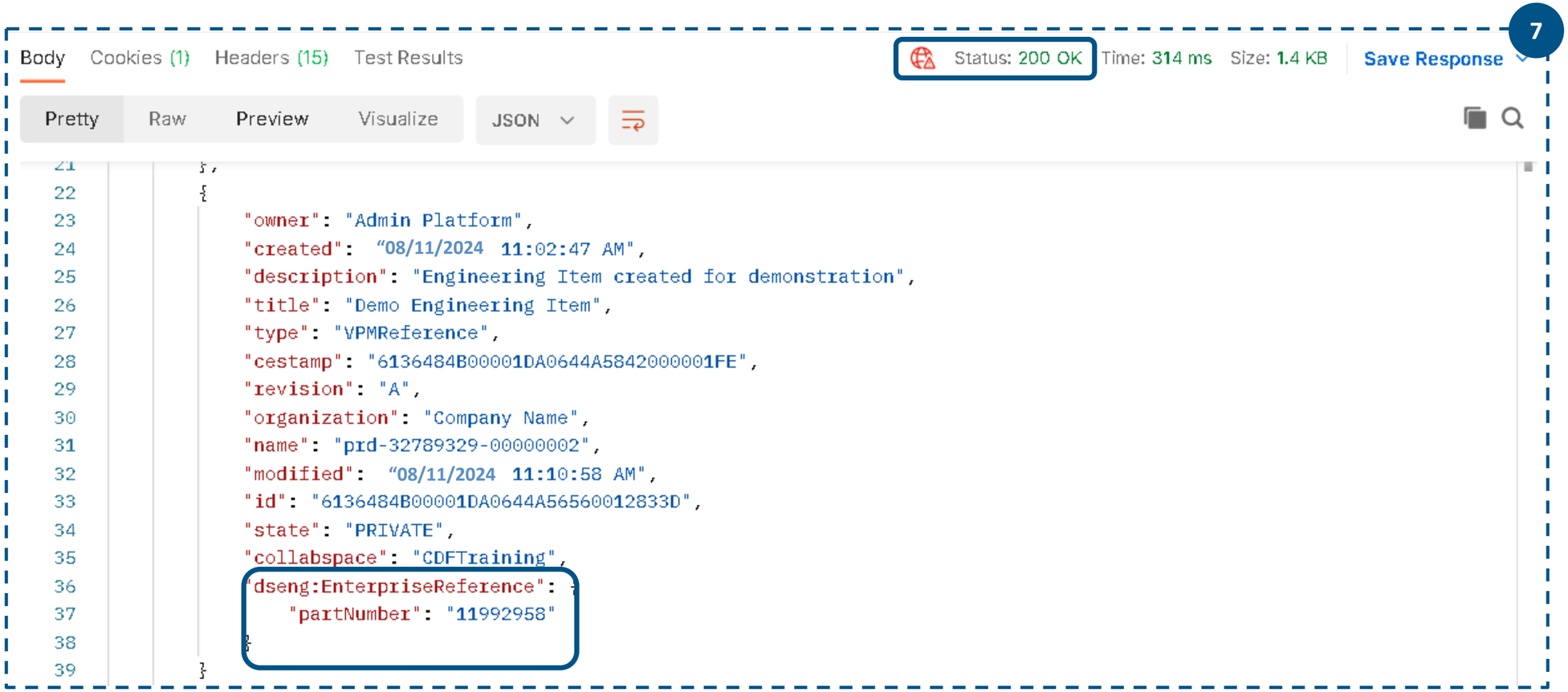






Fetch Engineering Item (2/2)

6. Click on the Send button, and observe the response.
7. The Status should be 200, meaning getting the objects successfully. The web-service should return the details of the created Enterprise Reference.



