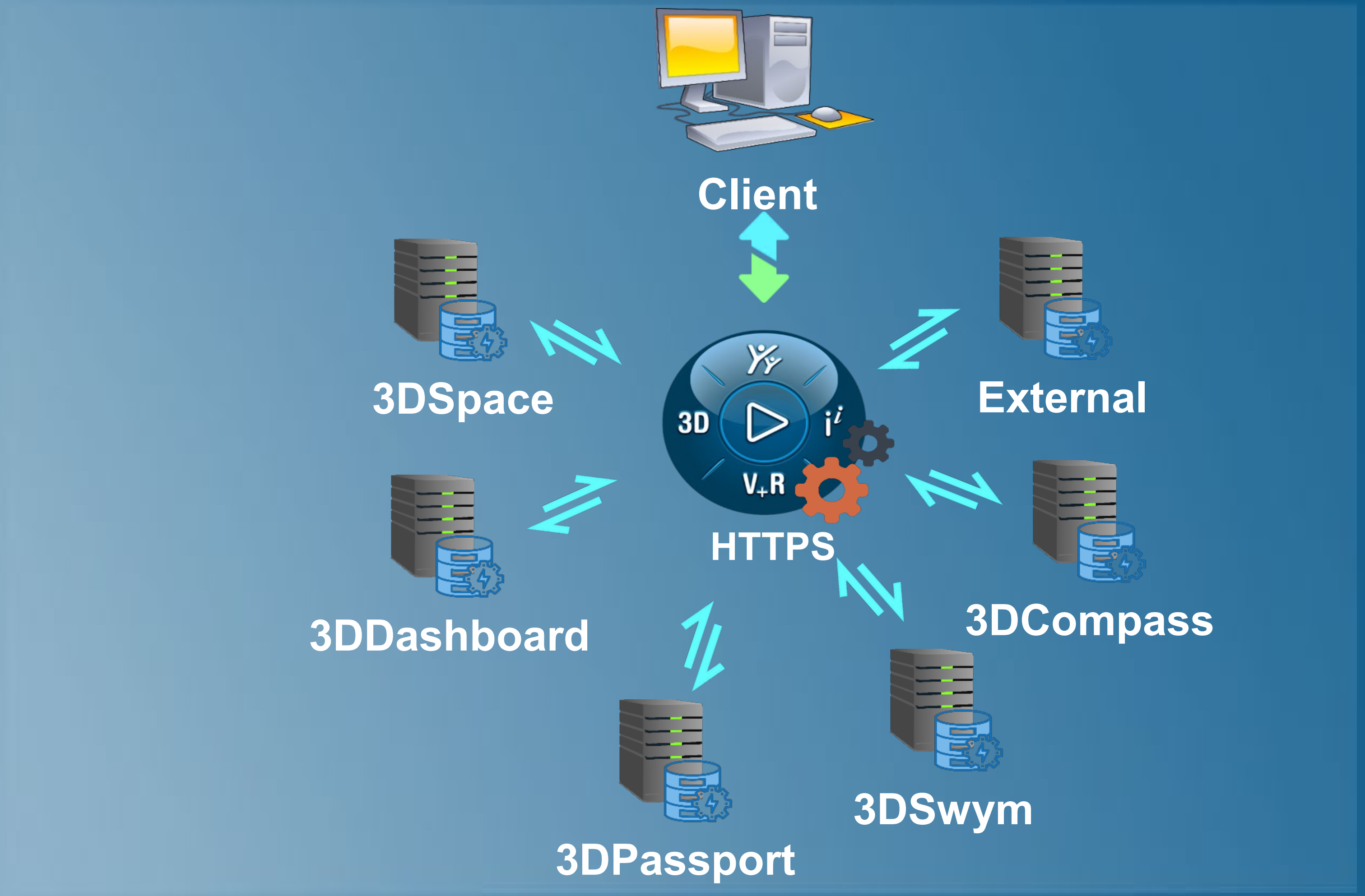
3DEXPERIENCE Web-Services

R2021x



About This Course

- **Course objectives**

Upon completion of this course you will acquire fundamentals notions required to:

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

- **Target audience**

Services Software Architect, Services Software Expert

- **Prerequisites**

The participant must have knowledge about:

- Java
- .NET/C#

6 Hours

Disclaimer

Caution/Warning

- This document is based on a various internal DS content for **3D**EXPERIENCE platform R2021x.
- 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 authori zation.

Table of Contents

Lesson 1: Introduction to 3DEXPERIENCE platform

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

Here are the topics to be covered:

1. 3DEXPERIENCE platform Architecture

3DEXPERIENCE platform Apps and Services

- Many different services, as can be seen below, come together to form what we know as the **3D**EXPERIENCE platform.

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

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

3DNotification
Stay Informed about events you subscribe to

3DPassport
Identity management and authentication across all apps.

3DCompass
Discover **3D**EXPERIENCE Worlds, access to roles and apps in one click

3DSwym
Leverage communities to share expertise, knowledge skills & conversations

3DMessaging
Connect instantly in context and share designs, collaborate through chat

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

3DPlay
Visualize and Review 3D designs in the real-time

3DDashboard
Aggregate all information that matters in personalized Dashboard

3DSearch & 6WTags
Search and Discover content. Add Semantics by Tagging the content 6 dimensions (Who-What-When-How-Where- Why)

3DEXCITE
CENTRICPLM
ENOVIA
SOLIDWORKS
CATIA
GEOVIA
BIOVIA
3D
V+R
NETVIBES
MEDIDATA
SIMULIA
DELMIA
3DVIA

3DPassport Control Center

Collaborative Spaces Control Center

Communities Control Center

Enterprise Control Center

Members Control Center

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

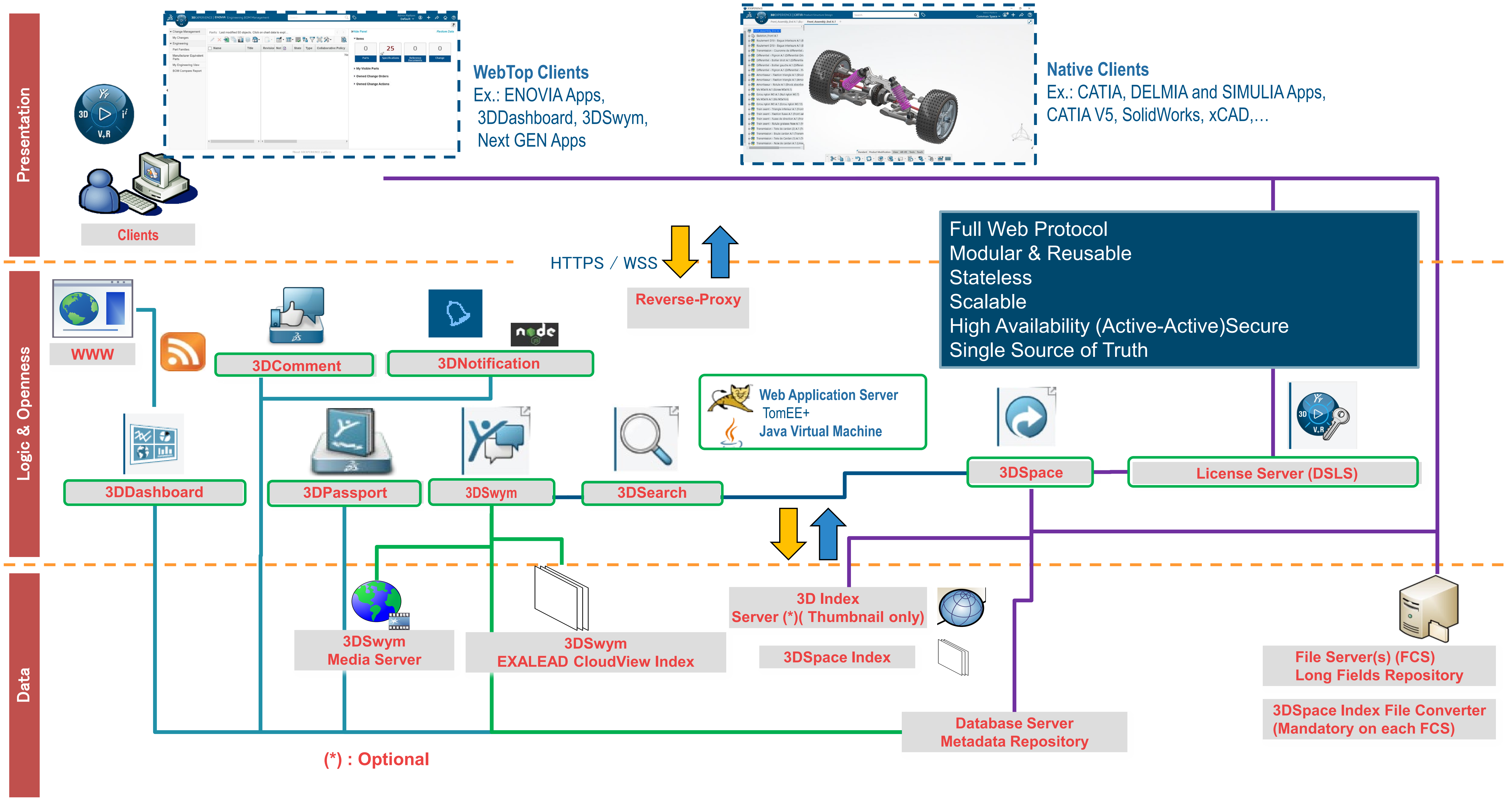
Mobile

On Cloud

3DPlay

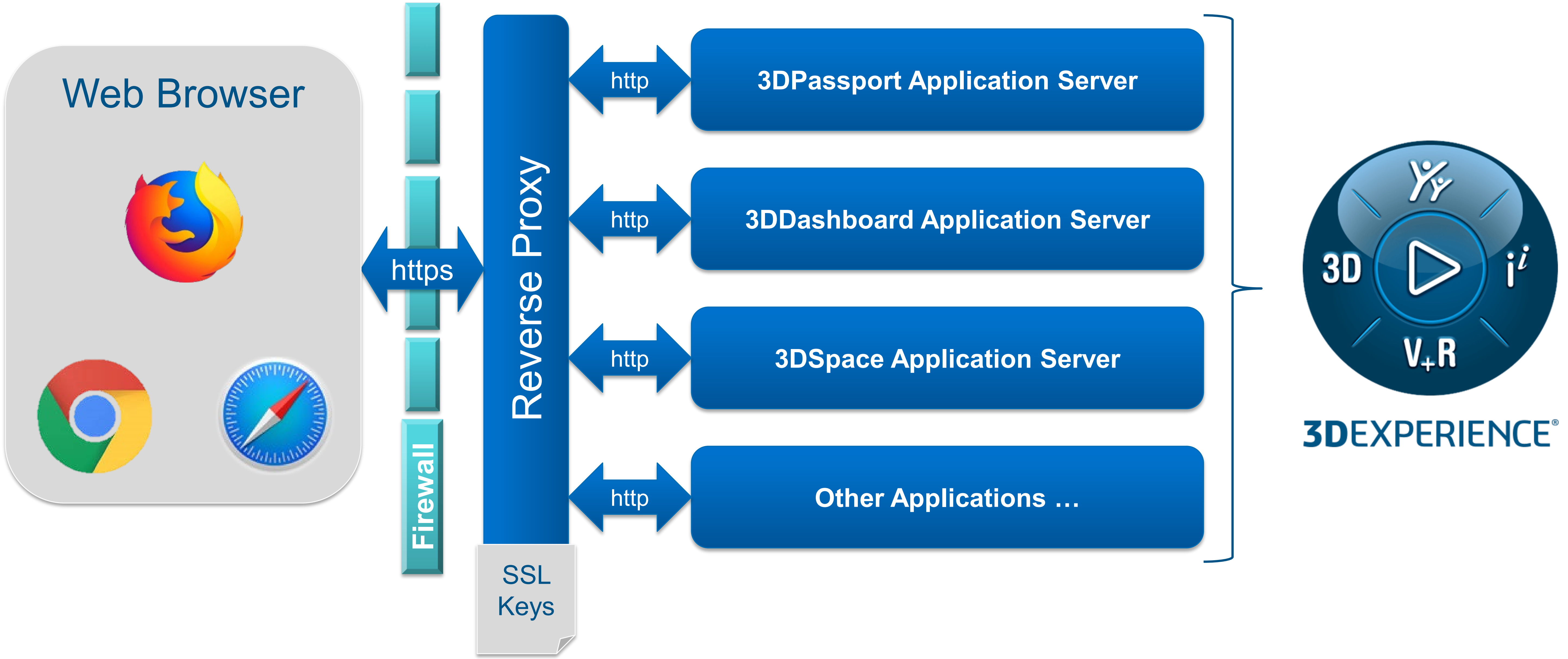
3DDrive

3DEXPERIENCE platform n-Tier Web Architecture (Logical View)



3DEXPERIENCE platform Application Interaction

- **3D**EXPERIENCE 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.



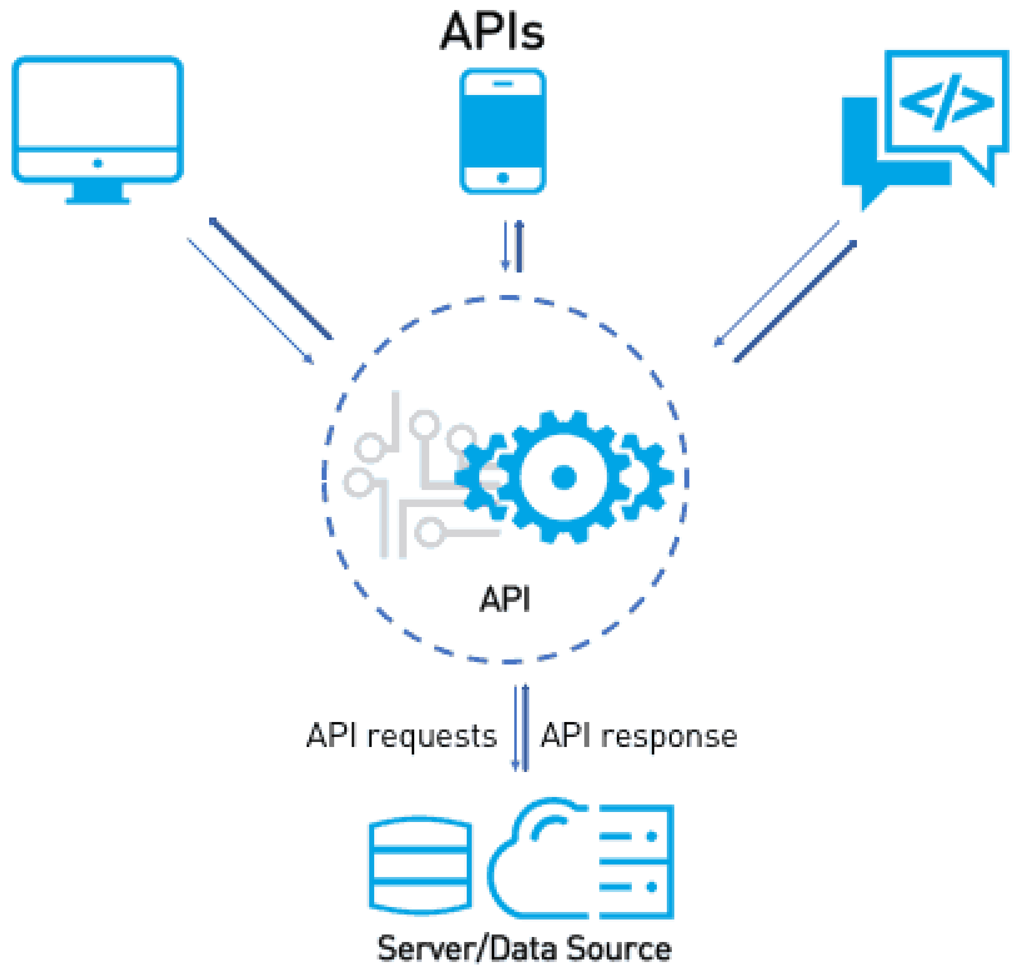
Lesson 2: 3DEXPERIENCE platform Web-Services

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

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

3.5 Hours

Introduction to Web-Services



Here are the topics to be covered:

1. **Introduction to Web-Services**
2. Types of Web-Services
3. Technologies used to Develop Web-Services
4. HTTP Methods for Web-Services
5. Building a Java Web-Service for **3D**EXPERIENCE 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.

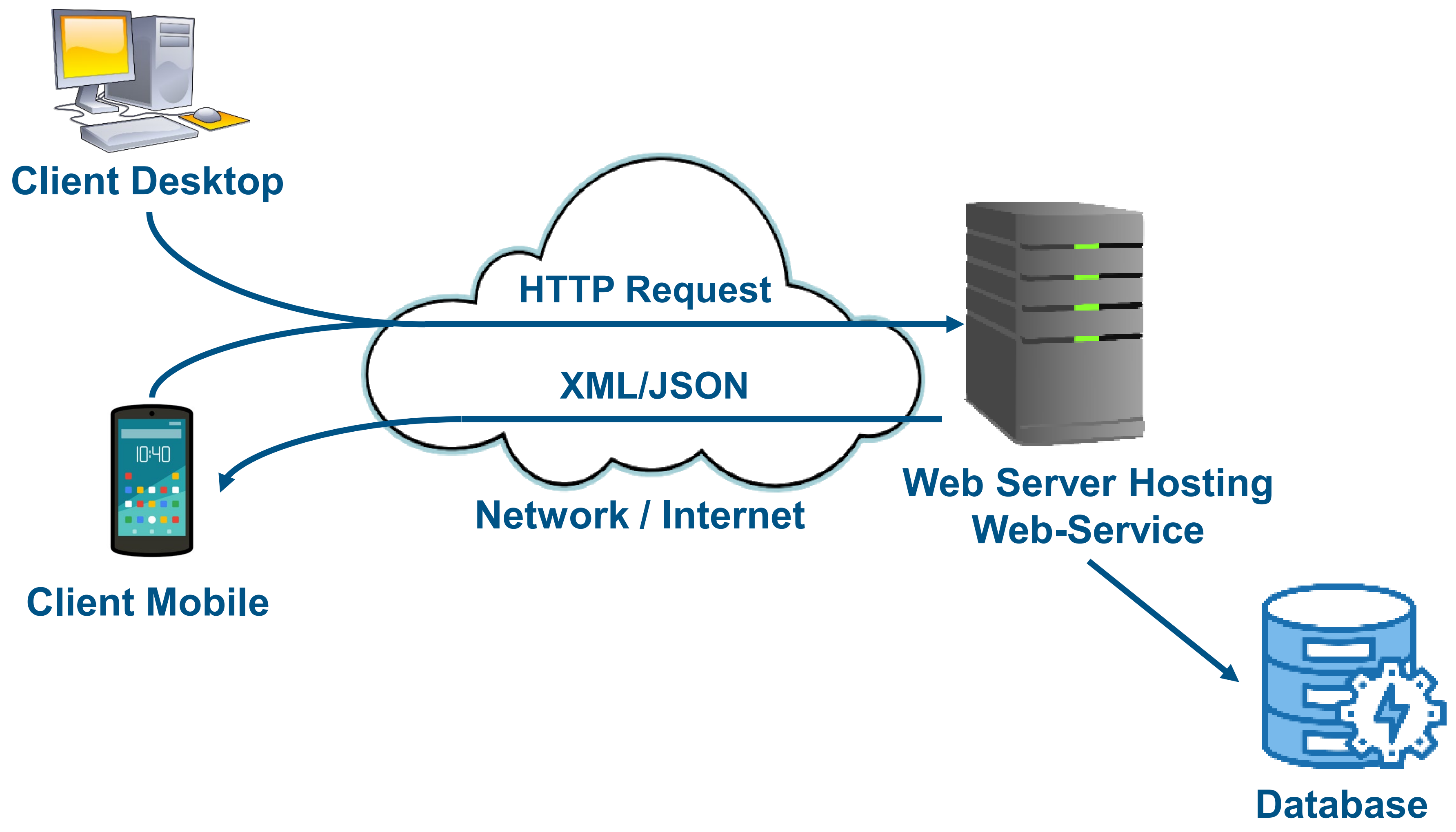
- Web-Services are typically designed to allow interoperability. This means that:
  - Web- Services are operating system independent.
    - The calling and responding applications may be hosted in environments with different operating systems.
  - Web-Services are programming language independent.
    - 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)

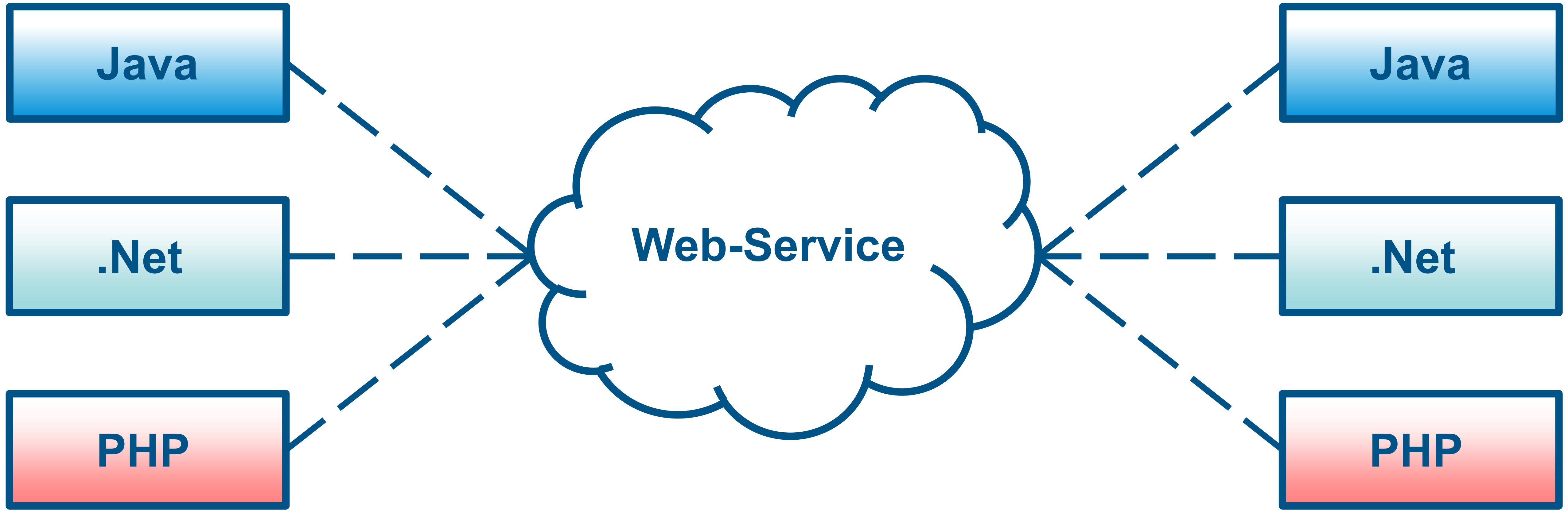
- In a standard Web-Service, the following hold true:
  - Standard protocols like HTTP are used for request and responses between Client machine and Server machine
  - Standard data exchange formats, like XML or JSON, are used to transfer data between Client machine and Server machine

- 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 clien ts 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.

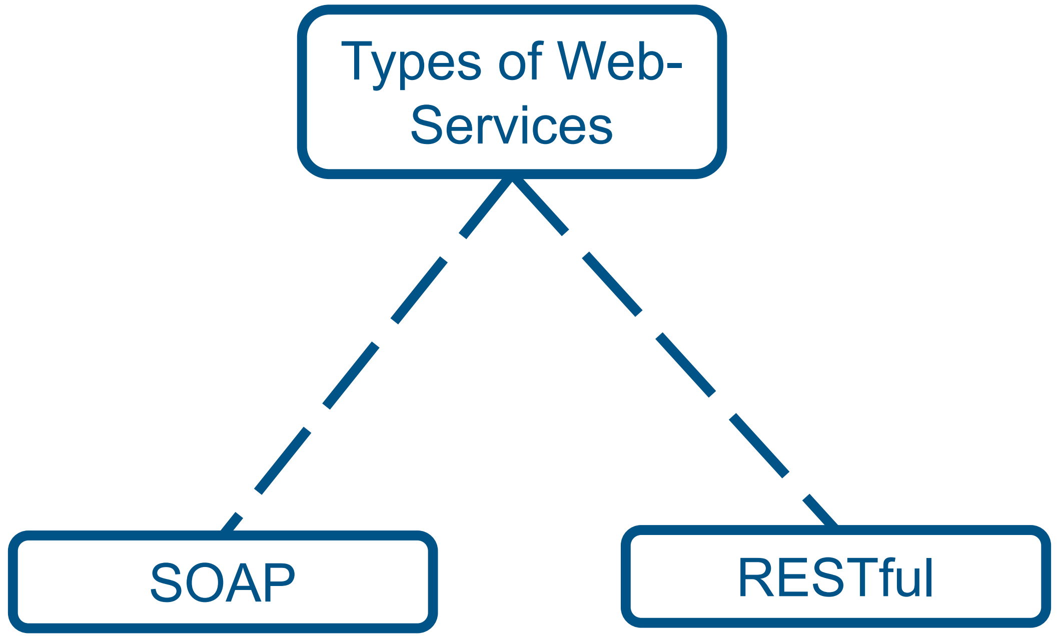
Types of Web-Services

In this lesson, we shall know about the different kinds of web-services and their features

Here are the topics to be covered:

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.

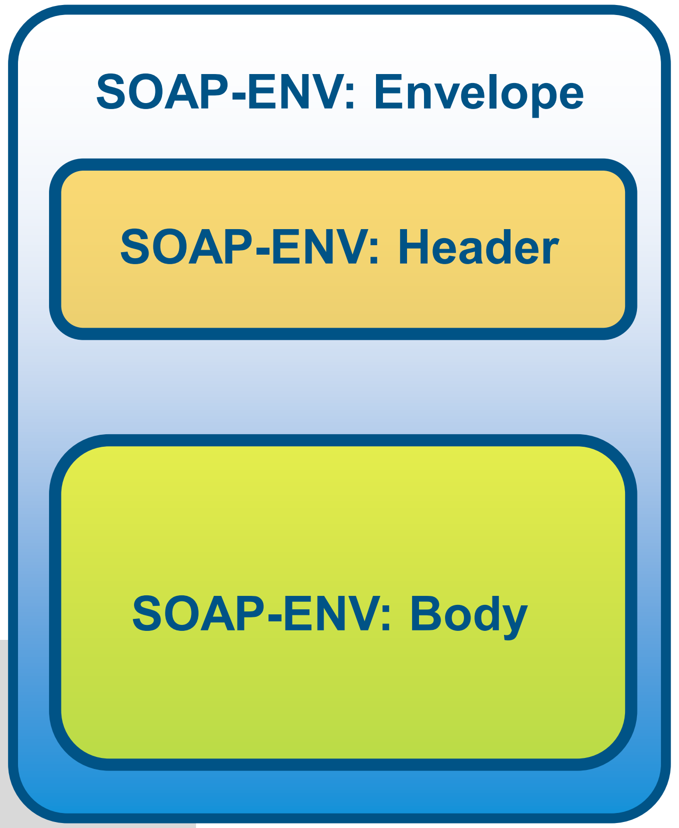
SOAP – Introduction

- **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 transpor t 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.

SOAP Message (1/5)

- The SOAP Message, or the data-packet that is transferred over the web-service, is an XML document which consists of the following components:

Element	Description	Mandatory
Envelope	Identifies the XML document as a SOAP message.	Yes
Header	Contains header information.	No
Body	Contains call and response information.	Yes
Fault	Provides information about errors that occurred while processing the message.	No



- **Example**: The below XML block is a SOAP message, where an application is requesting for stock price of a Company named XYZ.

```
POST /InStock HTTP/1.1
Host: www.example.org
Content-Type: application/soap+xml; charset=utf-8
Content-Length: 299
SOAPAction: "http://www.w3.org/2003/05/soap-envelope"

<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope" xmlns:m="http://www.example.org">
  <soap:Header>
  </soap:Header>
  <soap:Body>
    <m:GetStockPrice>
      <m:StockName>XYZ</m:StockName>
    </m:GetStockPrice>
  </soap:Body>
</soap:Envelope>
```

SOAP Message (2/5)

- The following rules are applicable for a SOAP message:
  - Every SOAP message needs to have a root Envelope element. It is absolutely mandatory.
  - Every Envelope element needs to have at least one SOAP Body element.
  - There can be only one Header tag, or none at all. The Header must appear as the first child of the Envelope, before the Body element.
  - A SOAP message must NOT contain a DTD reference
  - A SOAP message must NOT contain XML Processing Instructions
  - The Envelope changes when SOAP versions change.
    - A v1.1 -compliant SOAP processor generates a Fault upon receiving a message containing the v1.2 Envelope namespace.
    - A v1.2 -compliant SOAP processor generates a Version Mismatch fault if it receives a message that does not include the v1.2 envelope namespace.
  - The option Fault element, if present, must appear as a child element of the Body element.
    - A Fault element can only appear once in a SOAP message.

SOAP Message (3/5)

- The SOAP <**Header**> element contains application-specific information (like authentication, payment, etc.) about the SOAP message.

```
<?xml version="1.0"?>

<soap:Envelope
xmlns:soap="http://www.w3.org/2003/05/soap-envelope/"
soap:encodingStyle="http://www.w3.org/2003/05/soap-encoding">

<soap:Header>
  <m:Trans xmlns:m="https://www.xyz.com/transaction/" soap:mustUnderstand="1">234</m:Trans>
</soap:Header>
</soap:Envelope>
```

- The Header can have many attributes like:
  - **mustUnderstand**: Indicates whether the header entry is mandatory (value 1) or optional (value 0) for the recipient to process.
    Syntax: `soap:mustUnderstand="0|1"`

  - **actor**: A SOAP message may travel from a sender to a receiver by passing different endpoints along the message path. However, not all parts of a SOAP message may be intended for the ultimate endpoint, instead, it may be intended for one or more of the endpoints on the message path.

    The SOAP **actor** attribute is used to address the Header element to a specific endpoint.
    Syntax: `soap:actor="URI"`

  - **encodingStyle**: The encodingStyle attribute is used to define the data types used in the document. This attribute may appear on any SOAP element, and it will apply to that element's contents and all child elements. A SOAP message has no default encoding.
    Syntax: `soap:encodingStyle="URI"`

SOAP Message (4/5)

- The required SOAP **Body** element contains the actual SOAP message intended for the ultimate endpoint of the message.
- Immediate child elements of the SOAP Body element may be namespace-qualified.
- In the below examples we would see the SOAP messages for requesting the price of Apples from a company XYZ.
  - Note that the following elements in the below example are application-specific elements. They are not a part of the SOAP namespace
    - m:GetPrice
    - m:GetPriceResponse
    - m:Item
    - m:Price

Request

```
<?xml version="1.0"?>

<soap:Envelope
xmlns:soap="http://www.w3.org/2003/05/soap-envelope/"
soap:encodingStyle="http://www.w3.org/2003/05/soap-encoding">

<soap:Body>
  <m:GetPrice xmlns:m="https://www.xyz.com/prices">
    <m:Item>Apples</m:Item>
  </m:GetPrice>
</soap:Body>

</soap:Envelope>
```

Response

```
<?xml version="1.0"?>

<soap:Envelope
xmlns:soap="http://www.w3.org/2003/05/soap-envelope/"
soap:encodingStyle="http://www.w3.org/2003/05/soap-encoding">

<soap:Body>
  <m:GetPriceResponse xmlns:m="https://www.xyz.com/prices">
    <m:Price>1.90</m:Price>
  </m:GetPriceResponse>
</soap:Body>

</soap:Envelope>
```

SOAP Message (5/5)

- The optional SOAP Fault element is used to indicate error messages.
- The SOAP Fault element holds errors and status information for a SOAP Message. It has the following sub -elements:

Sub Element	Description
<faultcode>	A code for identifying the fault
<faultstring>	A human readable explanation of the fault
<faultactor>	Information about who caused the fault to happen
<detail>	Holds application specific error information related to the Body element

- The faultcode values defined below must be used in the faultcode element while describing faults:

Error	Description
VersionMismatch	Found an invalid namespace for the SOAP Envelope element
MustUnderstand	An immediate child element of the Header element, with the mustUnderstand attribute set to "1", was not understood
Client	The message was incorrectly formed or contained incorrect information
Server	There was a problem with the server so the message could not proceed

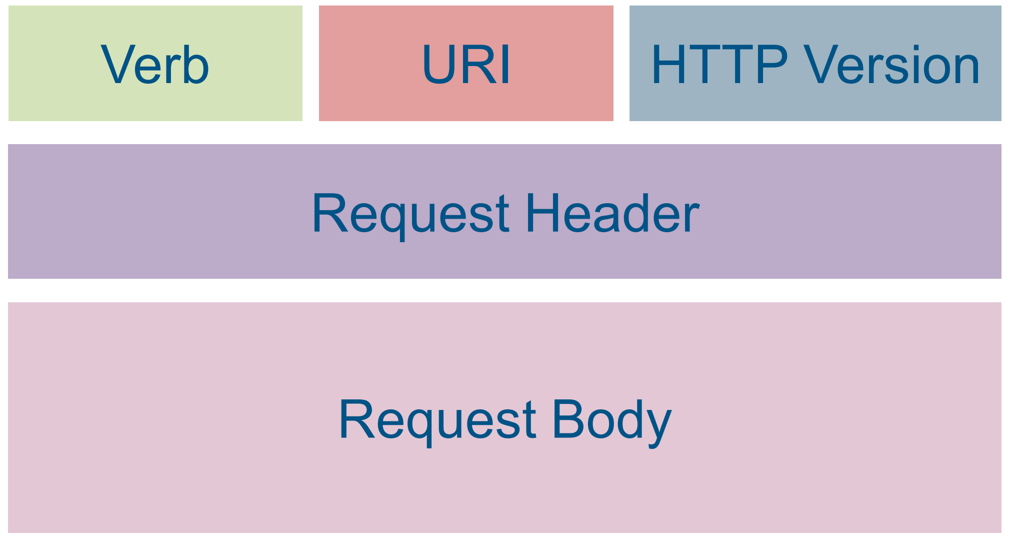
RESTful Web-Service – Introduction

- **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.

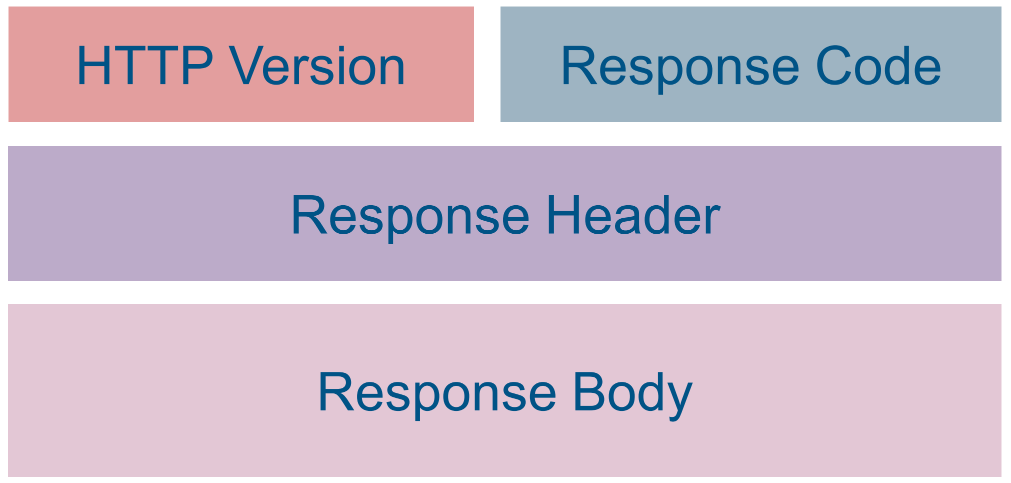
    In the adjacent example demonstrates a REST call, that transfers form data in the HTML body.



```
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:
  - **Status/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.

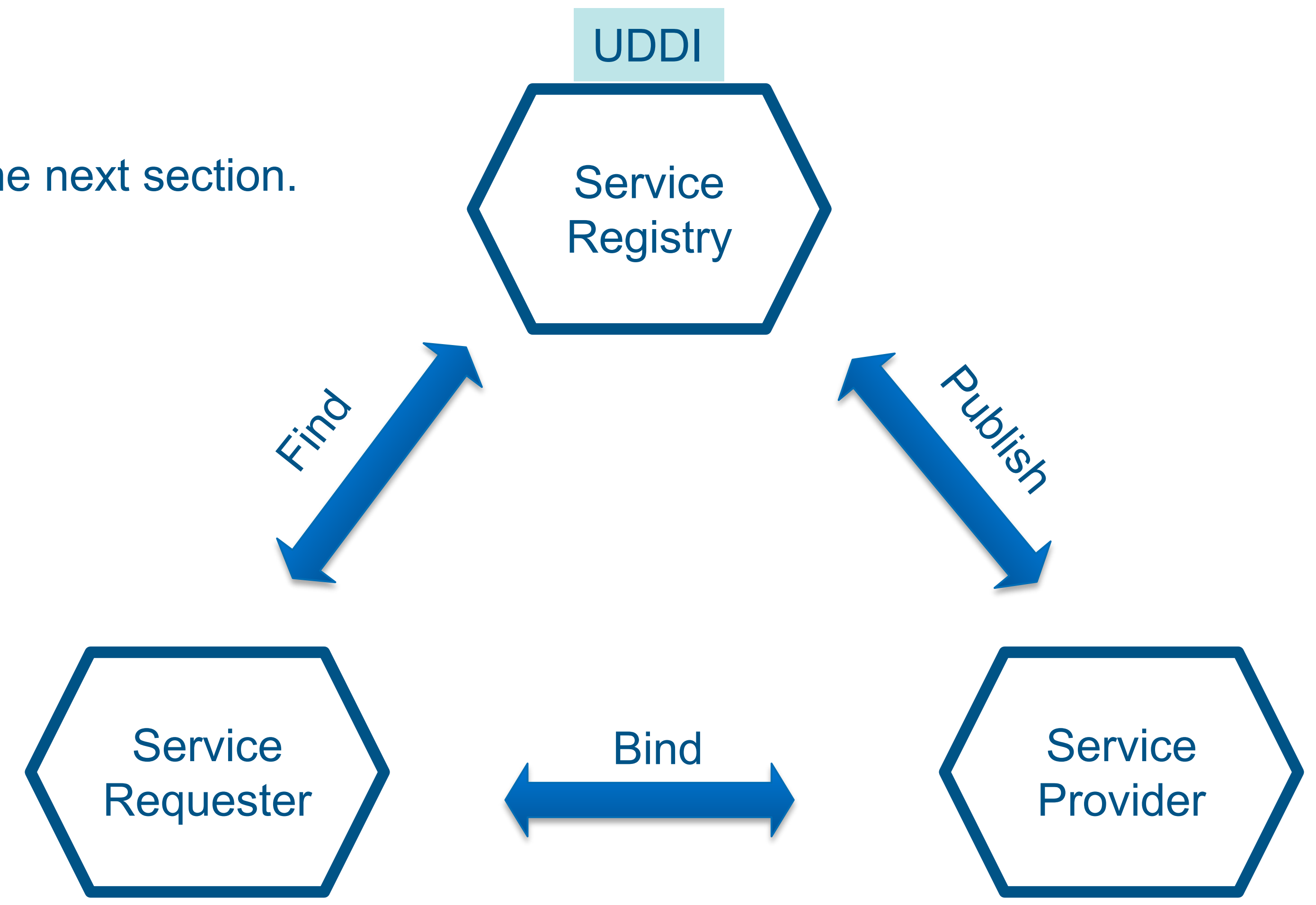
Technologies used to Develop Web-Services

In this lesson, we shall be introduced to the various technologies that are involved in implementing web-services

Here are the topics to be covered:

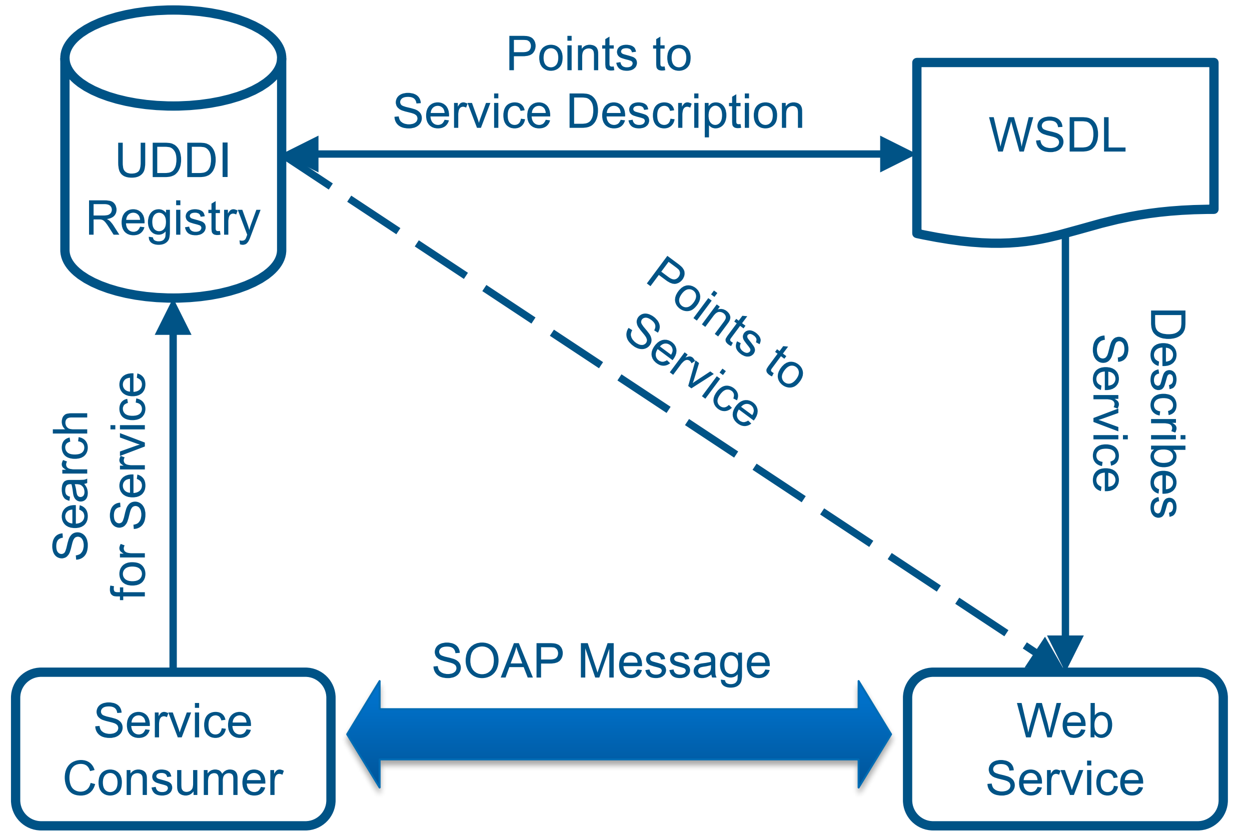
UDDI

- An organization can develop a Web-Service, but it will not be of use if the Users cannot find it, or if the Users do not know of its existence.
- **Universal Description, Discovery and Integration** (UDDI) is a platform independent protocol that includes an XML-based registry, by which businesses worldwide can list themselves on the Internet.
  - The **UDDI** also has mechanisms to register new services, and also to locate existing services
- Customers can lookup the UDDI to search for services provided by the registered companies.
  - The UDDI registry acts as the service-broker
- We shall discuss about the actual mechanism, with which UDDI works, in the next section.



UDDI – Working Architecture

- The adjacent diagram describes the working architecture of a SOAP Web-Service over the internet.



- The displayed architecture can be summarized as follows:
  - When a service provider wants to make its services available to service consumers, they would first describe the service in a **WSDL** document.
  - The service provider would then register the *WSDL* in a **UDDI** Registry.
  - The UDDI Registry would then maintain pointers to the WSDL document and to the Web-Service
  - When a Service Consumer wants to use a service, they would **lookup** the UDDI registry to find a services matching their requirement.
  - The UDDI then returns the WSDL description of the web-service, as well as the access point of the service.
  - The Service Consumer then uses the WSDL to **construct a SOAP Message**, which is then used to communicate with the actual web-service.
- We shall learn about the purpose and structure of the WSDL document in the next section.

WSDL

- **Web Services Description Language** (**WSDL**) is an XML format for describing network services as a set of endpoints operating on messages containing either document-oriented or procedure-oriented information.

- The operations and messages are described abstractly, and then bound to a concrete network protocol and message format to define an endpoint.

- In simple words, a WSDL document is an XML-based file that tells the client (Web-Service Consumer) about the following:
  - Purpose of the Web-Service
  - The exposed functionalities (methods)
  - How to call the Web-Service (expected inputs)
  - Expected output

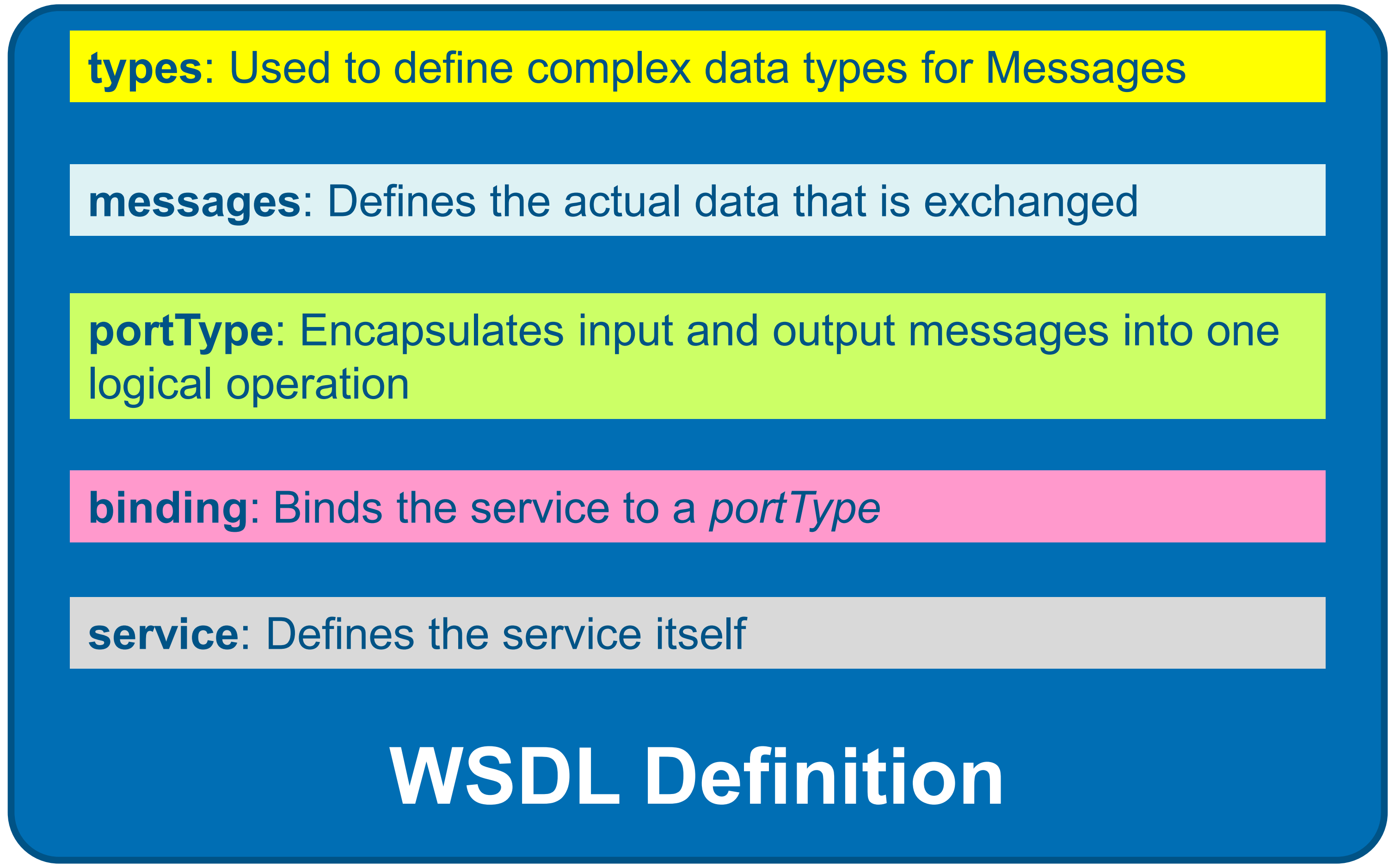
- The structure of a WSDL document is as shown in adjacent diagram.

  We shall discuss more about these elements in the following pages.

- If the URL of the Web-Service is `http://example.org:8080`

  then the WSDL file can be seen by the following URL call:

  `http://example.org:8080?wsdl`



types: Used to define complex data types for Messages

messages: Defines the actual data that is exchanged

portType: Encapsulates input and output messages into one logical operation

binding: Binds the service to a *portType*

service: Defines the service itself

WSDL Definition

Web Interface Definition Language (WIDL) was one of the pioneering specifications for description of remote Web-Services. It was based on an XML format, much similar to WSDL, and was suited to users of remote procedural technologies, such as RPC and CORBA. It, however, lost its significance with the rise of message-based XML technologies

WSDL – Example