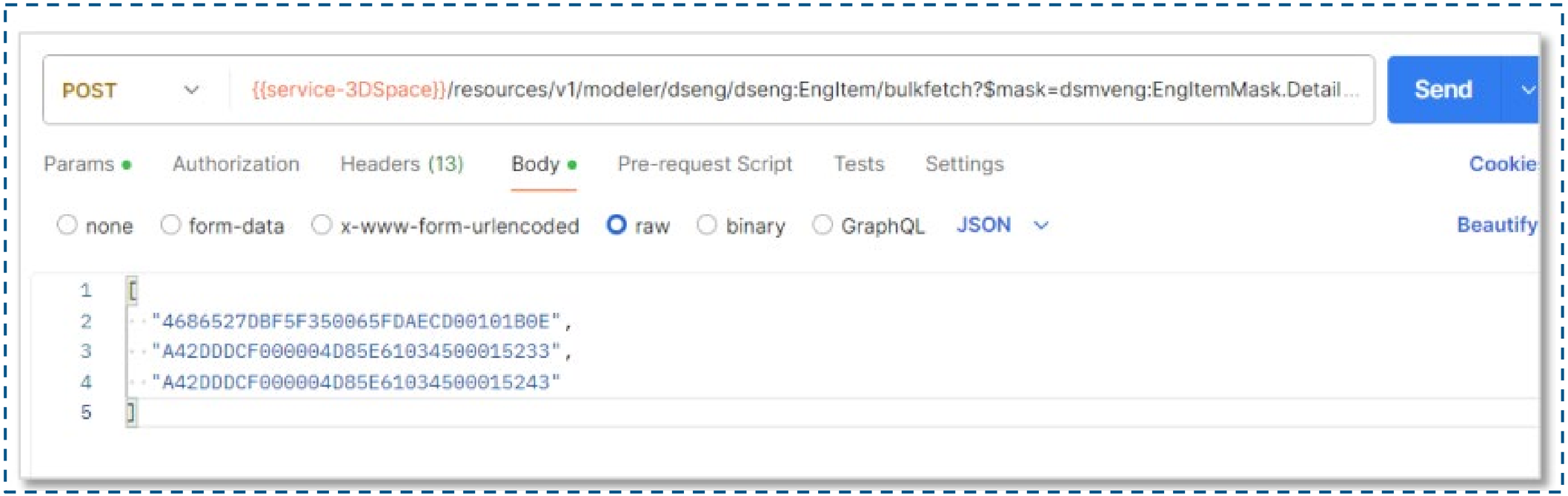
Exercise: Demonstration of BulkFetch

In this exercise we will Fetch the items in bulk.

30 minutes

Demonstration of bulk fetch

1. We can get multiple Engineering items which are indexed.
2. To do so, use the POST method.
3. Put multiple physical ids in the payload array, which you want to fetch.



Exercise: Calling Web-Service from Widget

Calling a 3DSpace Web-Service from a Widget in 3DDashboard

30 minutes

Call the Web Service in the Widget (1/3)

1. We shall use the following template: <**Widget_Root**>\SampleWidgetWAFData\WidgetWAFData.html
2. We shall use the library DS/WAFData/WAFData
3. Note the we added a new widget event :

```
widget.addEvent("onRefresh", myWidget.onLoadWidget);
```

4. Call the WebService to gather and display 3DSpace data :

```
var urlWAF = widget.getValue("urlREST");
var dataWAF={
   "type": widget.getValue('typeObj'),
   "selects": "attribute[*],current,name,revsion"
};
var headerWAF={
   "SecurityContext": widget.getValue("ctx")
};
var methodWAF="GET";
…
```

Call the Web Service in the Widget (2/3)

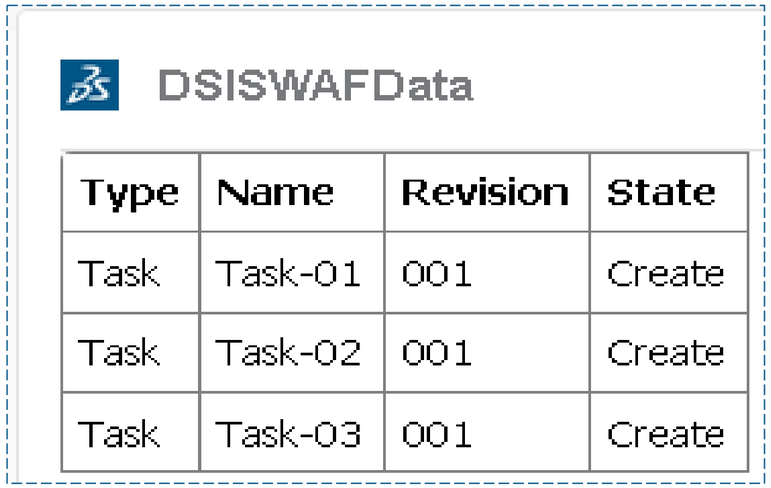
5. Call the Web-Service to gather and display 3DSpace data :

```
WAFData.authenticatedRequest(urlWAF, {
    method: methodWAF,
    data: dataWAF,
    headers: headerWAF,
    type: 'json',
    onComplete: function (dataResp) {
        if(dataResp.msg==="OK"){
            myWidget.dataFull=JSON.parse(dataResp.data);
            myWidget.displayData(myWidget.dataFull);
        }else{
            widget.body.innerHTML += "<p>Error in WebService Response</p>";
            widget.body.innerHTML += "<p>"+JSON.stringify(dataResp)+"</p>";
        }
    },
    onFailure: function(error){
        widget.body.innerHTML += "<p>Call Failure</p>";
        widget.body.innerHTML += "<p>"+JSON.stringify(error)+"</p>";
    }
});
```

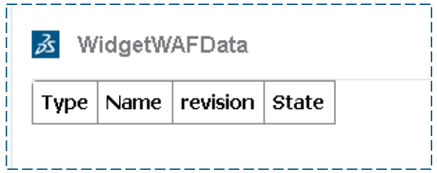
Call the Web Service in the Widget (3/3)