```
<definitions name="TermRequestServiceDefinition" targetNamespace="http://example.org/wsdl/TermRequestService.wsdl"
    xmlns:tns="http://www.w3.org/2001/XMLSchema" xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns="http://schemas.xmlsoap.org/wsdl/">
    <message name="getTermRequest">
        <part name="term" type="xsd:string"/>
    </message>
    <message name="getTermResponse">
        <part name="value" type="xsd:string"/>
    </message>
    <portType name="glossaryTerms">
        <operation name="getTerm">
            <input message="getTermRequest"/>
            <output message="getTermResponse"/>
        </operation>
    </portType>
    <binding type="glossaryTerms" name="binding1">
        <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http" />
        <operation>
            <soap:operation soapAction="http://example.org/getTerm"/>
             <input><soap:body use="literal"/></input>
             <output><soap:body use="literal"/></output>
        </operation>
    </binding>
    <service name="TermRequestService">
        <documentation>TermRequestService</documentation>
        <port name="TermRequestPort" binding="tns:glossaryTerms">
            <soap:address location="http://example.org/TermRequest"/>
        </port>
    </service>
</definitions>
```

WSDL – Tags Used (1/2)

- Below are the descriptions of the most important tags in a WSDL file:

  - **<definitions>**: Encloses the entire WSDL definition into one unit. All other tags are placed within the *definitions* tag.

  - **<types>**: This tag is used to define complex datatypes, which will be used in the messages exchanged between the client application and the web-service.

```
<types>
    <schema targetNamespace = "http://example.com/stockquote.xsd"
            xmlns = "http://www.w3.org/2000/10/XMLSchema"
            xmlns:xs = "http://www.w3.org/2000/10/XMLSchema">
        <element name = "personinfo">
            <complexType>
                <sequence>
                    <element name="firstname" type="xs:string"/>
                    <element name="lastname" type="xs:string"/>
                    <element name="address" type="xs:string"/>
                    <element name="city" type="xs:string"/>
                    <element name="country" type="xs:string"/>
                </sequence>
            </complexType>
        </element>
    </schema>
</types>
```

It is to be noted that:

- WSDL is not tied exclusively to a specific typing system
- If the service uses only XML Schema built-in simple types, such as strings and integers, then **types** element is not required.

WSDL – Tags Used (2/2)

- **<message>**: This tag is used to define the message which is exchanged between the client application and the web server. These messages will explain the input and output operations which can be performed by the web-service.
- **<portType>**: This tag is used to encapsulate every input and output message into one logical operation.
- **<binding>**: This tag is used to bind the operation to the particular port type. This tag defines the protocol and data format for each type.
- **<service>**: This tag is a name given to the Web-Service itself. Here, we define the URL that will be called upon invoking the Web-Service.

WADL

- **WADL** is an acronym for **W**eb **A**pplication **D**escription **L**anguage.
- **WADL** is an XML document which describes Web-Services that work over the HTTP protocol.
  - **WADL** is considered as the REST equivalent to SOAP's WSDL.
- **WADL** is lightweight, easier to understand, and easier to write as compared to WSDL.
  - WADL was championed by Sun Microsystems.
- In some aspects, *WADL* is not as flexible as WSDL (no binding for SMTP servers), but it is sufficient for any REST service and is much less verbose.
- Some key aspects of *WADL* are as follows:
  - WADL is intended to simplify the reuse of web-services that are based on the existing HTTP architecture of the Web.
  - The service described using a set of resource elements
    - Each resource contains param elements to describe the inputs, and method elements which describe the request and response of a resource.
  - Use of WADL is not considered standard.
  - REST Web-Services do not actually need any separate WADL definition. They may use URIs to allow discovery of their resources.

WADL – Example (1/2)

- Let us assume the following REST application code:

```
@Path("/")
public class MyResource {
    @Path("/test1")
    @GET
    public Response handle() {
        Response r = Response.ok("Response test1 with GET, body content").build();
        return r;
    }
    @Path("/test2")
    @POST
    public Response handle2() {
        Response r = Response.ok("Response test2 with POST, body content").build();
        return r;
    }
}
```

We shall discuss more about REST service code in other chapters.

- Let us assume that the resources of the above application is accessed over the URI: `http://example.org:8080`

- In the above case, assuming that the REST service has been developed using Jersey API, then the WADL may be seen by calling the following URL:

  `http://example.org:8080/application.wadl`

WADL – Example (2/2)

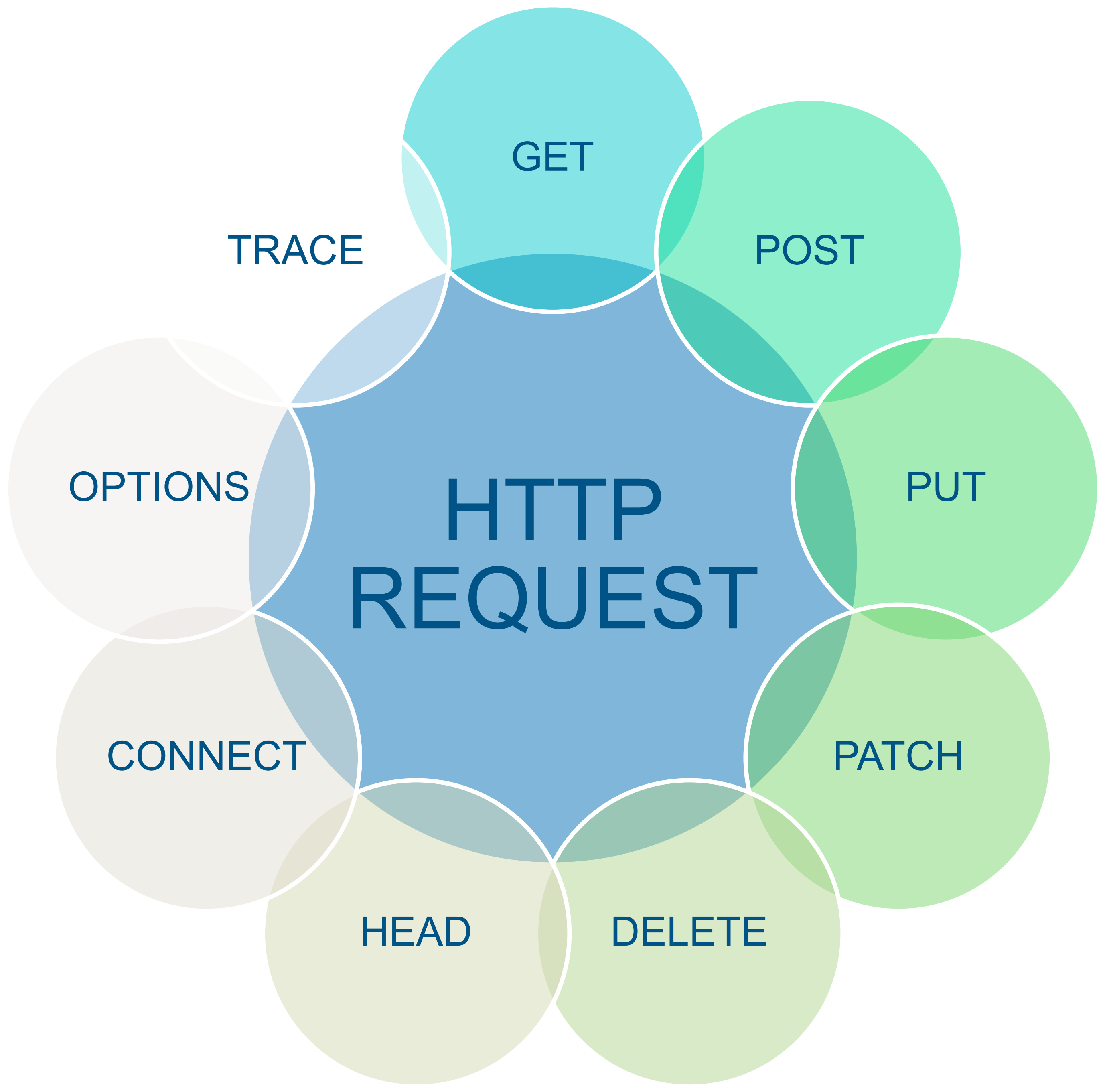
- The generated WADL document may look similar to below:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<application xmlns="http://wadl.dev.java.net/2009/02">
    <doc xmlns:jersey="http://jersey.java.net/" jersey:generatedBy="Jersey: 2.25.1 2021-05-19 17:19:50"/>
    <doc xmlns:jersey="http://jersey.java.net/" jersey:hint="This is simplified WADL with user and core
resources only. To get full WADL with extended resources use the query parameter detail. Link:
http://example.org:8080/application.wadl?detail=true"/>
    <grammars/>
    <resources base="http://example.org:8080/">
        <resource path="/">
            <resource path="/test1">
                <method id="handle1" name="GET">
                    <response>
                        <representation mediaType="*/*"/>
                    </response>
                </method>
            </resource>
            <resource path="/test2">
                <method id="handle2" name="POST">
                    <response>
                        <representation mediaType="*/*"/>
                    </response>
                </method>
            </resource>
        </resource>
    </resources>
</application>
```

Web-Service Discovery in 3DEXPERIENCE platform

- Although the **3D**EXPERIENCE 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 **3D**EXPERIENCE platform
  - New web-services developed using SOAP protocol will not be guaranteed by DS
- Only REST web-services are now supported in the **3D**EXPERIENCE platform. REST web-services are discovered using their URI.
  - The URI of a JAX-RS web-service is configured by the use of **@ApplicationPath** and **@Path** annotations. We shall discuss about these annotations in detail in a later chapter.

HTTP Methods for Web-Services



Here are the topics to be covered:

- ✓ 1. *Introduction to Web-Services*
- ✓ 2. *Types of Web-Services*
- ✓ 3. *Technologies used to Develop Web-Services*
- **4. HTTP Methods for Web-Services**
- 5. Building a Java Web-Service for **3D**EXPERIENCE platform

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 – GET vs POST

- The following table compares the two HTTP methods: **GET** and **POST**

	GET	**POST**
BACK button/Reload	Harmless	Data will be re-submitted (the browser should alert the user that the data are about to be re-submitted)
Bookmarked	Can be bookmarked	Cannot be bookmarked
Cached	Can be cached	Not cached
Encoding type	application/x-www-form-urlencoded	application/x-www-form-urlencoded or multipart/form-data. Use multipart encoding for binary data
History	Parameters remain in browser history	Parameters are not saved in browser history
Restrictions on data length	Yes, when sending data, the GET method adds the data to the URL; and the length of a URL is limited (maximum URL length is 2048 characters)	No restrictions
Restrictions on data type	Only ASCII characters allowed	No restrictions. Binary data is also allowed
Security	GET is less secure compared to POST because data sent is part of the URL Never use GET when sending passwords or other sensitive information!	POST is a little safer than GET because the parameters are not stored in browser history or in web server logs
Visibility	Data is visible to everyone in the URL	Data is not displayed in the URL

Building a Java Web-Service for 3DEXPERIENCE platform

In this lesson, we shall explore the following:

- **Java Libraries required to create a standard 3DEXPERIENCE platform web-service**
- **Sample Java web-service code**

Here are the topics to be covered:

✓ 1. *Introduction to Web-Services*
✓ 2. *Types of Web-Services*
✓ 3. *Technologies used to Develop Web-Services*
✓ 4. *HTTP Methods for Web-Services*
5. **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 **3D**EXPERIENCE 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

JAX-RS API – Introduction

- **JAX-RS** is a JAVA based programming language API and specification to provide support for creating RESTful Web-Services.
- **JAX-RS** uses annotations available from Java SE 5 to simplify the development of web-services and its subsequent deployment.
- From version 1.1 on, **JAX-RS** is an official part of Java EE 6. A notable feature of being an official part of Java EE is that no configuration is necessary to start using JAX-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
- **JAX-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

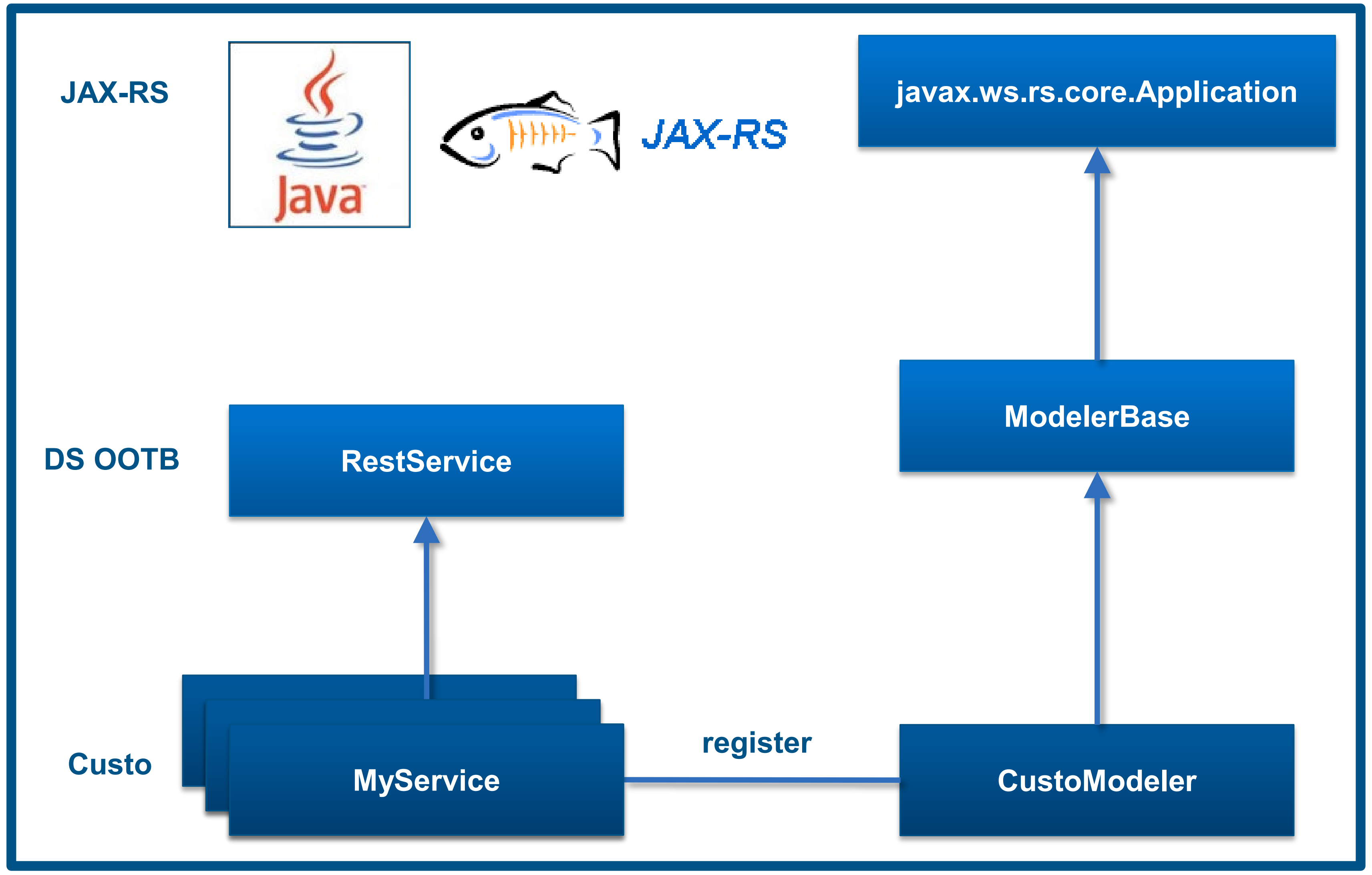
Annotation	Description
@ApplicationPath	Identifies the application path that serves as the base URI for all resources. This annotation is applied only to a subclass of `javax.ws.rs.core.Application`
@Path	Identifies the *Relative* path of resource class/method, relative to @ApplicationPath
@PathParam	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 javax.ws.rs.ApplicationPath;

@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 javax.json.Json;
import javax.json.JsonObjectBuilder;
import javax.servlet.http.HttpServletRequest;
import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.Produces;
import javax.ws.rs.QueryParam;
import javax.ws.rs.core.Response;

@Path("/add")
@Produces({"application/json"})
public class DSISAdd extends RestService {
    @GET
    public Response addFunction(@javax.ws.rs.core.Context HttpServletRequest request, @QueryParam("num1") String num1, @QueryParam("num2") String num2) {
        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 javax.json.Json;
import javax.json.JsonObjectBuilder;
import javax.servlet.http.HttpServletRequest;
import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.Produces;
import javax.ws.rs.QueryParam;
import javax.ws.rs.core.Response;

@Path("/subtract")
@Produces({"application/json"})
public class DSISSubtract extends RestService {
    @GET
    public Response subtractFunction(@javax.ws.rs.core.Context HttpServletRequest request, @QueryParam("num1") String num1, @QueryParam("num2") String num2) {
        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**

```java
package com.webservices;

import com.dassault_systemes.platform.restServices.RestService;
import javax.json.Json;
import javax.json.JsonObjectBuilder;
import javax.servlet.http.HttpServletRequest;
import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.Produces;
import javax.ws.rs.QueryParam;
import javax.ws.rs.core.Response;

@Path("/multiply")
@Produces({"application/json"})
public class DSISMultiply extends RestService {
    @GET
    public Response multiplyFunction(@javax.ws.rs.core.Context HttpServletRequest request, @QueryParam("num1") String num1, @QueryParam("num2") String num2) {
        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)

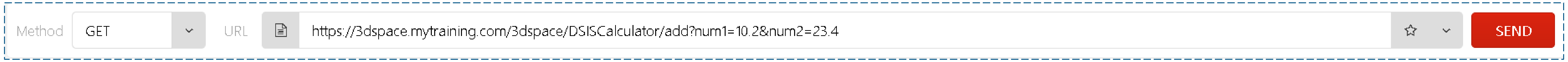
- Similarly, we have the service class **DSISDivide**

```
package com.webservices;
import com.dassault_systemes.platform.restServices.RestService;
import javax.json.Json;
import javax.json.JsonObjectBuilder;
import javax.servlet.http.HttpServletRequest;
import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.Produces;
import javax.ws.rs.QueryParam;
import javax.ws.rs.core.Response;
@Path("/divide")
@Produces({"application/json"})
public class DSISDivide extends RestService {
    @GET
    public Response divideFunction(@javax.ws.rs.core.Context HttpServletRequest request, @QueryParam("num1") String num1, @QueryParam("num2") String num2) {
        float val1, val2;
        String result = new String();
        JsonObjectBuilder output = Json.createObjectBuilder();
        try {
            val1 = Float.parseFloat(num1);
            val2 = Float.parseFloat(num2);
            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)

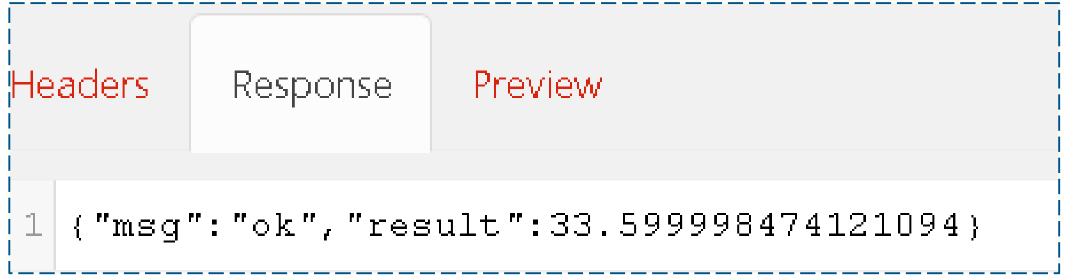
- Let us assume the following about the deployment of the developed web-service:
  - The Web-Service is deployed to 3DSpace
  - The root URL of the 3DSpace application is `https://3dspace.mytraining.com/3dspace`
- We wish to add two numbers **10.2** and **23.4**
- After studying the code in the previous 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`



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

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



```
{"msg":"ok","result":33.599998474121094}
```

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 secur ity context for the session.

Lesson 3: Consuming 3DEXPERIENCE platform Web-Services

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

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

2 Hours

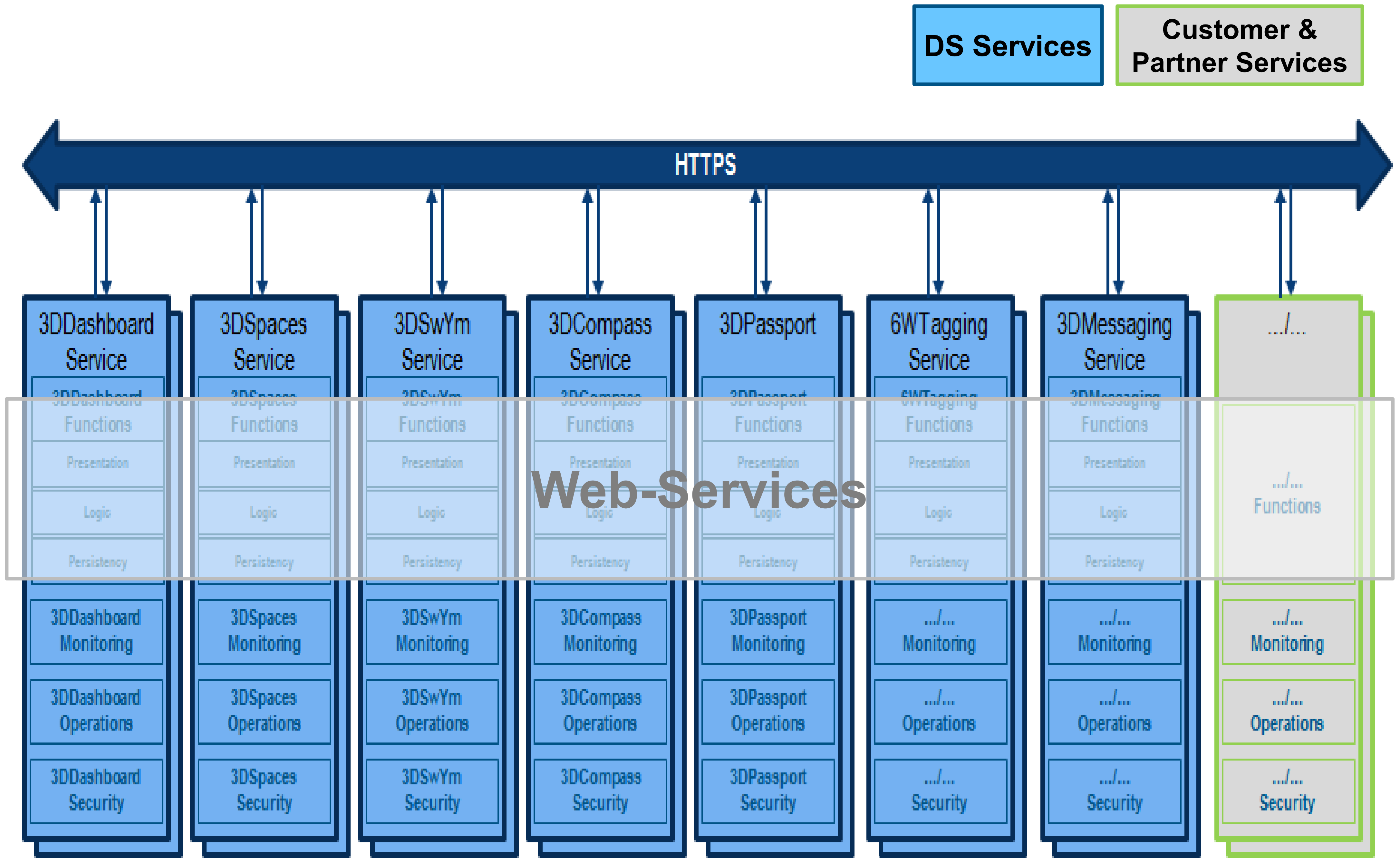
Consuming Web-Services in 3DEXPERIENCE platform

In this chapter, we shall be briefly introduced to how Web-Services are consumed in 3DEXPERIENCE platform

Here are the topics to be covered:

Consuming Web-Services in 3DEXPERIENCE platform (1/3)

- 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 3DEXPERIENCE platform (2/3)

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

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

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 exposed.

- 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 – Widgets

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

- For calling **3D**EXPERIENCE 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>";
    }
});
```

It is to be noted that only **Additional App** type widgets are allowed to call web-services that are hosted on the **3D**EXPERIENCE platform.

This is because *Additional App* widgets are placed on the trusted domain.

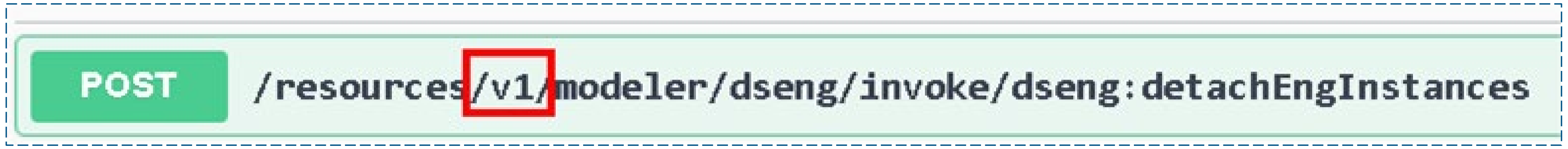
- Please refer to the *3DEXPERIENCE Widget Development Fundamentals* course for more technical details about web-services by 3DDashboard widgets.

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 **3D**EXPERIENCE 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 **3D**EXPERIENCE platform OOTB.
- The version number of an OOTB web-service can be spotted on it URI, as highlighted in the below image:



```
POST  /resources/v1/modeler/dseng/invoke/dseng:detachEngInstances
```

- 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

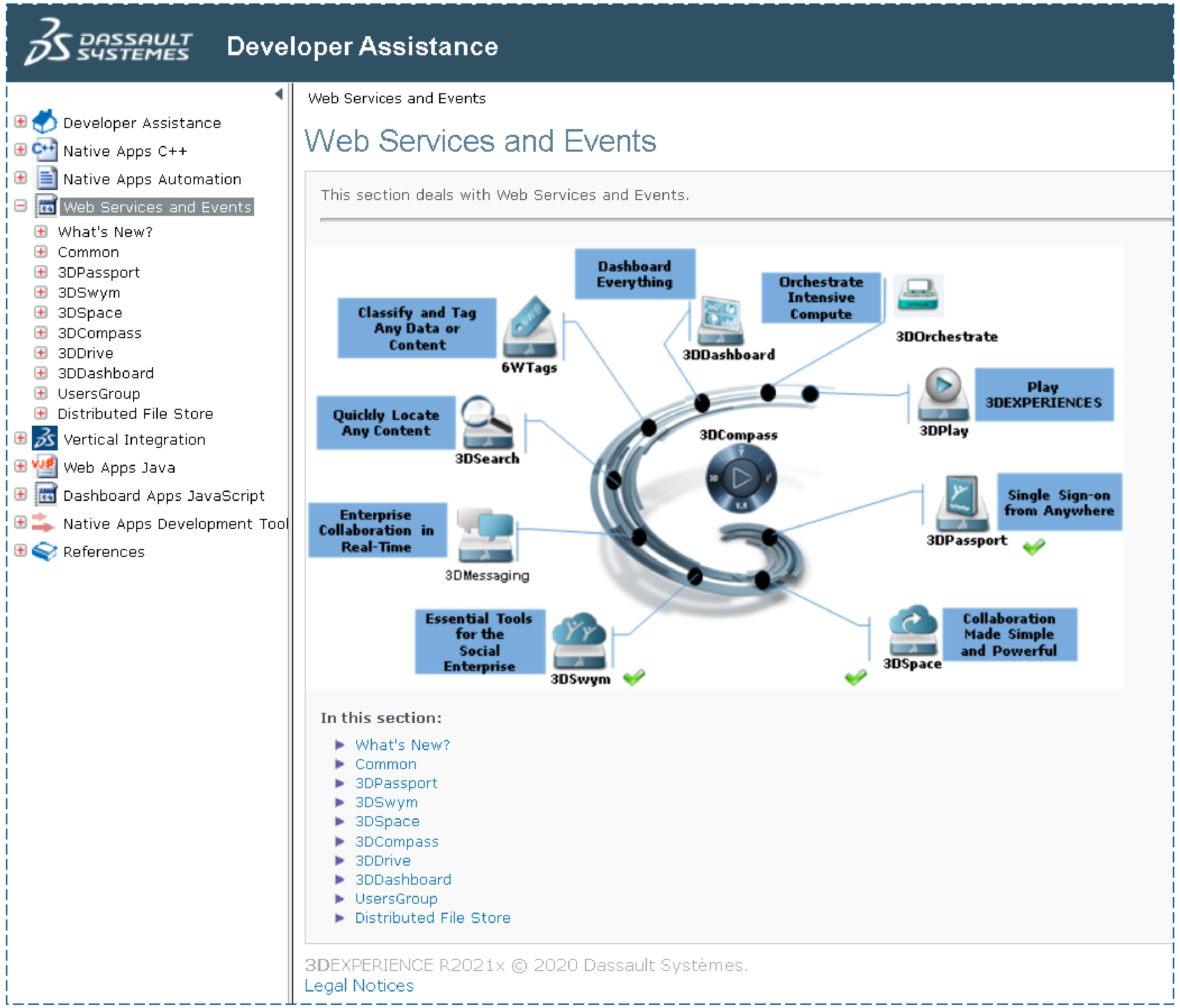
Here are the topics to be covered:

✓ 1. *Consuming Web-Services in **3D**EXPERIENCE platform*
2. **Documentation for 3DEXPERIENCE platform Web-Services**
3. Tools for Testing Web-Services
4. 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/R2021x/en/DSDoc.htm?show=CAAiamREST/CAATciamRESTToc.htm



DASSAULT SYSTEMES Developer Assistance

- Developer Assistance
- Native Apps C++
- Native Apps Automation
- Web Services and Events
  - What's New?
  - Common
  - 3DPassport
  - 3DSwym
  - 3DSpace
  - 3DCompass
  - 3DDrive
  - 3DDashboard
  - UsersGroup
  - Distributed File Store
- Vertical Integration
- Web Apps Java
- Dashboard Apps JavaScript
- Native Apps Development Tool
- References

Web Services and Events

Web Services and Events

This section deals with Web Services and Events.

Dashboard Everything
Classify and Tag Any Data or Content
6WTags
3DDashboard
Orchestrate Intensive Compute
3DOrchestrate
Play 3DEXPERIENCES
3DPlay
3DCompass
Quickly Locate Any Content
3DSearch
Single Sign-on from Anywhere
3DPassport
Enterprise Collaboration in Real-Time
3DMessaging
Essential Tools for the Social Enterprise
3DSwym
Collaboration Made Simple and Powerful
3DSpace

In this section:

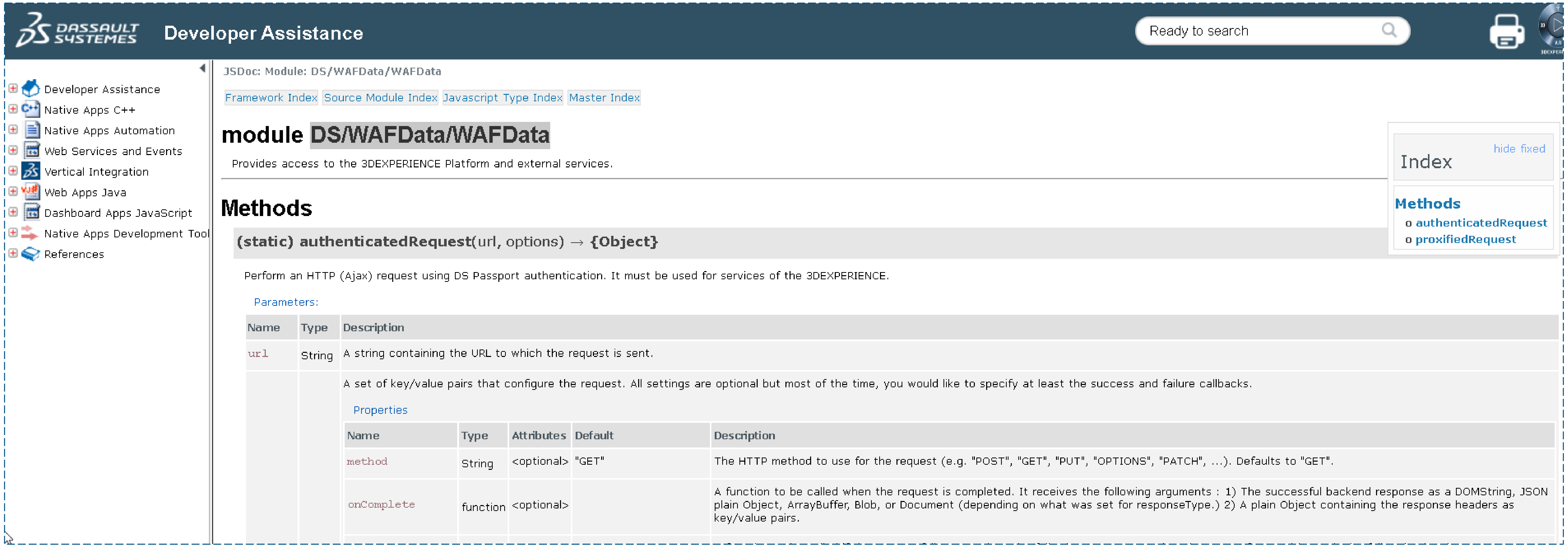
- What's New?
- Common
- 3DPassport
- 3DSwym
- 3DSpace
- 3DCompass
- 3DDrive
- 3DDashboard
- UsersGroup
- Distributed File Store

3DEXPERIENCE R2021x © 2020 Dassault Systèmes.
Legal Notices

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/R2021x/en/DSDoc.htm?show=../generated/js/WebAppsFoundations-WAFData/module-DS_WAFData_WAFData.htm



JSDoc: Module: DS/WAFData/WAFData

Framework Index | Source Module Index | Javascript Type Index | Master Index

module DS/WAFData/WAFData

Provides access to the 3DEXPERIENCE Platform and external services.

Methods

(static) authenticatedRequest(url, options) → {Object}

Perform an HTTP (Ajax) request using DS Passport authentication. It must be used for services of the 3DEXPERIENCE.

Parameters:

Name	Type	Description
url	String	A string containing the URL to which the request is sent.
		A set of key/value pairs that configure the request. All settings are optional but most of the time, you would like to specify at least the success and failure callbacks.

Properties

Name	Type	Attributes	Default	Description
method	String	<optional>	"GET"	The HTTP method to use for the request (e.g. "POST", "GET", "PUT", "OPTIONS", "PATCH", ...). Defaults to "GET".
onComplete	function	<optional>		A function to be called when the request is completed. It receives the following arguments : 1) The successful backend response as a DOMString, JSON plain Object, ArrayBuffer, Blob, or Document (depending on what was set for responseType.) 2) A plain Object containing the response headers as key/value pairs.

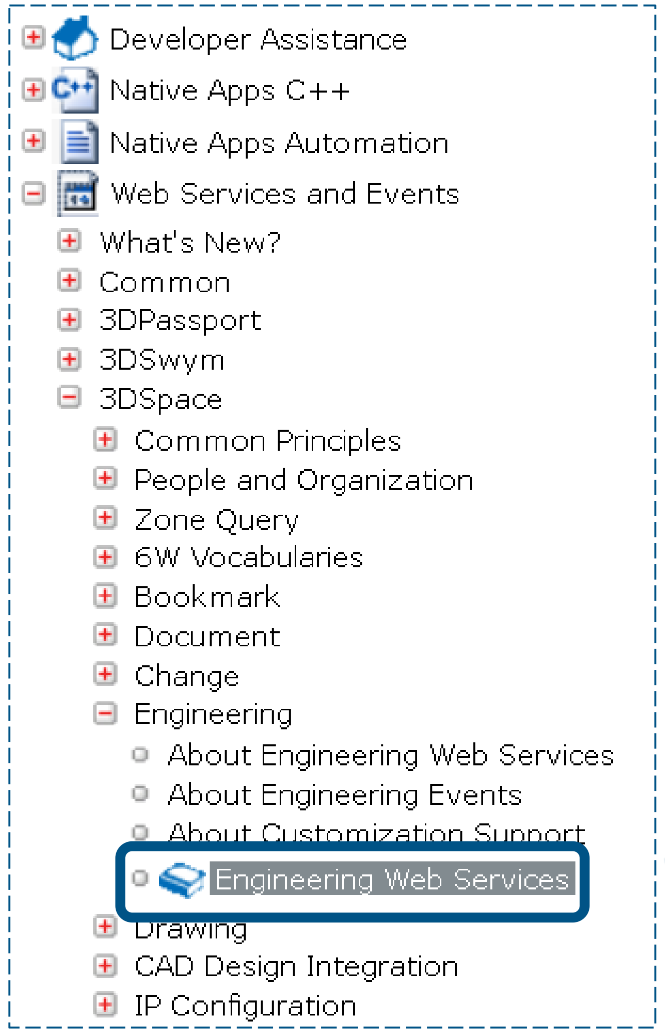
Reading Developer Assistance – Example

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

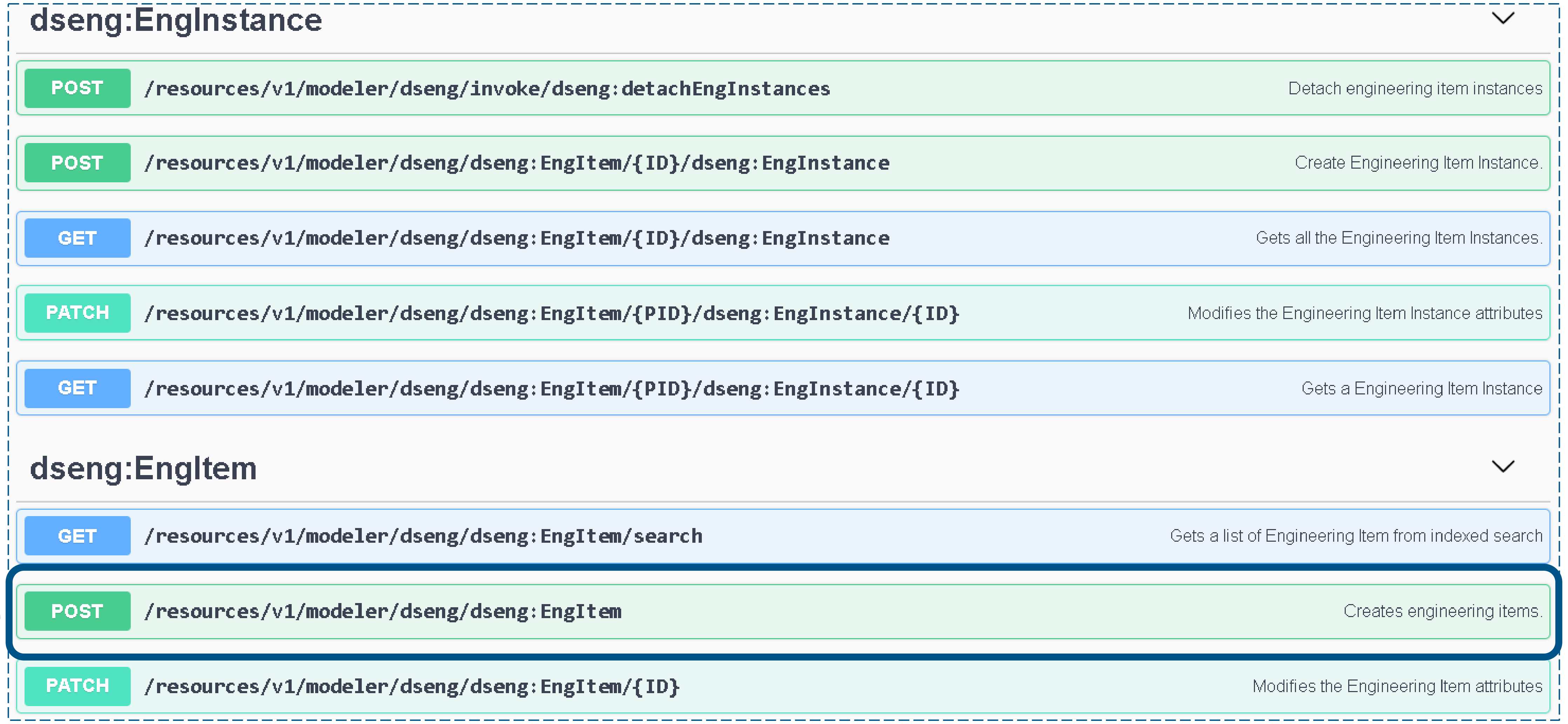
- In the following example, we shall try to find the documentation for web-service to create an *Engineering Item*.
  - Engineering Items come under the scope of 3DSpace services. Hence the relevant information would be found with the following navigation:

    `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**.



- Developer Assistance
- Native Apps C++
- Native Apps Automation
- Web Services and Events
  - What's New?
  - Common
  - 3DPassport
  - 3DSwym
  - 3DSpace
    - Common Principles
    - People and Organization
    - Zone Query
    - 6W Vocabularies
    - Bookmark
    - Document
    - Change
    - Engineering
      - About Engineering Web Services
      - About Engineering Events
      - About Customization Support
      - Engineering Web Services
    - Drawing
    - CAD Design Integration
    - IP Configuration



dseng:EngInstance

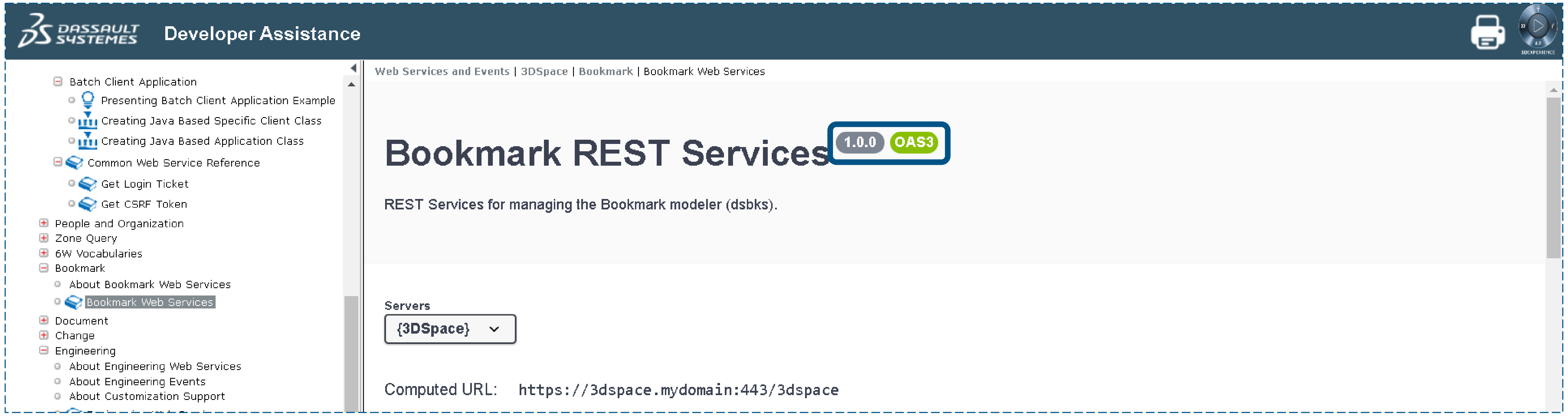
POST	/resources/v1/modeler/dseng/invoke/dseng:detachEngInstances	Detach engineering item instances
POST	/resources/v1/modeler/dseng/dseng:EngItem/{ID}/dseng:EngInstance	Create Engineering Item Instance.
GET	/resources/v1/modeler/dseng/dseng:EngItem/{ID}/dseng:EngInstance	Gets all the Engineering Item Instances.
PATCH	/resources/v1/modeler/dseng/dseng:EngItem/{PID}/dseng:EngInstance/{ID}	Modifies the Engineering Item Instance attributes
GET	/resources/v1/modeler/dseng/dseng:EngItem/{PID}/dseng:EngInstance/{ID}	Gets a Engineering Item Instance

dseng:EngItem

GET	/resources/v1/modeler/dseng/dseng:EngItem/search	Gets a list of Engineering Item from indexed search
POST	/resources/v1/modeler/dseng/dseng:EngItem	Creates engineering items.
PATCH	/resources/v1/modeler/dseng/dseng:EngItem/{ID}	Modifies the Engineering Item attributes

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:



- Batch Client Application
  - Presenting Batch Client Application Example
  - Creating Java Based Specific Client Class
  - Creating Java Based Application Class
- Common Web Service Reference
  - Get Login Ticket
  - Get CSRF Token
- People and Organization
- Zone Query
- 6W Vocabularies
- Bookmark
  - About Bookmark Web Services
  - Bookmark Web Services
- Document
- Change
- Engineering
  - About Engineering Web Services
  - About Engineering Events
  - About Customization Support

Web Services and Events | 3DSpace | Bookmark | Bookmark Web Services

Bookmark REST Services 1.0.0 OAS3

REST Services for managing the Bookmark modeler (dsbks).

Servers

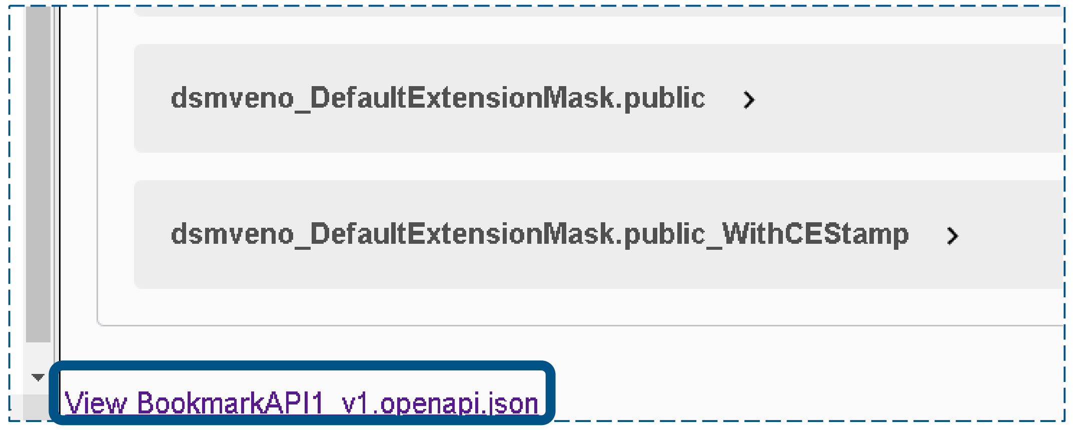
{3DSpace}

Computed URL: `https://3dspace.mydomain:443/3dspace`

- 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)

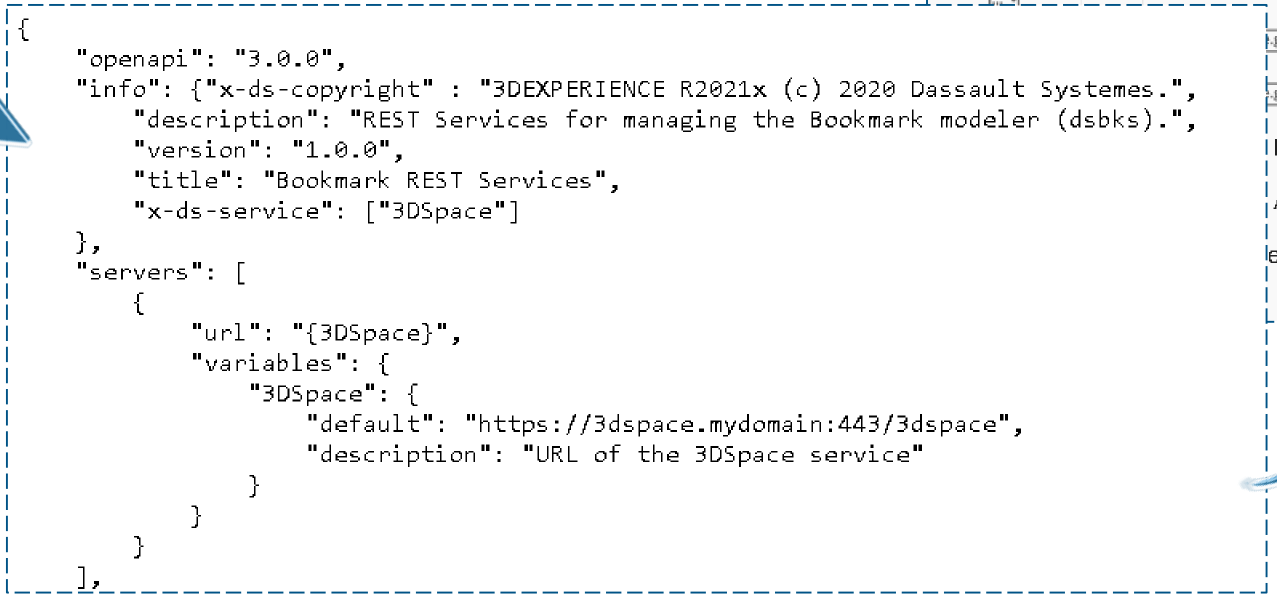
- Since Developer Assistance documentation for **3D**EXPERIENCE platform web-services implements OAS3, such documentation can be downloaded in JSON format.
  - The downloaded JSON file can then be uploaded in tools like *Postman* for easy and quick testing of the web-services.
- At the end of every Developer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



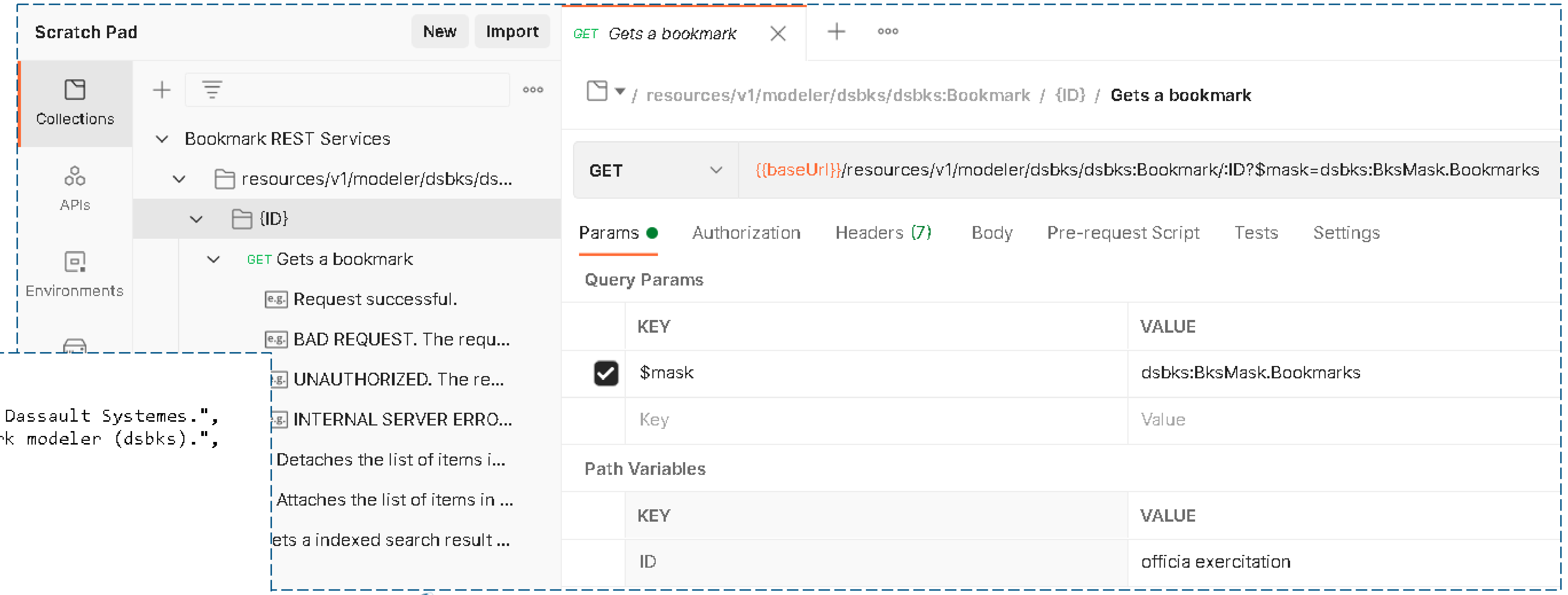
dsmveno_DefaultExtensionMask.public ›

dsmveno_DefaultExtensionMask.public_WithCEStamp ›

View BookmarkAPI1_v1.openapi.json