6. Save the scripts.

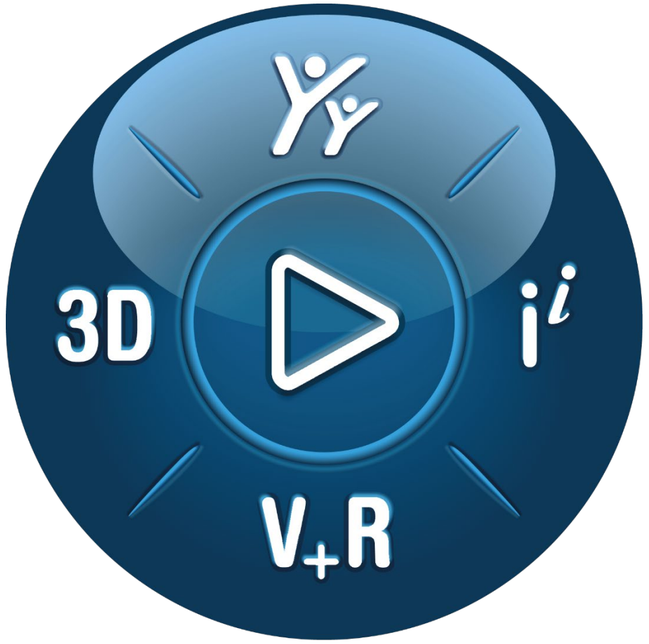
7. Create an Additional App as mentioned previously to see the data as fetched by the web-service.



Please note the following:
By default the widget displays objects of type Task. The type may be changed by modifying Widget preferences. Some business objects that are accessible by the current user context, need to be already present for the widget to display data. If no such data is found, a blank page with just the table headers are displayed, as shown:



Enterprise Integration Framework



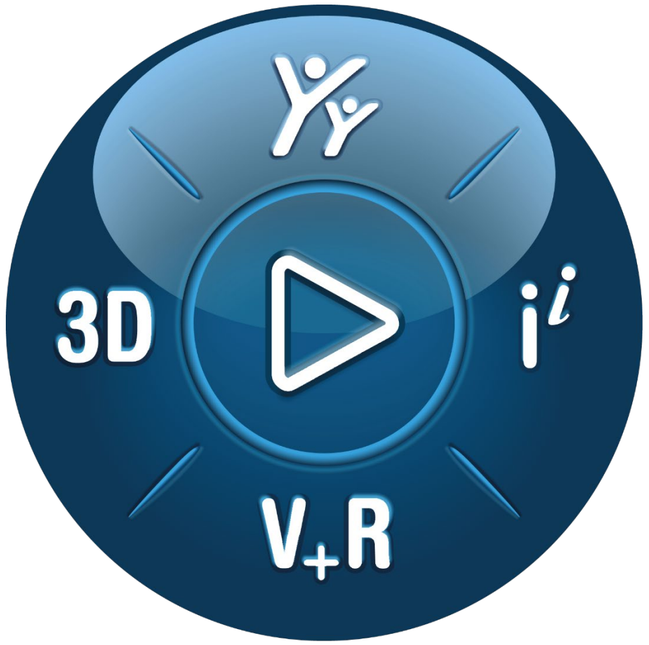
3DEXPERIENCE®

In this lesson we are going to know about the following apps and their associated roles:

- Agent Management
- Event Publishing
- Enterprise IP Integration Management

1 Hours

Enterprise Integration Framework



3DEXPERIENCE®

In this lesson we shall know about the integration of 3DEXPERIENCE platform On-Cloud with other Enterprise Systems.

Here is the topic to be covered:

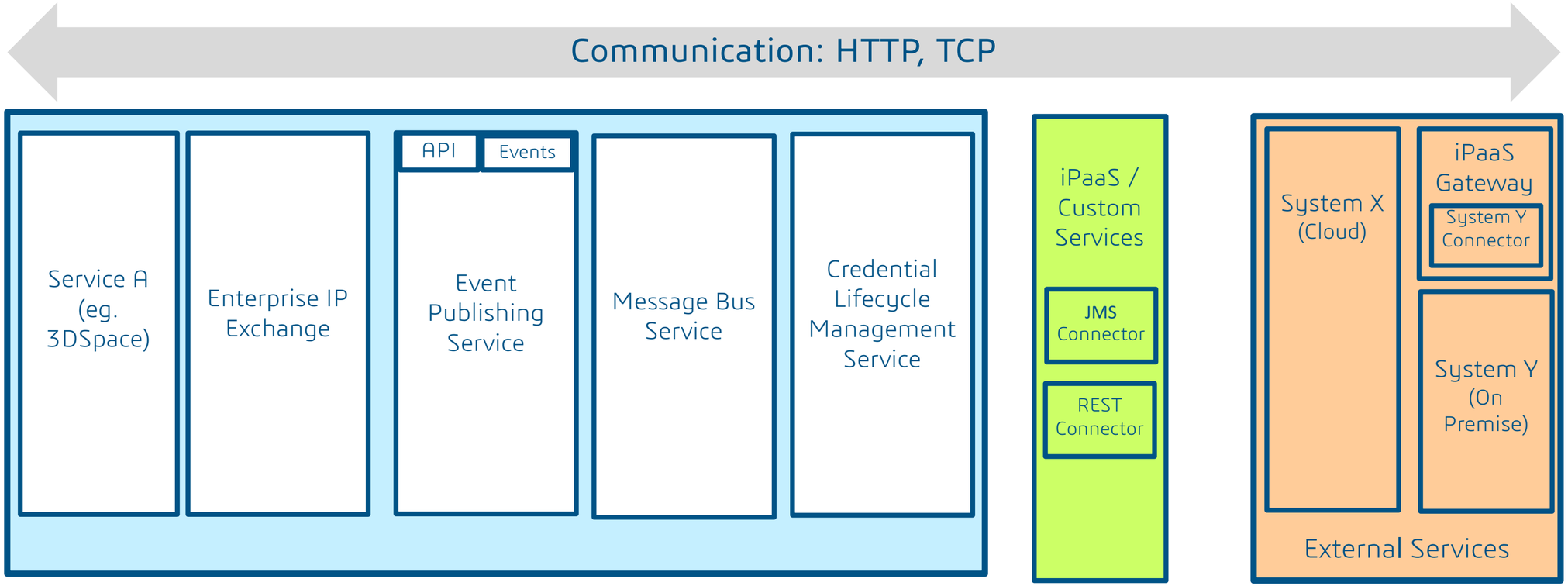
- Enterprise Integration Framework

Enterprise Integration Framework – Introduction

- **Enterprise Integration Framework** was introduced for allowing seamless integration of **3D**EXPERIENCE platform on-cloud with customer's Enterprise solutions, such as ERP systems.

- *Enterprise Integration Framework* also allows coexistence of **3D**EXPERIENCE platform with other legacy PDM systems.

- With *Enterprise Integration Framework*, data exchange or data synchronization happens based on events in the **3D**EXPERIENCE platform. Communication happens over HTTPS and TCP protocols



Enterprise Integration Framework – Architecture Overview (1/2)

- The Enterprise Integration Framework consists of the following services:

  - **REST Web-Services**: REST Web-Services are used for actual data exchange between applications across the cloud.

  - **Automation Services**: This includes the following:
    - **Credential Lifecycle Management (CLM) Services**: REST Web-Services have been used for quite some time to interact with **3D**EXPERIENCE platform. Until recently, user authentication for integrations using REST Web-Services were a bit tedious.

      Instead of using traditional tools for Authentication, On-Cloud solutions can make use of *Credential Lifecycle Management* services, which allow us to create **Integration Agents**. *Integration Agents* are Users that **proxy** on behalf of actual platform users.

    - Integration Agents can be used to authenticate integrations without having to expose the actual login credentials (username and password) of the platform user.

    - As many Integration Agents can be created as required, and they can also be managed.

    - **Message Bus Services**: This service provides event messages, which is based on *Active MQ*. Whenever an event (like creation of a VPMReference object, etc.) happens, the system could be configured to send out messages. This activity of sending out messages is taken care of by the Message Bus Services.

      - *Message Bus Services* use *Integration Agents* created in CLM to control authentication and subscription to messages from the platform.

    - **Event Publishing Services:** This service handles the allowed events for the **3D**EXPERIENCE platform, based on which, the messages may be sent out. The events are controlled by Platform Administrator, who can chose to turn on or off an event, as required. This services also ensures that events are published only after complete indexation.

Enterprise Integration Framework – Architecture Overview (2/2)

- **Engineering Exchange Services:** Involves use of **Enterprise IP Exchange** app, for asynchronous bulk export of data from **3D**EXPERIENCE platform. Data exchange happens in Standards Compliant formats.
  - This service only supports Export of data from **3D**EXPERIENCE platform at the moment.