```
{
    "openapi": "3.0.0",
    "info": {"x-ds-copyright" : "3DEXPERIENCE R2021x (c) 2020 Dassault Systemes.",
        "description": "REST Services for managing the Bookmark modeler (dsbks).",
        "version": "1.0.0",
        "title": "Bookmark REST Services",
        "x-ds-service": ["3DSpace"]
    },
    "servers": [
        {
            "url": "{3DSpace}",
            "variables": {
                "3DSpace": {
                    "default": "https://3dspace.mydomain:443/3dspace",
                    "description": "URL of the 3DSpace service"
                }
            }
        }
    ],
```



/ resources/v1/modeler/dsbks/dsbks:Bookmark / {ID} / Gets a bookmark

GET {{baseUrl}}/resources/v1/modeler/dsbks/dsbks:Bookmark/:ID?$mask=dsbks:BksMask.Bookmarks

Query Params

	KEY	VALUE
✓	$mask	dsbks:BksMask.Bookmarks
	Key	Value

Path Variables

KEY	VALUE
ID	officia exercitation

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

POSTMAN

RESTClient

Here are the topics to be covered:

- ✓ 1. *Consuming Web-Services in **3D**EXPERIENCE platform*
- ✓ 2. *Documentation for **3D**EXPERIENCE platform Web-Services*
- **3. Tools for Testing Web-Services**
- 4. 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

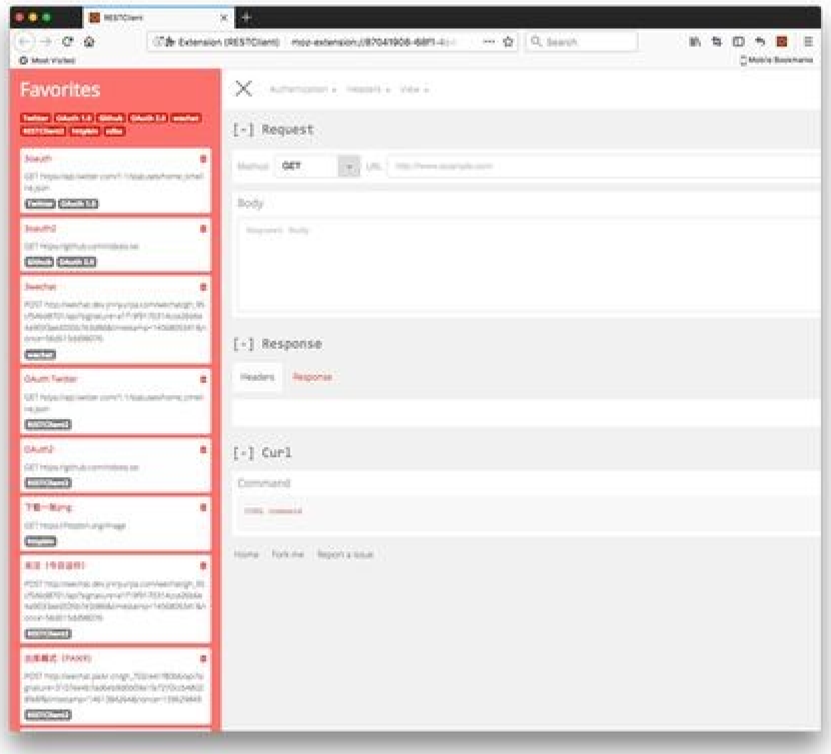
- It is to be noted that the above testing tools are external, industry standard tools.

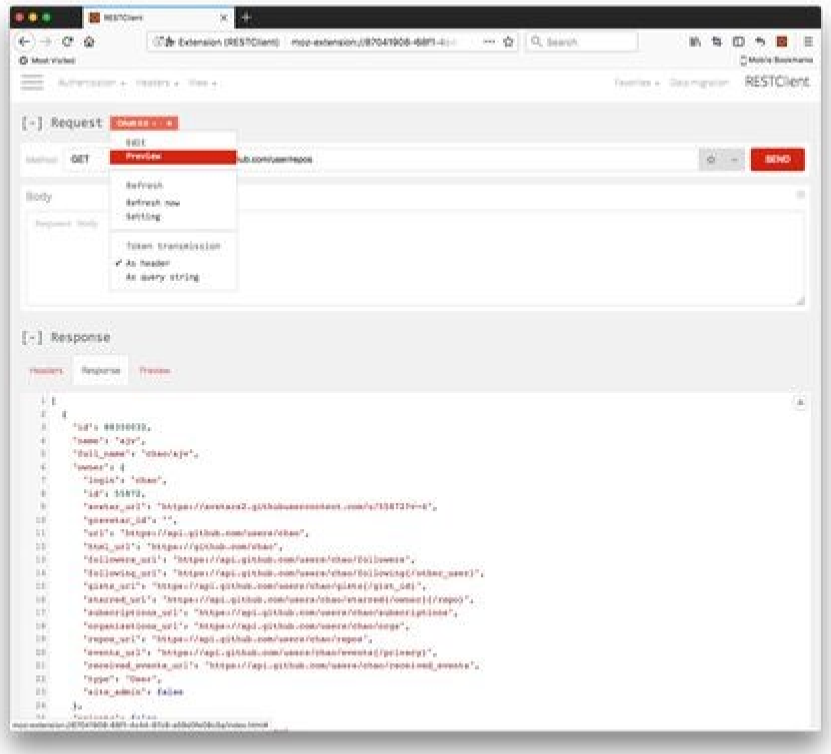
  - Since these tools are external to **3D**EXPERIENCE platform, they cannot provide authentication for web-services which is needed for testing the developed APIs.

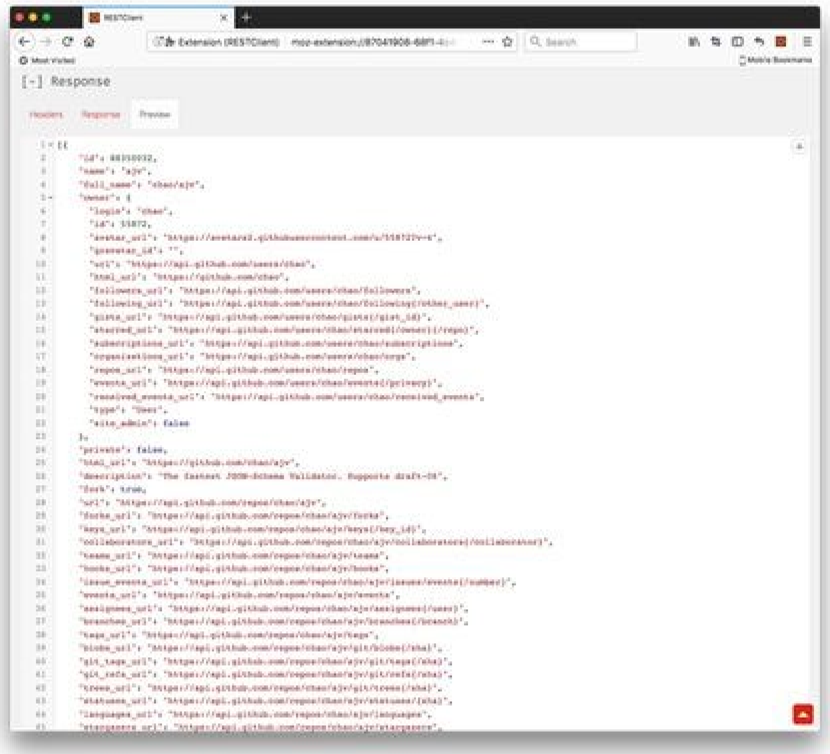
Test Web-Services – RESTClient

- **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 sa me directly against the server.

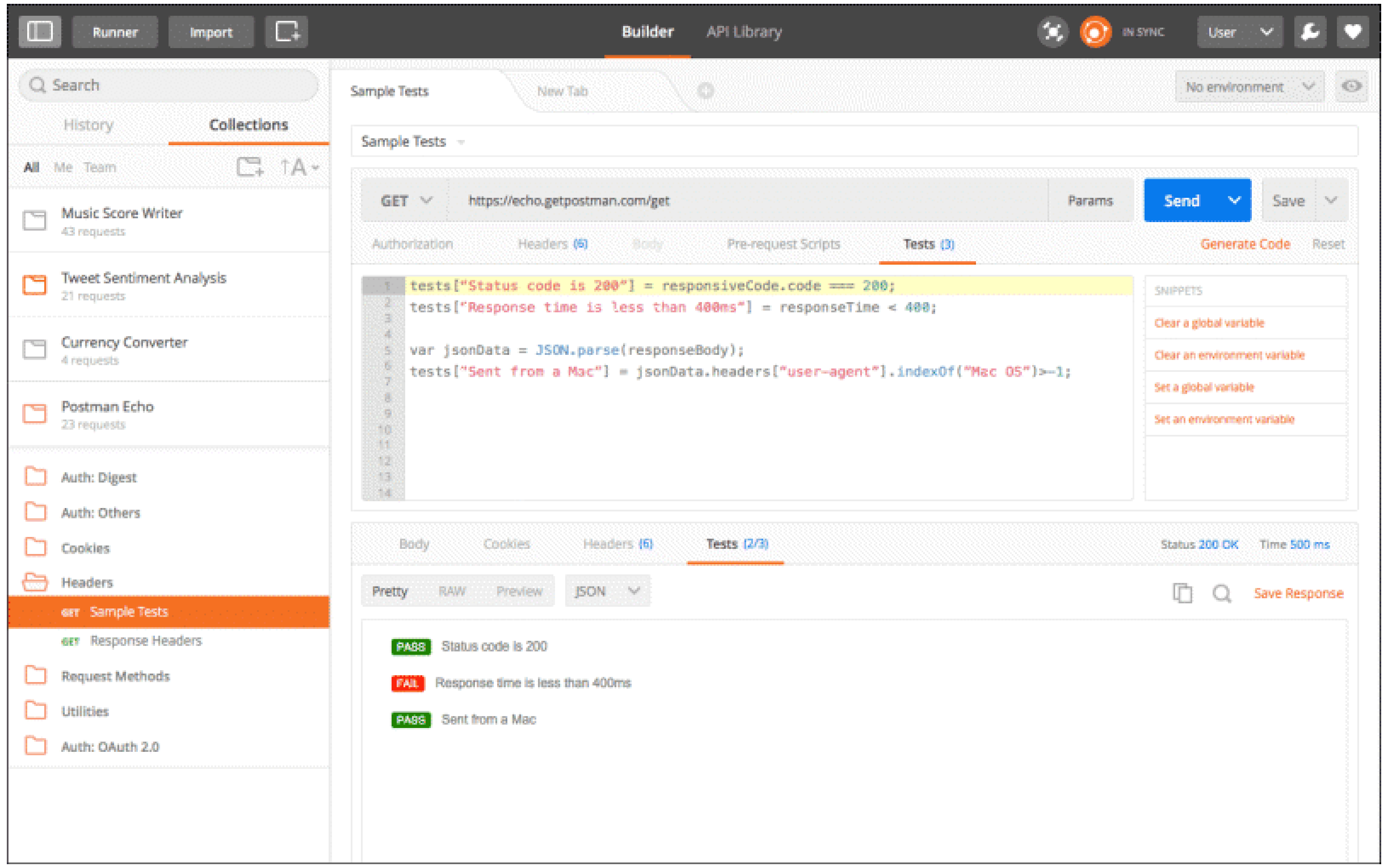






Test Web-Services – Postman

- **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.



```
tests["Status code is 200"] = responsiveCode.code === 200;
tests["Response time is less than 400ms"] = responseTime < 400;

var jsonData = JSON.parse(responseBody);
tests["Sent from a Mac"] = jsonData.headers["user-agent"].indexOf("Mac OS")>-1;
```

```
PASS Status code is 200
FAIL Response time is less than 400ms
PASS Sent from a Mac
```

Demonstration with Engineering Web-Services

In this lesson we shall demonstrate how to consume web-services, while using standard testing tools

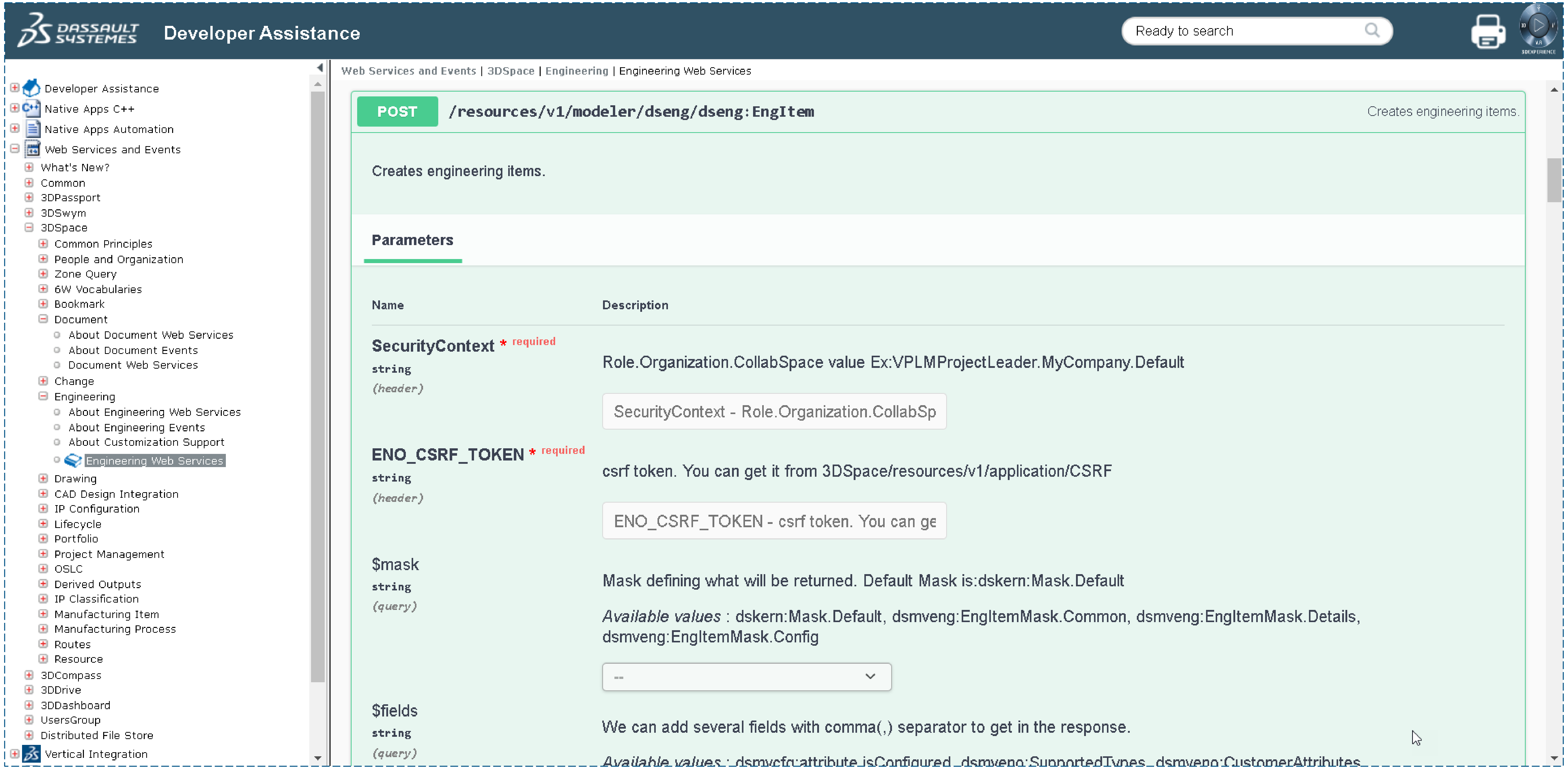
Here are the topics to be covered:

Create Engineering Item – Demonstration – Introduction

- In this demonstration, we shall create an Engineering Item (type **VPMReference**) 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:
  - 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.



Web Services and Events | 3DSpace | Engineering | Engineering Web Services

POST `/resources/v1/modeler/dseng/dseng:EngItem` Creates engineering items.

Creates engineering items.

Parameters

Name	Description
SecurityContext * required string *(header)*	Role.Organization.CollabSpace value Ex:VPLMProjectLeader.MyCompany.Default SecurityContext - Role.Organization.CollabSp
ENO_CSRF_TOKEN * required string *(header)*	csrf token. You can get it from 3DSpace/resources/v1/application/CSRF ENO_CSRF_TOKEN - csrf token. You can ge
$mask string *(query)*	Mask defining what will be returned. Default Mask is:dskern:Mask.Default *Available values* : dskern:Mask.Default, dsmveng:EngItemMask.Common, dsmveng:EngItemMask.Details, dsmveng:EngItemMask.Config --
$fields string *(query)*	We can add several fields with comma(,) separator to get in the response. *Available values* : dsmvcfg:attribute.isConfigured, dsmveng:SupportedTypes, dsmveng:CustomerAttributes

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

Continued to next page…

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:

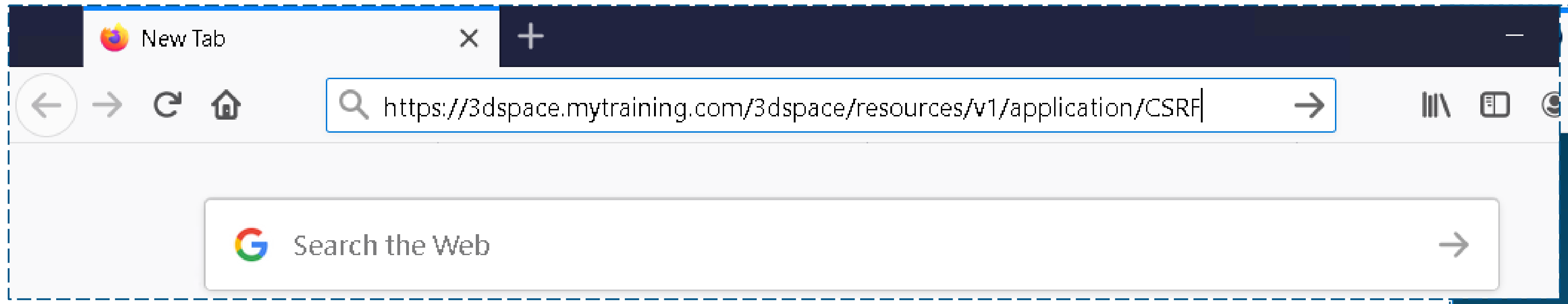
```
{
  “items”: [
    {
      “attributes”: {
        “title”: “Engineering Item Title”,
        “description”: “Engineering Item Description”,
        “dseng:EnterpriseReference”: {
          “partNumber”: “<Part Number Value>”
        },
        “dseno:EnterpriseAttributes”: {
          “<Attribute DB Name>”: “<Attribute DB Value>”
        },
        “dscfg:Configured”: {
          “enabledCriteria”: [
            “ModelVersion”,
            “Variant”
          ]
        }
      }
    }
  ]
}
```

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

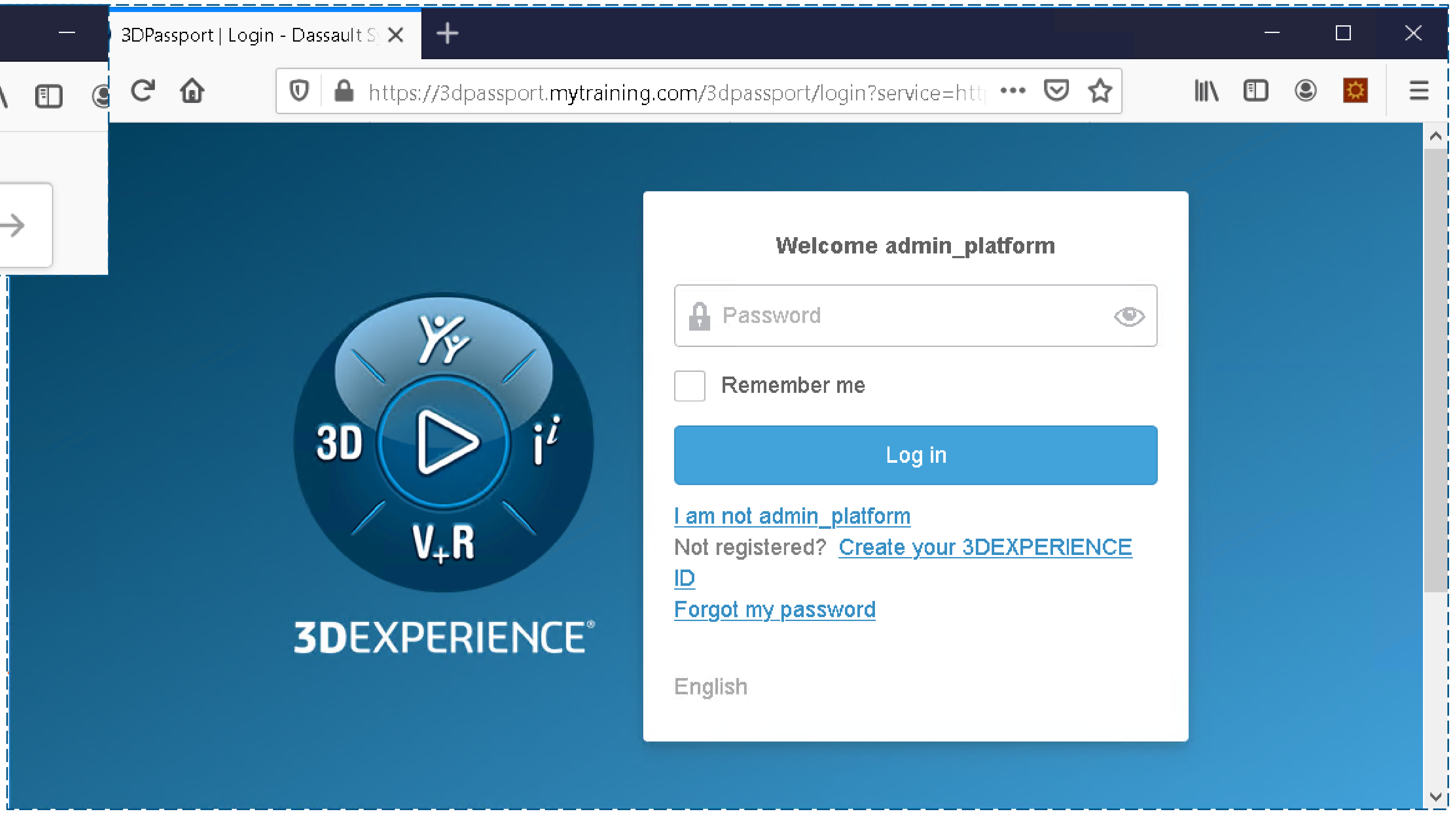
- If we analyze the parameters, we would see that we need to first obtain ENO_CSRF_TOKEN.
- As per Developer Assistance, ENO_CSRF_TOKEN can be obtained by calling the URL:

```
<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:


New Tab

https://3dspace.mytraining.com/3dspace/resources/v1/application/CSRF


3DPassport | Login - Dassault S

https://3dpassport.mytraining.com/3dpassport/login?service=htt

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)

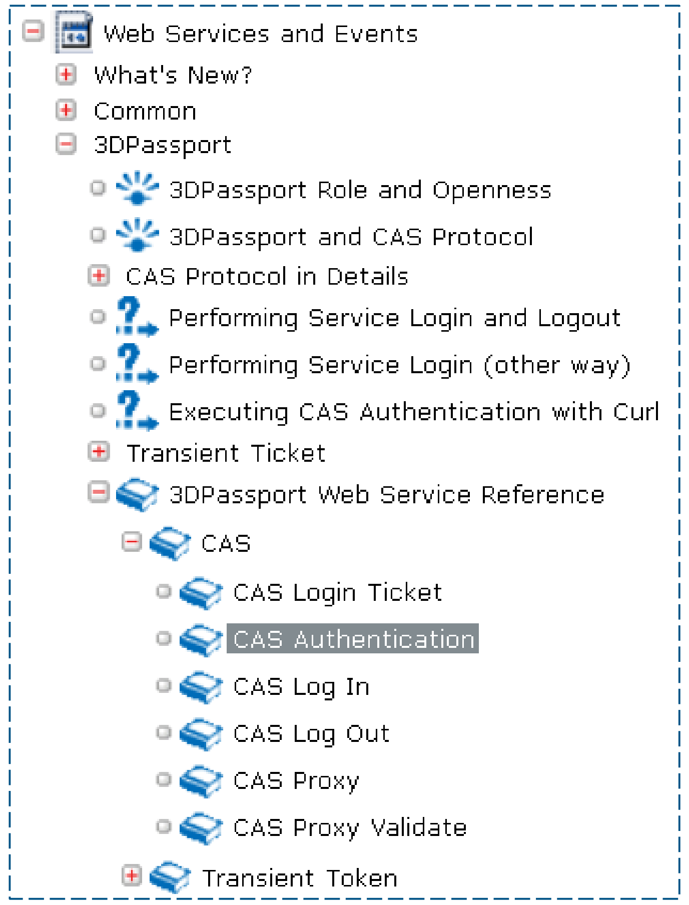
- Let us take a look at the *Developer Assistance* documentation for **3DPassport CAS Authentication** Web-Service:
  - **URL**: `<3DPassport_URL>/login`
  - **HTTP Verb**: POST
  - **Parameters**:

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.

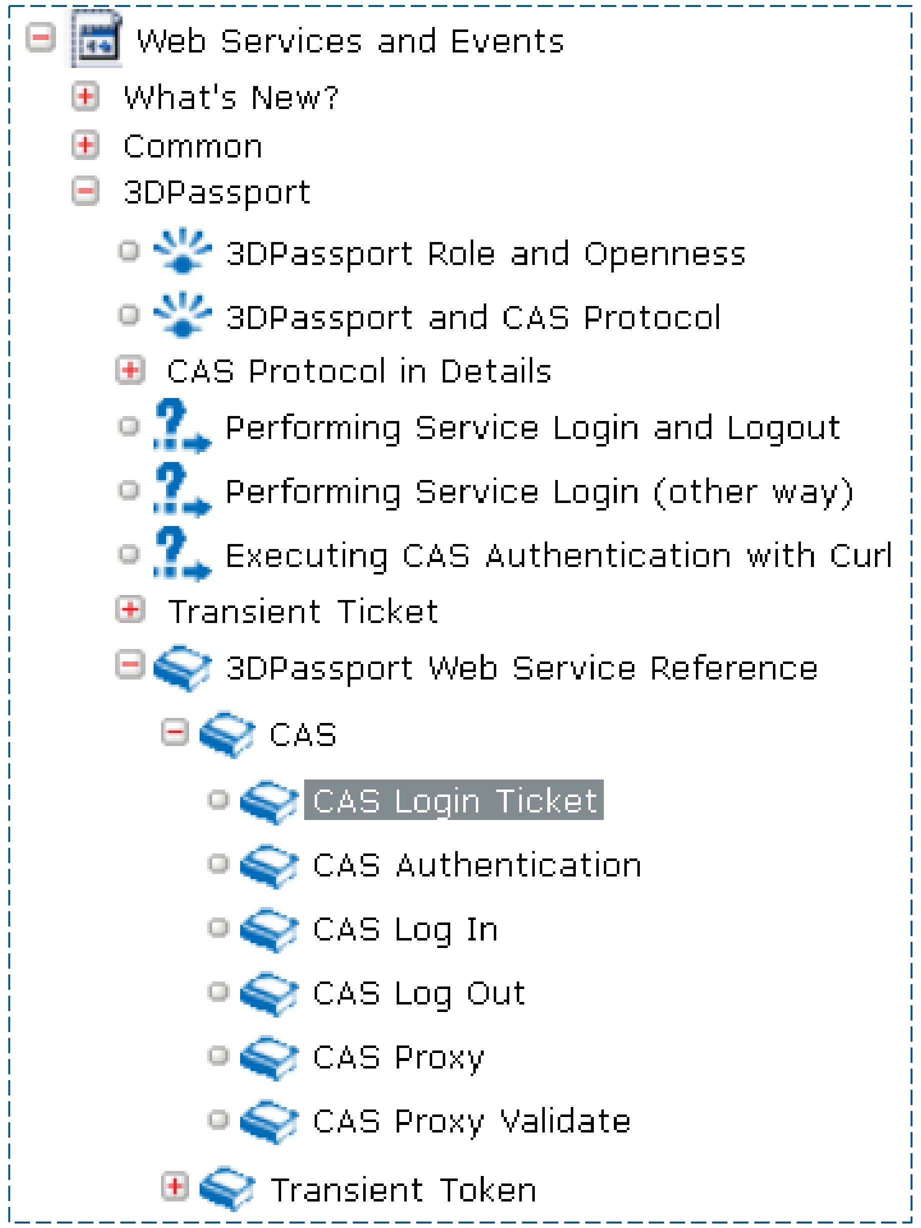
One of the mandatory parameters is "lt" or Login Ticket, which can be obtained from a separate 3DPassport web-service that returns login-tickets. This is described in the following section.


Web Services and Events
What's New?
Common
3DPassport
3DPassport Role and Openness
3DPassport and CAS Protocol
CAS Protocol in Details
Performing Service Login and Logout
Performing Service Login (other way)
Executing CAS Authentication with Curl
Transient Ticket
3DPassport Web Service Reference
CAS
CAS Login Ticket
CAS Authentication
CAS Log In
CAS Log Out
CAS Proxy
CAS Proxy Validate
Transient Token

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`



- Web Services and Events
  - What's New?
  - Common
  - 3DPassport
    - 3DPassport Role and Openness
    - 3DPassport and CAS Protocol
    - CAS Protocol in Details
    - Performing Service Login and Logout
    - Performing Service Login (other way)
    - Executing CAS Authentication with Curl
    - Transient Ticket
    - 3DPassport Web Service Reference
      - CAS
        - CAS Login Ticket
        - CAS Authentication
        - CAS Log In
        - CAS Log Out
        - CAS Proxy
        - CAS Proxy Validate
      - Transient Token

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

1. In the following exercise, we shall call OOTB (Out-Of-The-Box) **3D**EXPERIENCE platform Web-Services, as previously mentioned, to create Engineering Items.

2. A similar procedure may be adopted to use other Web-Services as well.

3. Before executing any **3D**EXPERIENCE 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

1. In this exercise, we shall demonstrate the manual creation of an *Engineering Item* by 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.

1.5 Hours

Create Engineering Item – Demonstration – Steps (1/14)

Setup Postman Environment:

1. Open Postman. Navigate as follows:

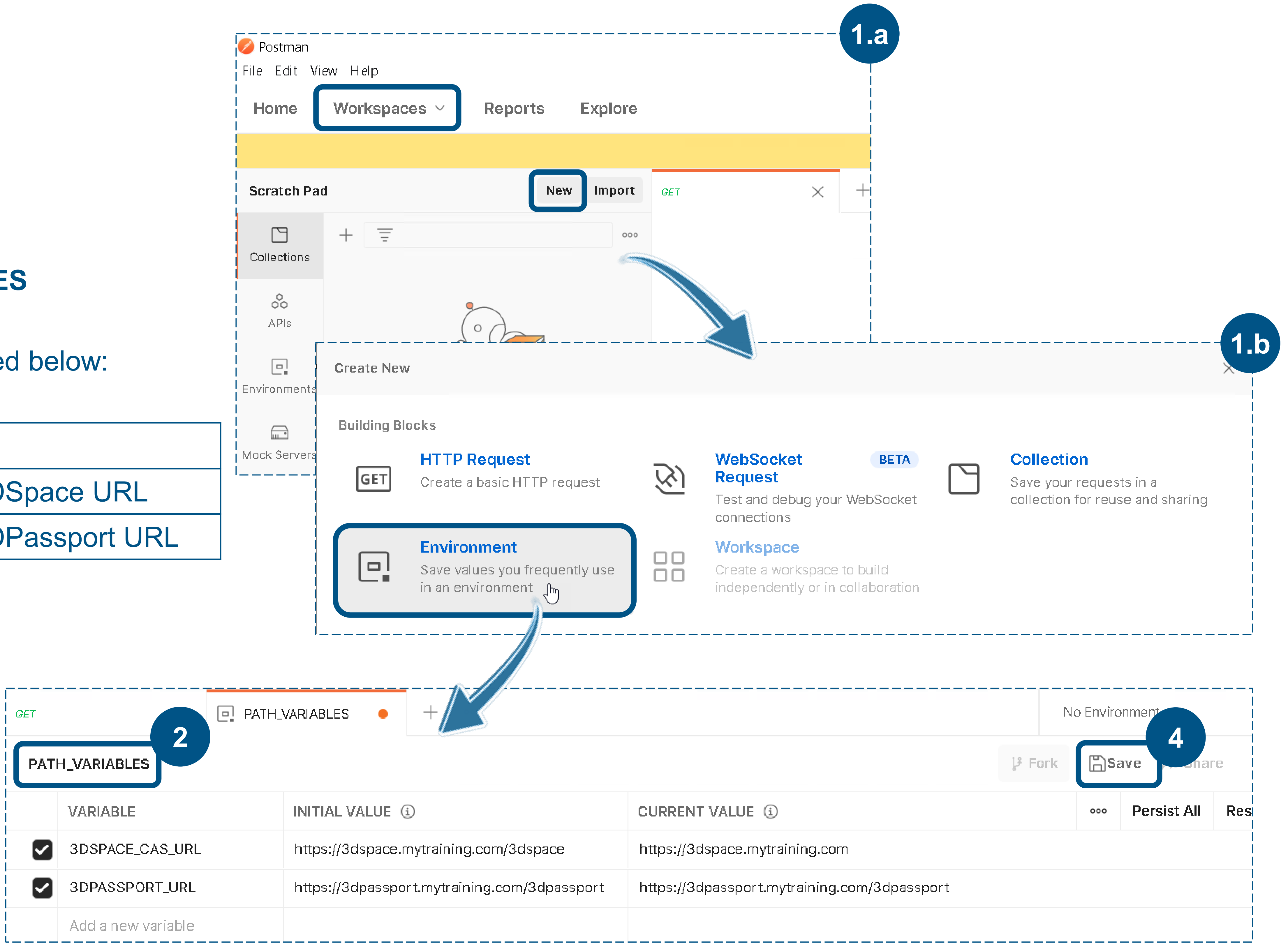
   Workspaces tab → New → Environment

2. Rename the environment to **PATH_VARIABLES**

3. Create two environment variables as mentioned below:

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

4. Click on **Save** button.

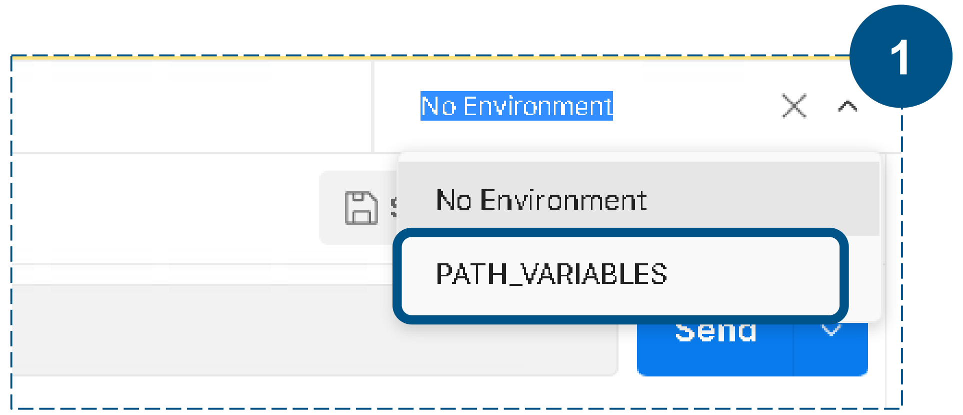


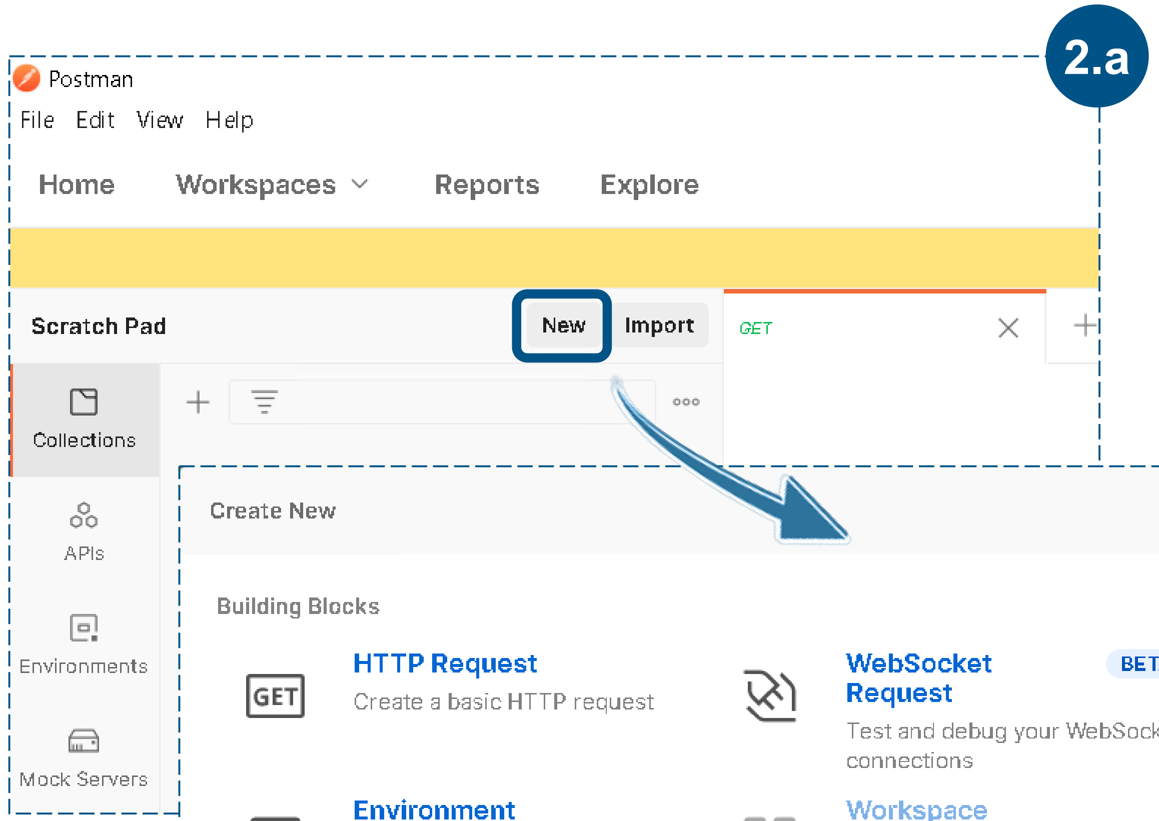
	VARIABLE	INITIAL VALUE	CURRENT VALUE
✓	3DSPACE_CAS_URL	https://3dspace.mytraining.com/3dspace	https://3dspace.mytraining.com
✓	3DPASSPORT_URL	https://3dpassport.mytraining.com/3dpassport	https://3dpassport.mytraining.com/3dpassport
	Add a new variable		