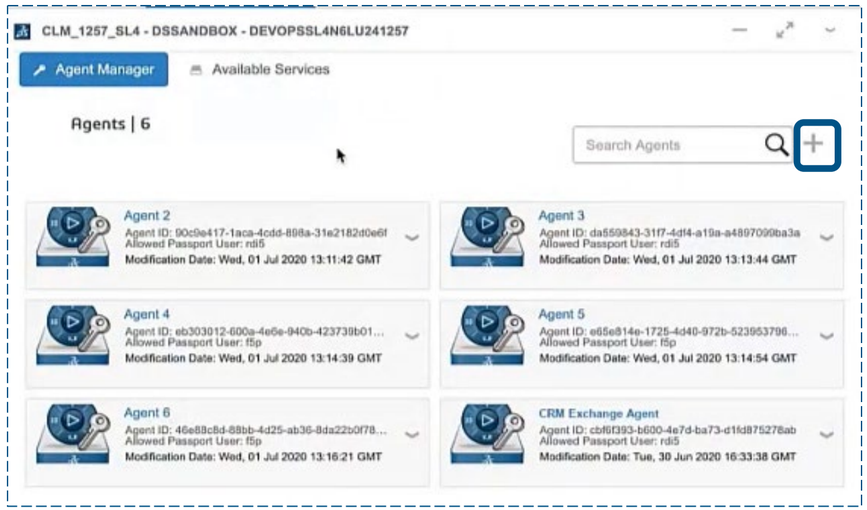
- **iPaaS / Customer Services**: iPaaS (Integration Platform as a Service) is a Service that acts as a medium to standardize data-exchange across enterprise solutions in an organization. iPaaS is recommended but it is not mandatory.
  - Such services are external to the **3D**EXPERIENCE platform.
  - There are many standard services, are available in the market, which could be used to implement *iPaaS*. For example, Apache nifi, Apache Camel, etc. Such services are properly tried and tested. Otherwise, the customer could develop their own solution as well.

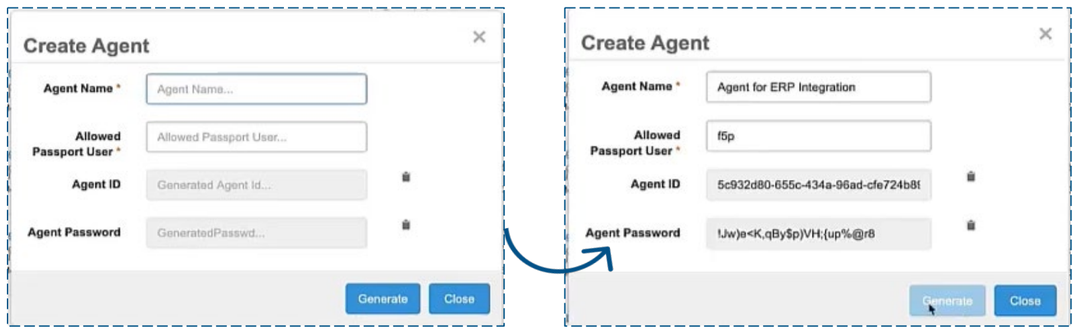
  - These services provides JMS connector and REST connector to achieve integration between other enterprise applications and **3D**EXPERIENCE platform.

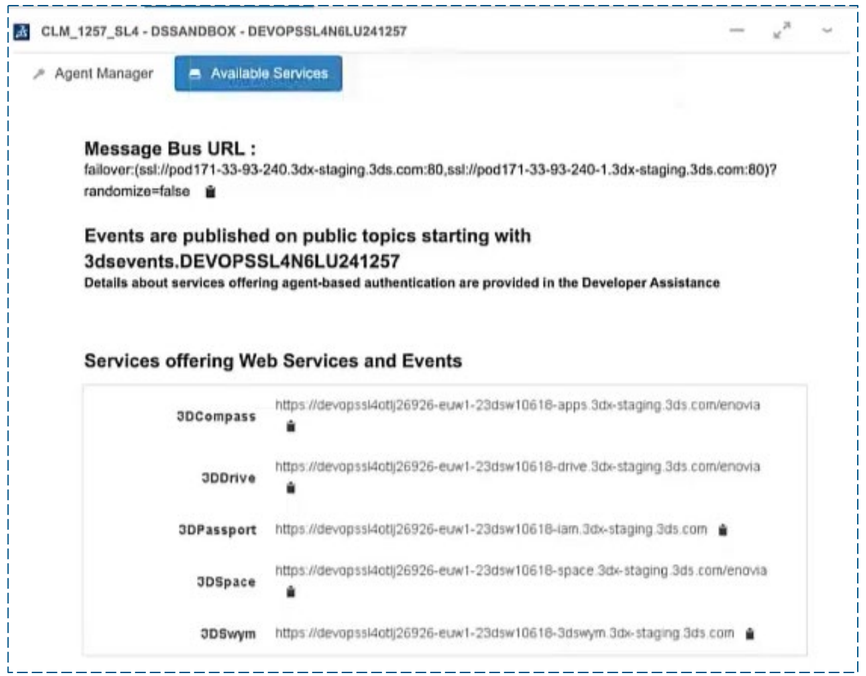
Enterprise Integration Architect Role (1/2)

- The **Platform Manager** can be assigned the **Enterprise Integration Architect** role (**PFI** license). This role grants access to the following two apps:

- **Agent Management App**: This app allows *Platform Administrator* to work with *Credential Lifecycle Management* Services to create Integration Agents. A new *Integration Agent* can be created by providing a name for the agent, and the username of an existing Passport user. The app can also be used to see the URLs of existing platform services







Enterprise Integration Architect Role (2/2) SCREENSHOT CHANGE

- **Event Publishing App**: This app allows *Platform Administrator* to turn on or off publishing of **3D**EXPERIENCE platform events

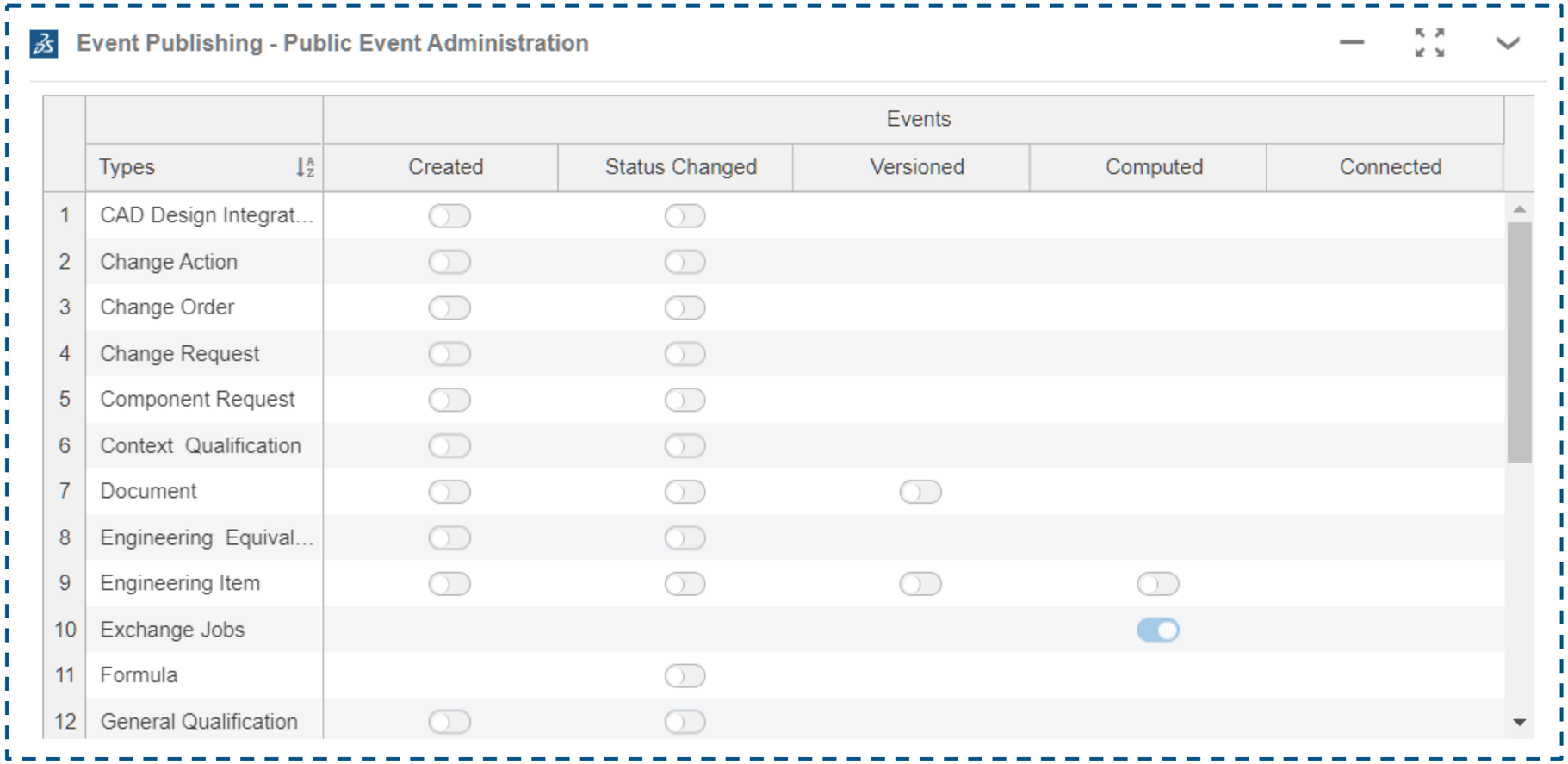
Event Publishing - Public Event Administration

		Events				
	Types	Created	Status Changed	Versioned	Computed	Connected
1	CAD Design Integrat...	○	○			
2	Change Action	○	○			
3	Change Order	○	○			
4	Change Request	○	○			
5	Component Request	○	○			
6	Context Qualification	○	○			
7	Document	○	○	○		
8	Engineering Equival...	○	○			
9	Engineering Item	○	○	○	○	
10	Exchange Jobs				●	
11	Formula		○			
12	General Qualification	○	○			

(3DEXPERIENCE R2024x FD01)