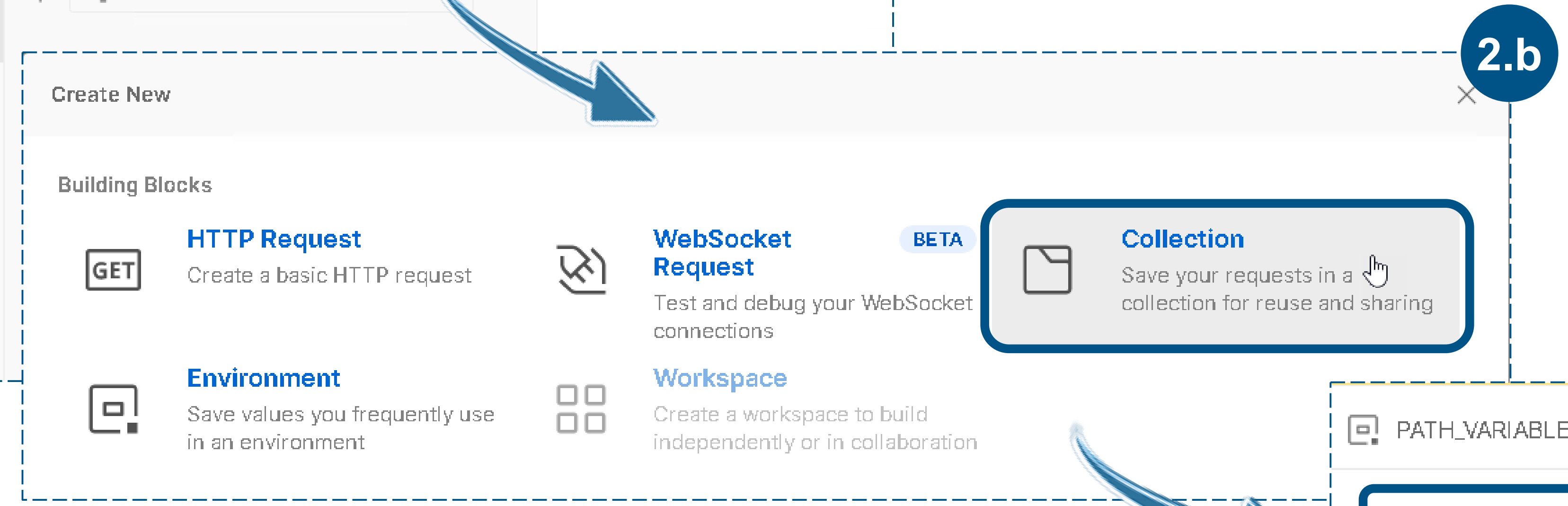
Create Engineering Item – Demonstration – Steps (2/14)

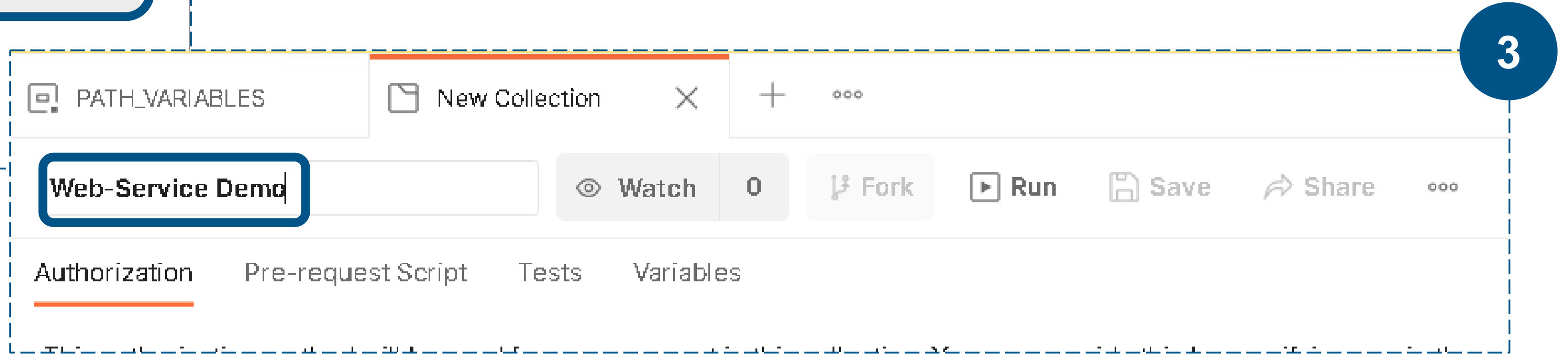
Create New Collection:

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






2.b
Create New
Building Blocks
HTTP Request
Create a basic HTTP request
WebSocket Request BETA
Test and debug your WebSocket connections
Collection
Save your requests in a collection for reuse and sharing
Environment
Save values you frequently use in an environment
Workspace
Create a workspace to build independently or in collaboration



Create Engineering Item – Demonstration – Steps (3/14)

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*.

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:

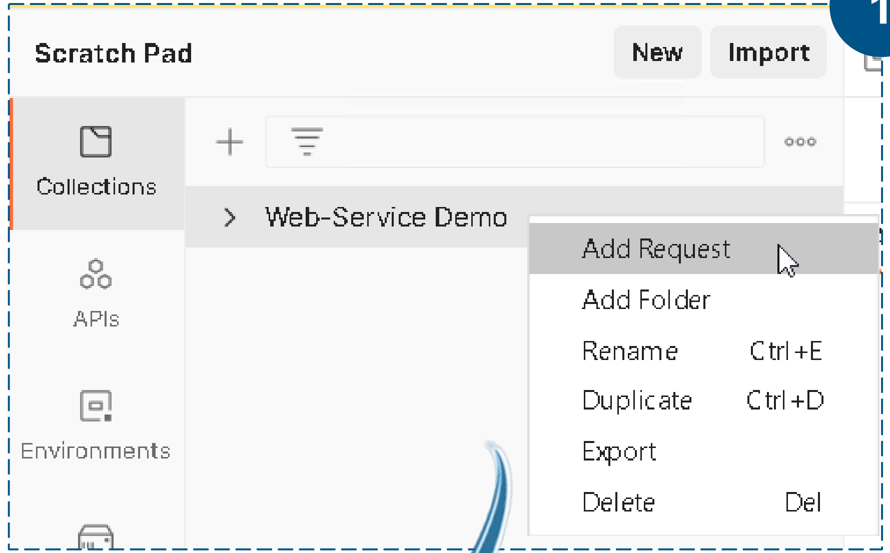
   ```
   {{3DPASSPORT_URL}}/login
   ```

3. Under **Params** section, enter Query parameter, as shown in screenshot.

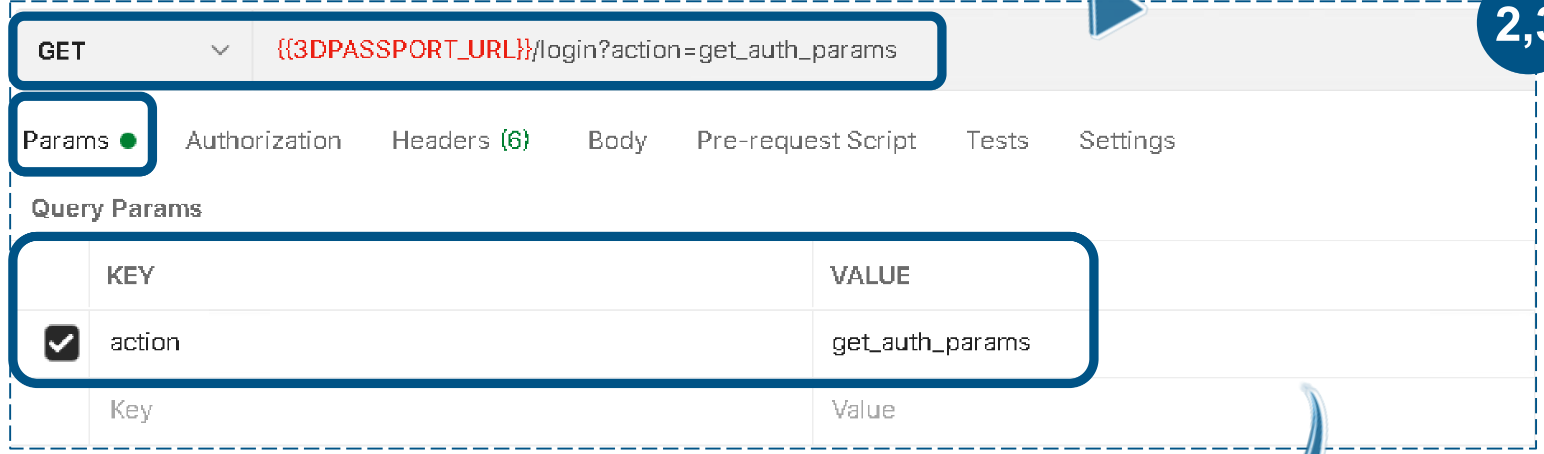
4. Under **Headers** sections, enter parameters as shown in screenshot.

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

1



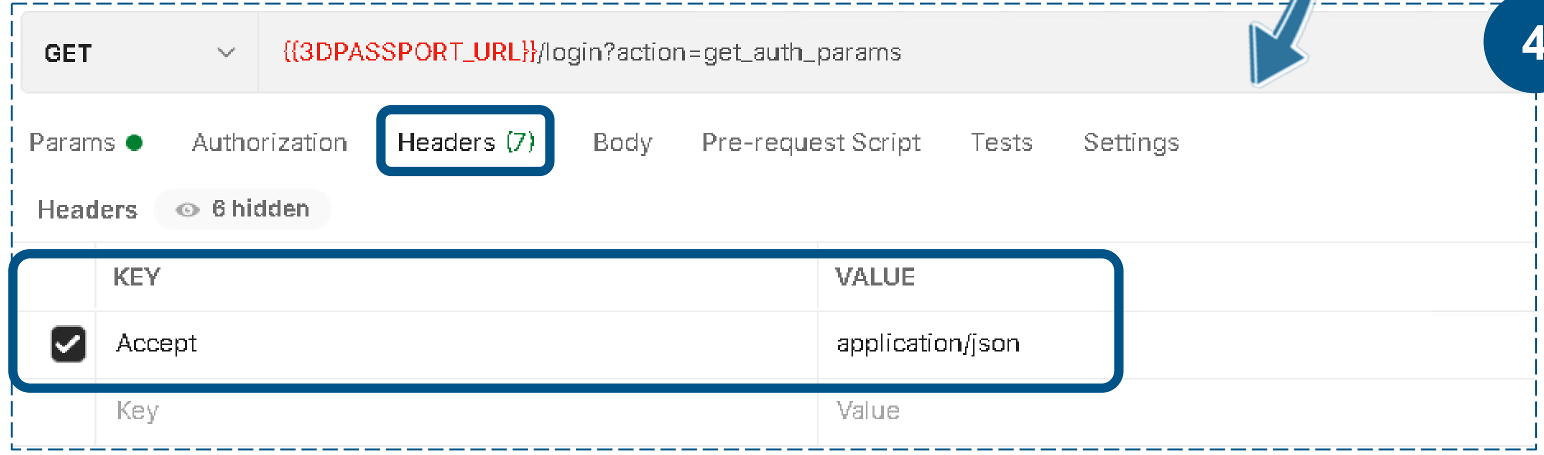
2,3



GET {{3DPASSPORT_URL}}/login?action=get_auth_params

	KEY	VALUE
☑	action	get_auth_params
	Key	Value

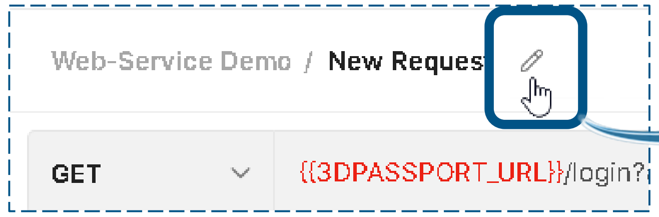
4



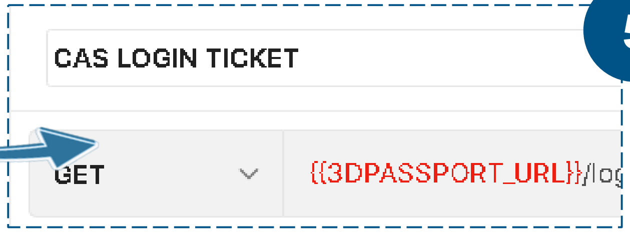
GET {{3DPASSPORT_URL}}/login?action=get_auth_params

	KEY	VALUE
☑	Accept	application/json
	Key	Value

5



GET {{3DPASSPORT_URL}}/login?


CAS LOGIN TICKET

GET {{3DPASSPORT_URL}}/log

Create Engineering Item – Demonstration – Steps (4/14)

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:

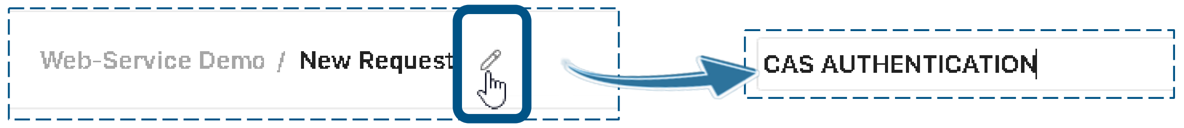
   `{{3DPASSPORT_URL}}/login`

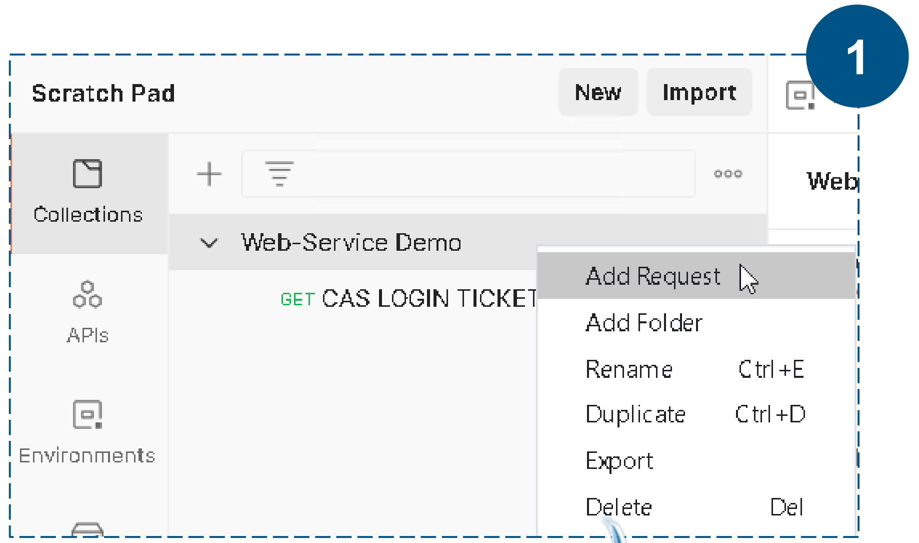
3. Under **Headers** section, enter parameter, as shown in screenshot.

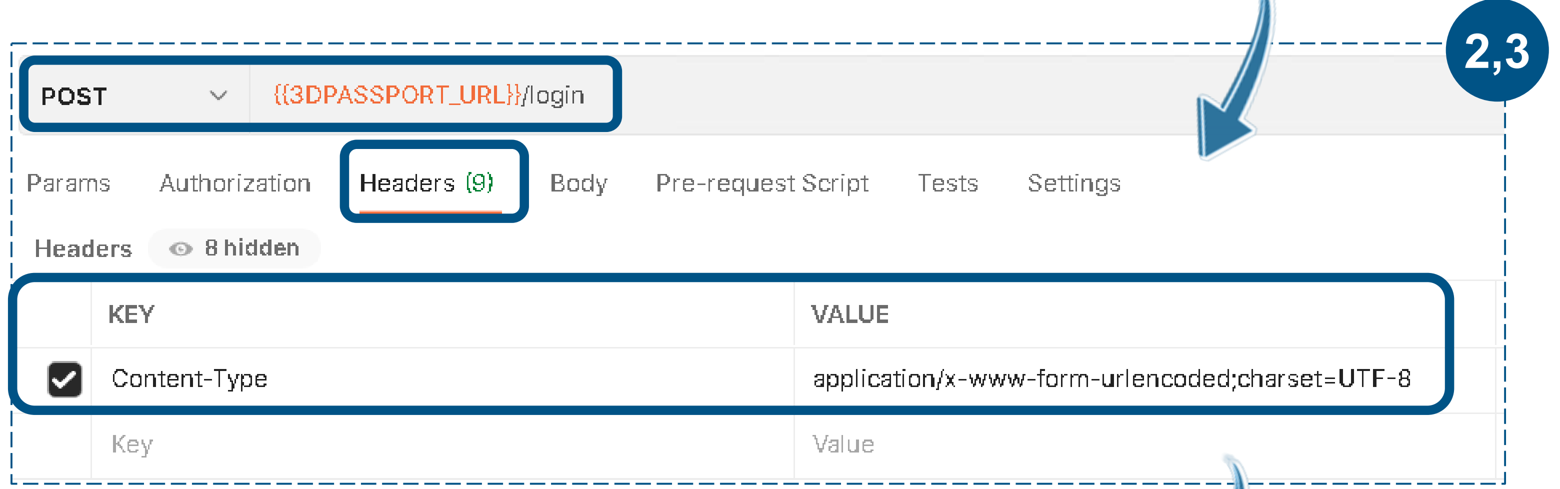
4. Under **Body** section, choose option **x-www-form-urlencoded**. Enter the following key-value pairs:

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.

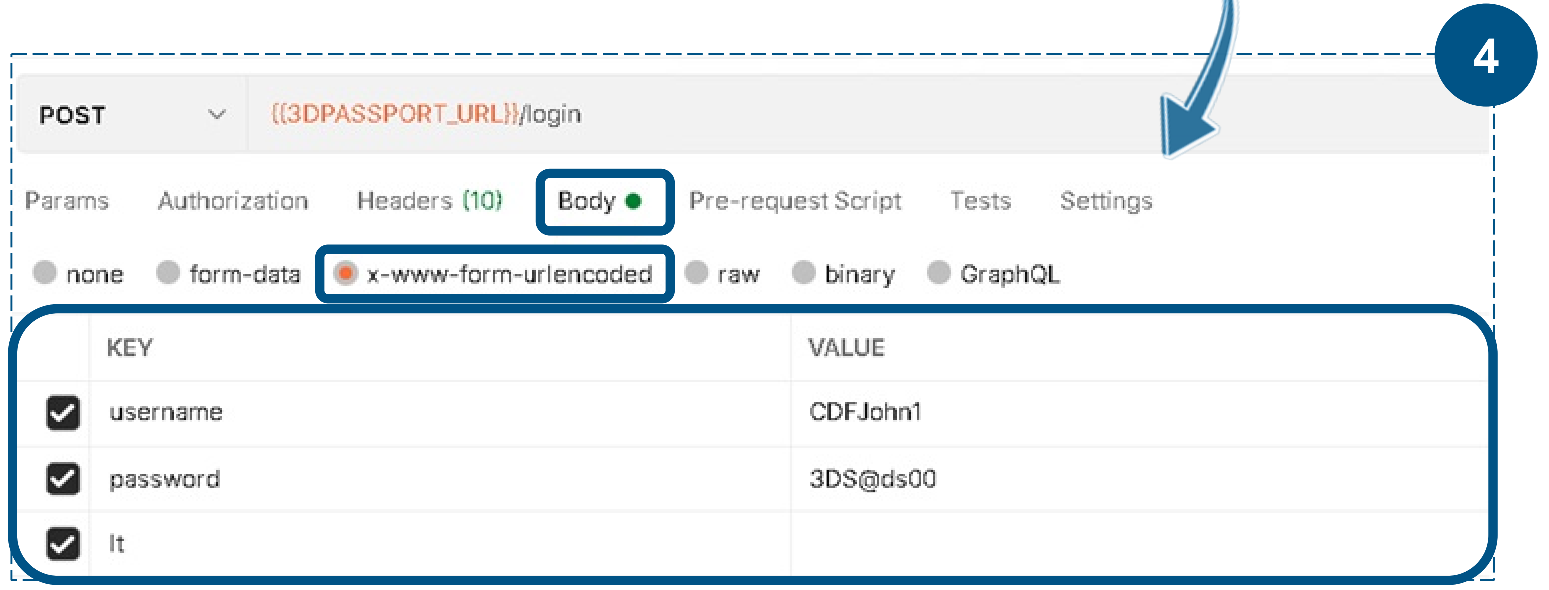






POST {{3DPASSPORT_URL}}/login

	KEY	VALUE
✓	Content-Type	application/x-www-form-urlencoded;charset=UTF-8
	Key	Value



POST {{3DPASSPORT_URL}}/login

	KEY	VALUE
✓	username	CDFJohn1
✓	password	3DS@ds00
✓	lt	

Create Engineering Item – Demonstration – Steps (5/14)

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
   ```

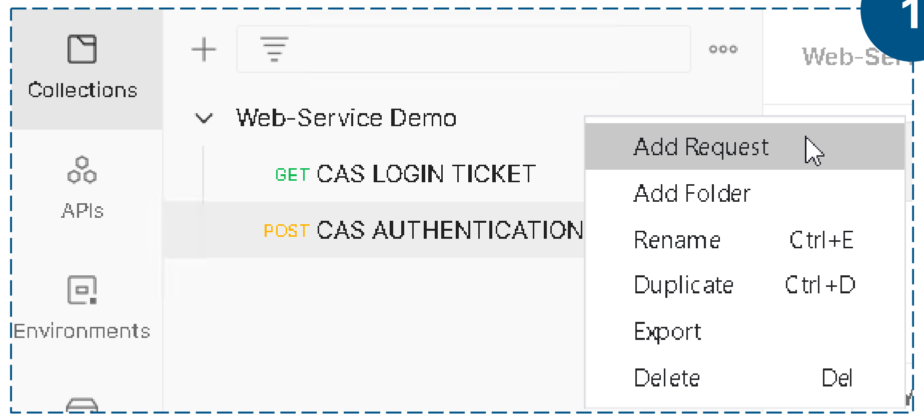
3. Rename the request to **FETCH CSRF TOKEN**, as shown, and **Save** the changes.

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

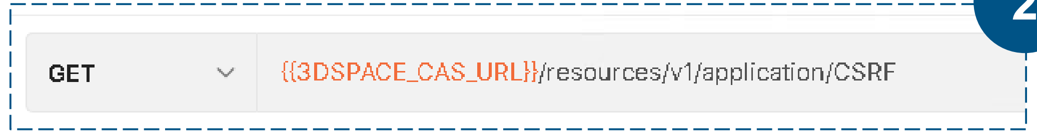
5. The **Status** should be 200, meaning a successful fetch of CSRF token.

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

1

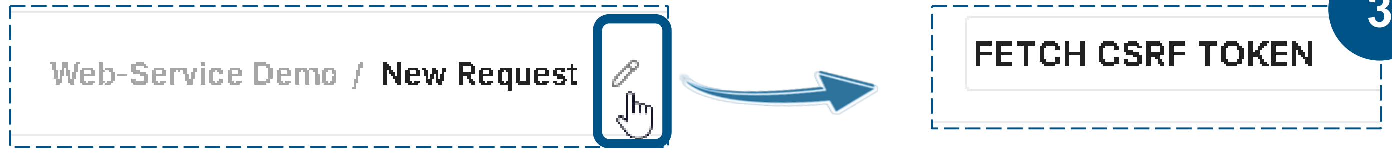


2

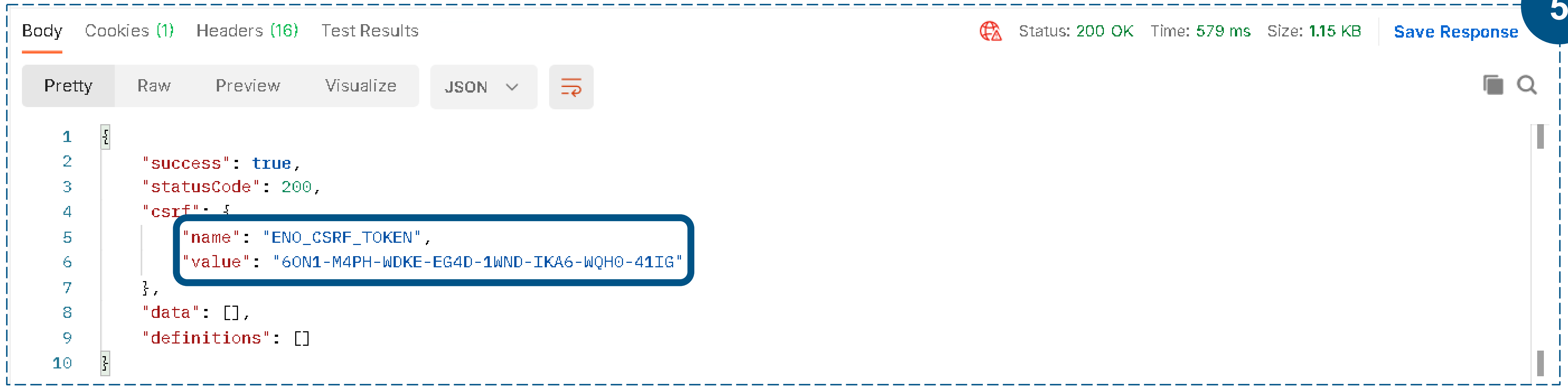


GET {{3DSPACE_CAS_URL}}/resources/v1/application/CSRF

3



5



Status: 200 OK Time: 579 ms Size: 1.15 KB