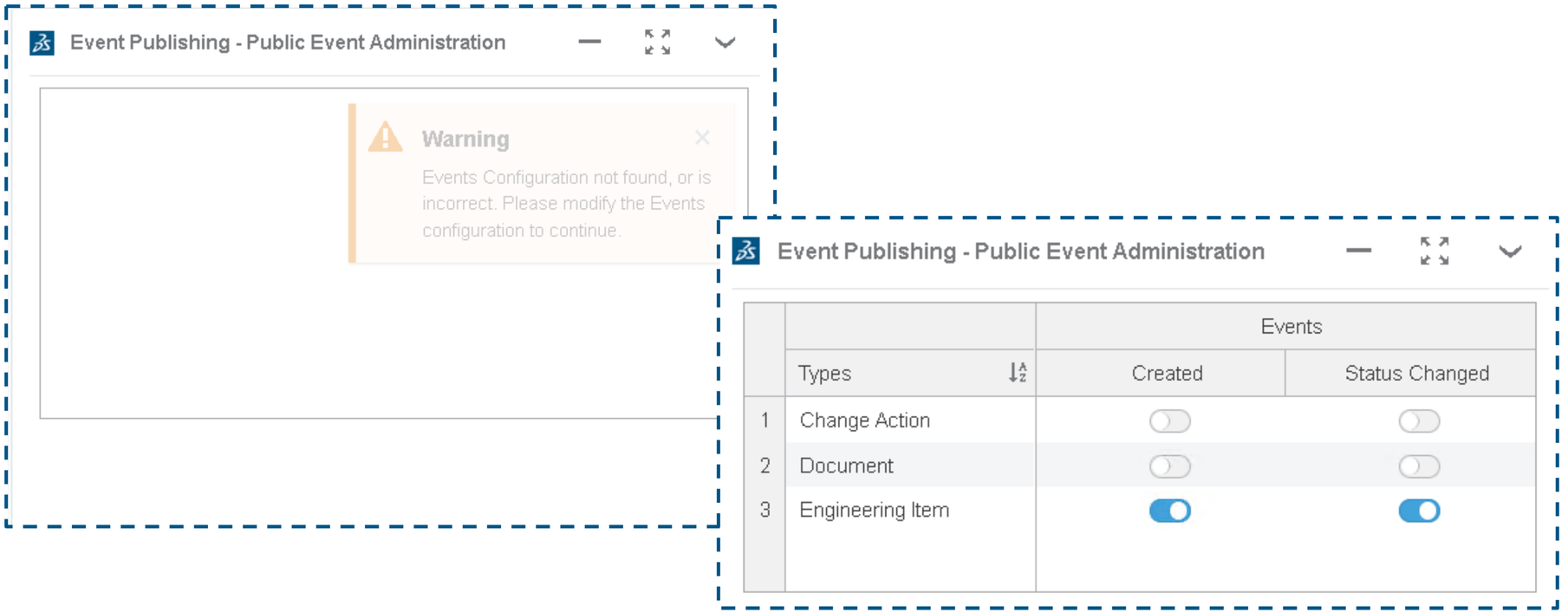
Event Publishing App – On Cloud

- For on-Cloud the events are pre-configured.
- From Event Publishing App you can enable the events for listed types.



Event Publishing App – On-premise

- For on-premise the events are not pre-configured.
- We have to Configure Event Mapping in the to view the events in Event Publishing app.
- **EventPublishDictionary.xml** page object is used to configure event type mapping.
- Event Publishing app uses **EventPublishDictionary** page object to view the events



Enterprise IP Integration Roles (1/2)

- An *IP Exchange Administrator* need to be assigned the following roles:

  - **Enterprise IP Integration Manager (XXH) -** This is a named user license. This Role is used to request for an IP exchange with other Enterprise apps. This user (who has *XXH* license) is used to create an *Integration Agent* using the *Agent Management* app, which is then subsequently used to request for the IP exchange.

    Also, this role allows access to **Enterprise IP Integration Management** app to monitor the IP Exchange jobs.

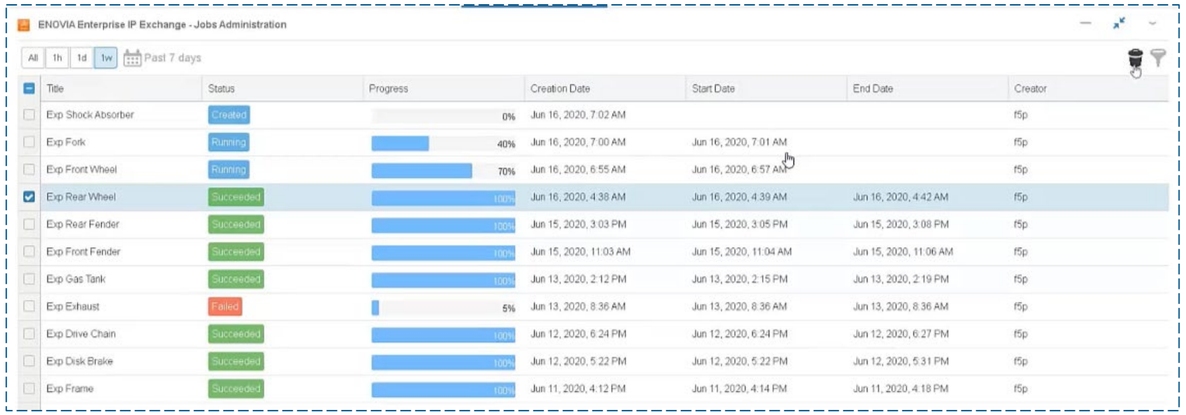
  - **Enterprise IP Integration Pack (XXC) -** This is a volume or credit based license. Each license comes with 10000 credits, and each credit can export up to 10 items. So overall, with the *XXC* license, user can export up to 100000 items with one pack. More packs can be purchased, as and when required. These credits are to be consumed during bulk export only.

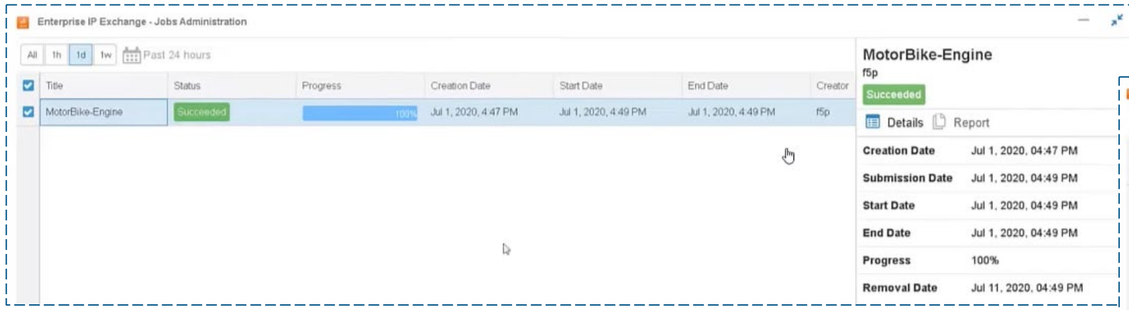
Enterprise IP Integration roles were originally introduced as a substitute for XPDM export/import functionality for On-Cloud implementations.

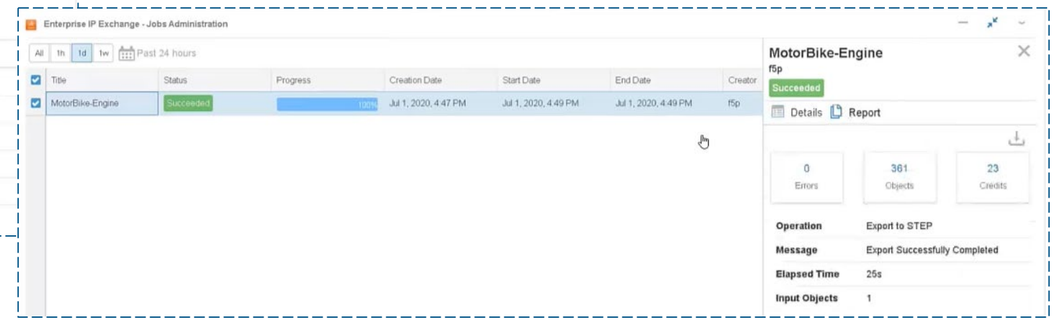
Enterprise IP Integration Roles (2/2)

- The **Enterprise IP Integration Management** app allows the *IP Exchange Administrator* to monitor IP exchange jobs, and generate reports after their execution.

- This app is accessible through the **PAU** license.







Enterprise Integration Framework – Logical Sequence of Steps

- In order to use the *Enterprise Integration Framework*, there is a sequence of steps that needs to be followed, which are as below:

  - **Step 1**: In this step, the *Platform Manager* creates the *Integration Agent* using the *Credential Lifecycle Management* Services (Agent Management App)

  - **Step 2**: Using the above created Integration Agent, we subscribe to a Tenant topic, using the *Message Bus Services*. There are two kinds of message topics:
    - User topics, which pertain to 3DSpace user events, like promote/demote, create, delete, etc.
    - Admin topics, which pertain to events like Enterprise IP Exchange job completion

    In Step 2, we can create subscription agents to listen to the above topics. Also, custom code could be developed for the subscription agents to respond to above topics

  - **Step 3**: Based on the topics that are subscribed to in *Step 2*,
    - REST Web-Services are to be used to perform further actions. For example, upon release (event) of an Engineering Item, a report could be generated.
    - Using Integration Agent created in *Step 1*, IP Exchange jobs can be created.

Congratulations!
You have completed this training.