```
{
    "success": true,
    "statusCode": 200,
    "csrf": {
        "name": "ENO_CSRF_TOKEN",
        "value": "6ON1-M4PH-WDKE-EG4D-1WND-IKA6-WQHO-41IG"
    },
    "data": [],
    "definitions": []
}
```

Create Engineering Item – Demonstration – Steps (6/14)

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:

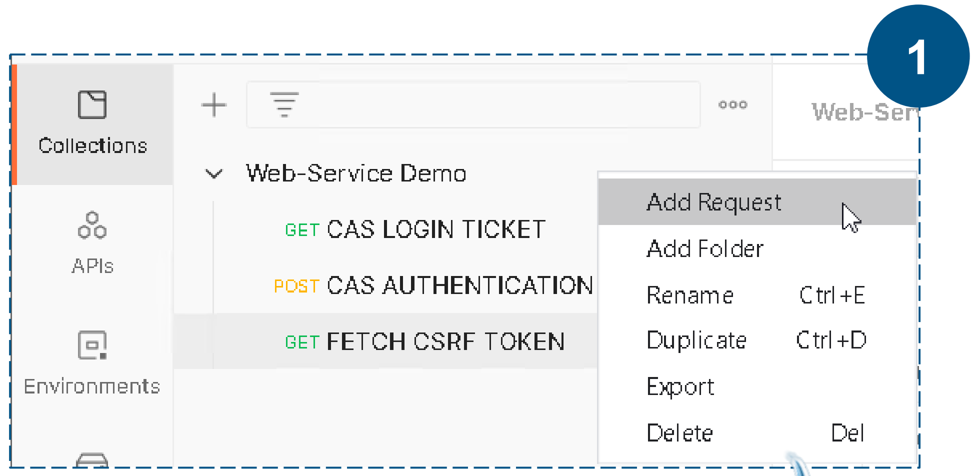
   `{{3DSPACE_CAS_URL}}/resources/v1/modeler/dseng/dseng:EngItem`

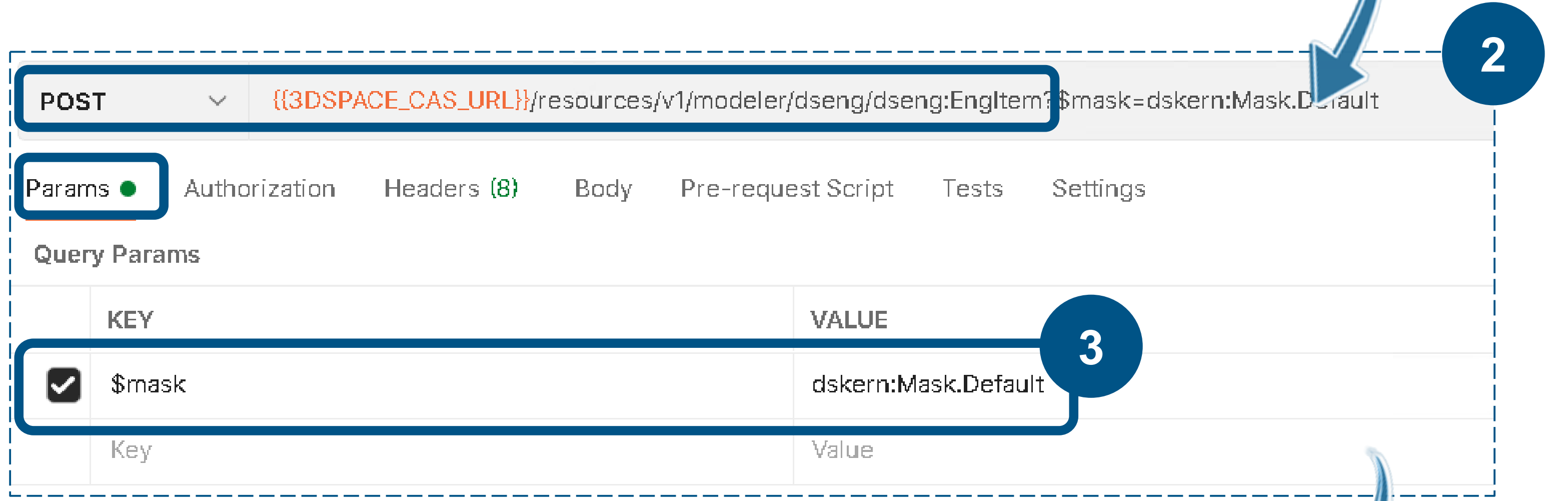
3. Under **Params** section, enter Query parameter, as mentioned below.

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.

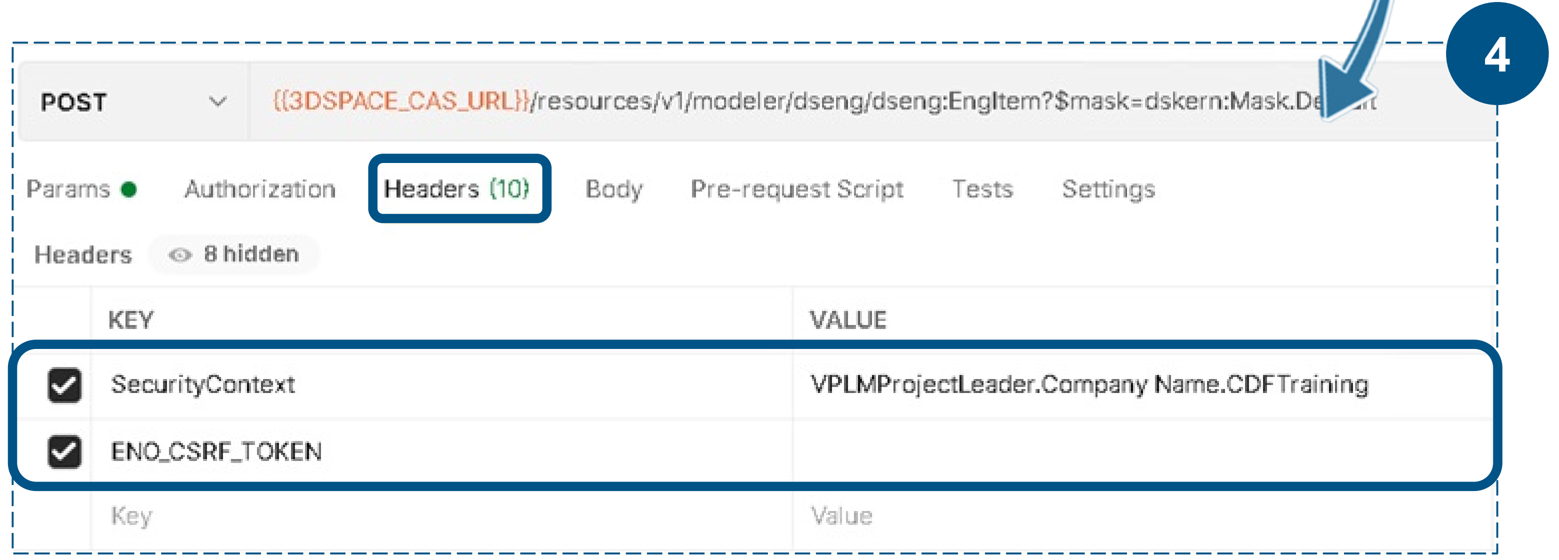
Key	Value
SecurityContext	Valid security context required to create Engineering Item
ENO_CSRF_TOKEN	*Empty*. We will copy ENO_CSRF_TOKEN from output of *FETCH CSRF TOKEN* web-service, during execution.





{{3DSPACE_CAS_URL}}/resources/v1/modeler/dseng/dseng:EngItem?$mask=dskern:Mask.Default

KEY	VALUE
$mask	dskern:Mask.Default



{{3DSPACE_CAS_URL}}/resources/v1/modeler/dseng/dseng:EngItem?$mask=dskern:Mask.De

KEY	VALUE
SecurityContext	VPLMProjectLeader.Company Name.CDFTraining
ENO_CSRF_TOKEN	

Create Engineering Item – Demonstration – Steps (7/14)

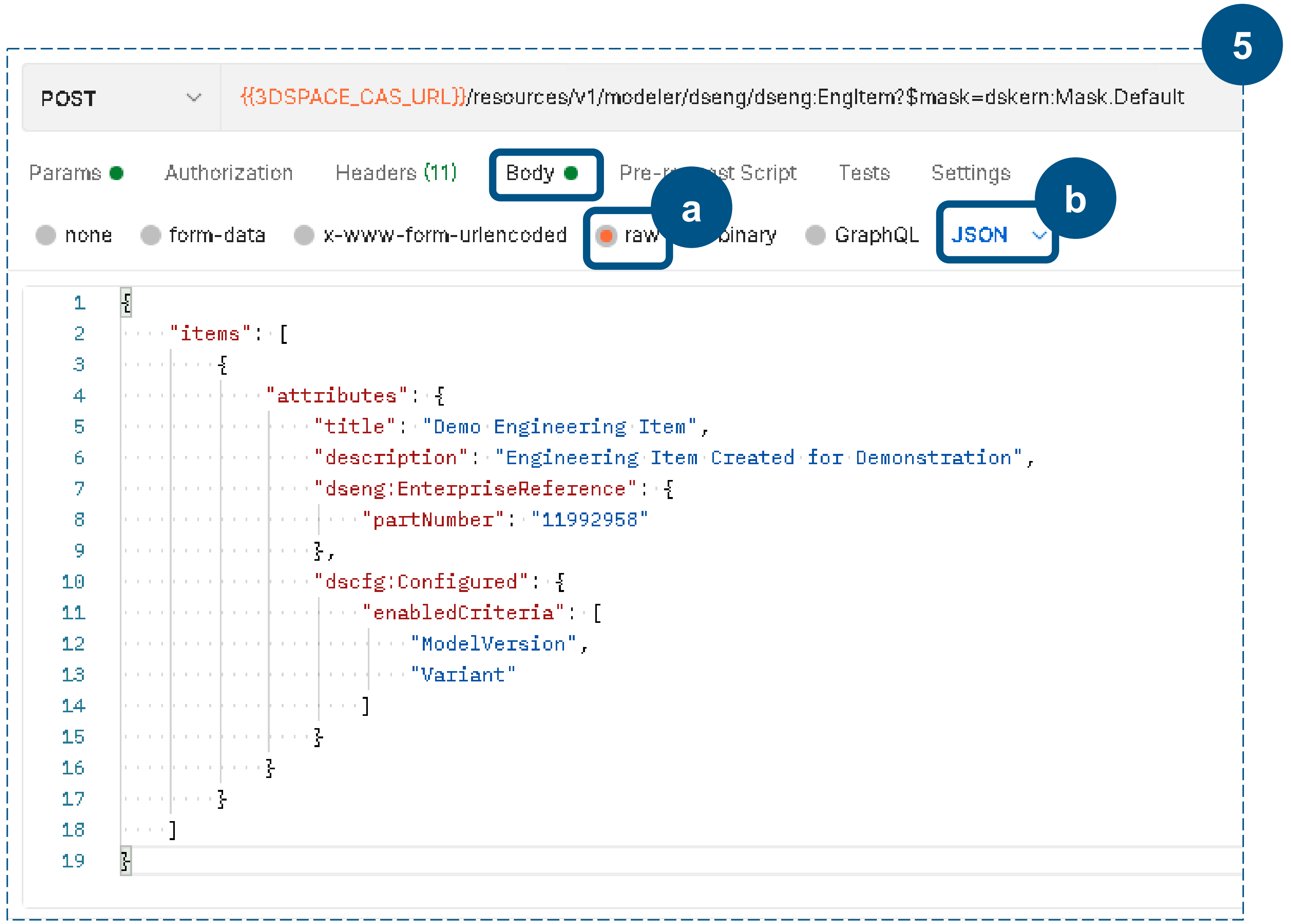
Create Request for creating Engineering Item (continued…):

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

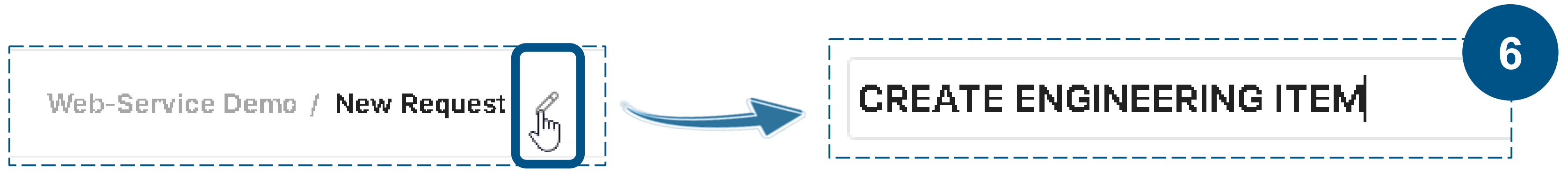
   b) Select format as **JSON**

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



```
{{3DSPACE_CAS_URL}}/resources/v1/modeler/dseng/dseng:EngItem?$mask=dskern:Mask.Default
```

```
{
    "items": [
        {
            "attributes": {
                "title": "Demo Engineering Item",
                "description": "Engineering Item Created for Demonstration",
                "dseng:EnterpriseReference": {
                    "partNumber": "11992958"
                },
                "dscfg:Configured": {
                    "enabledCriteria": [
                        "ModelVersion",
                        "Variant"
                    ]
                }
            }
        }
    ]
}
```



Create Engineering Item – Demonstration – Steps (8/14)

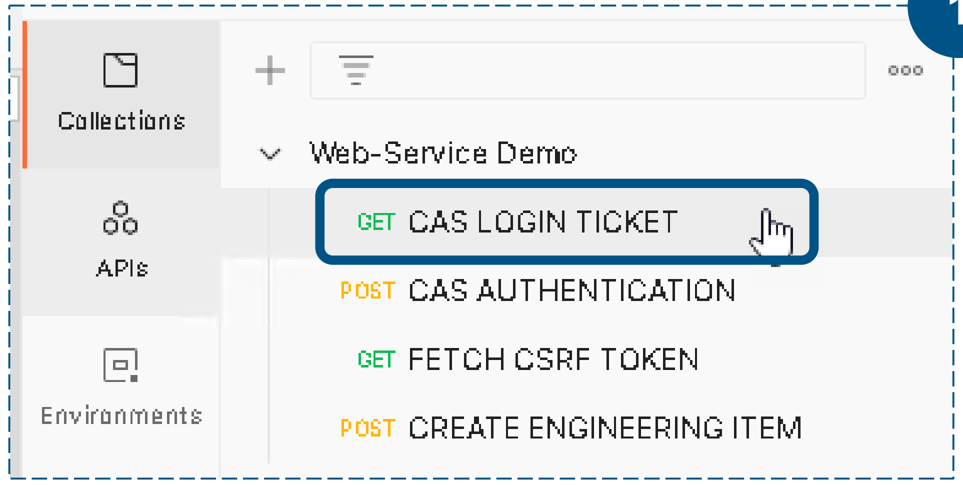
Execute Request for CAS Login Ticket:

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

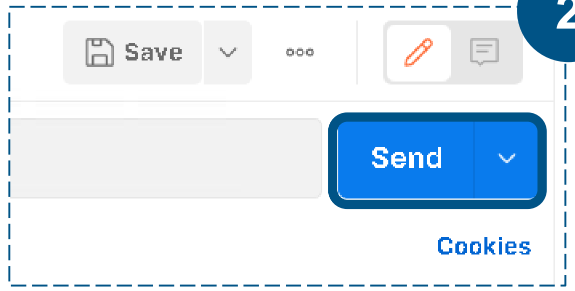
2. Click on the **Send** button, and observe the response.

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

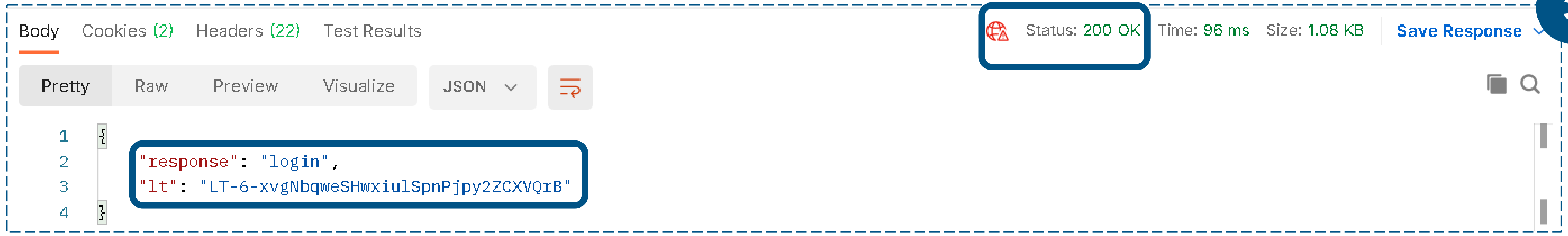
1



2



3



```
{
"response": "login",
"lt": "LT-6-xvgNbqweSHwxiulSpnPjpy2ZCXVQrB"
}
```

4. 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 (9/14)

Execute Request for CAS Authentication:

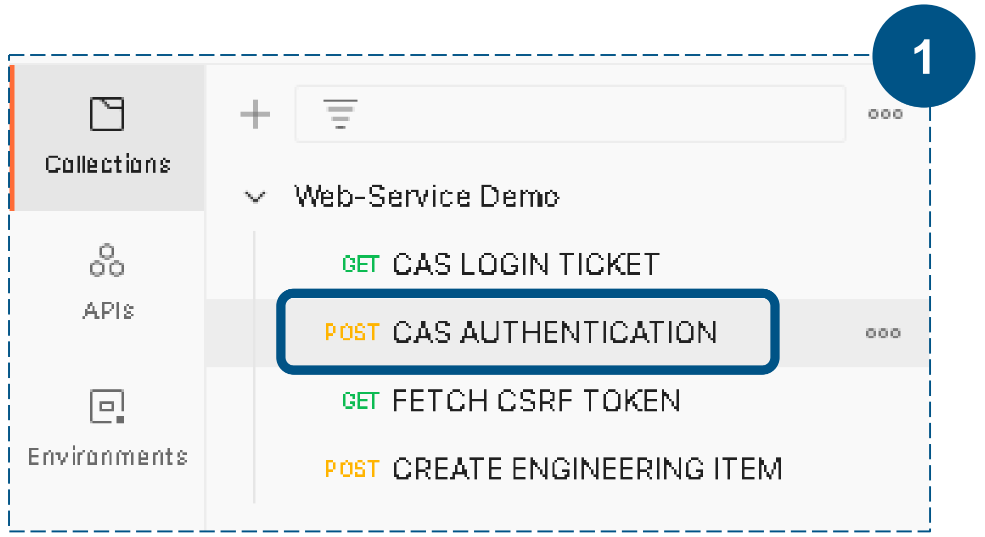
1. From the list of Web-Service requests created in *Collections* tab, open **CAS AUTHENTICATION**.

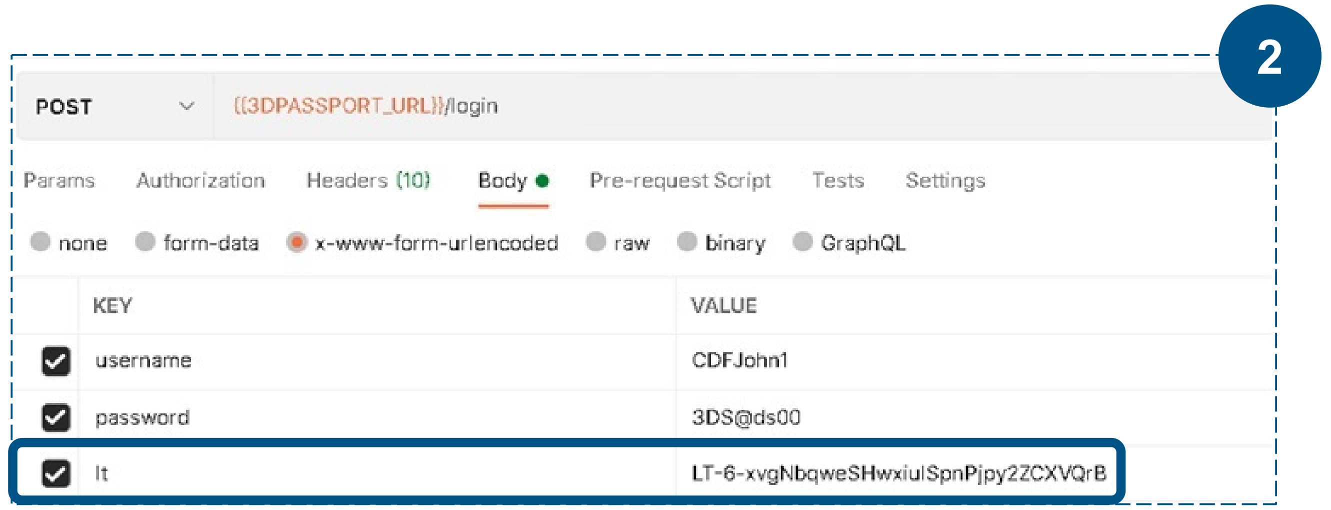
2. Under **Body** section, paste the *Login Ticket*, as copied from the output of the previously executed **CAS LOGIN TICKET** web-service, against the key named "**lt**".

3. **Save** the changes.

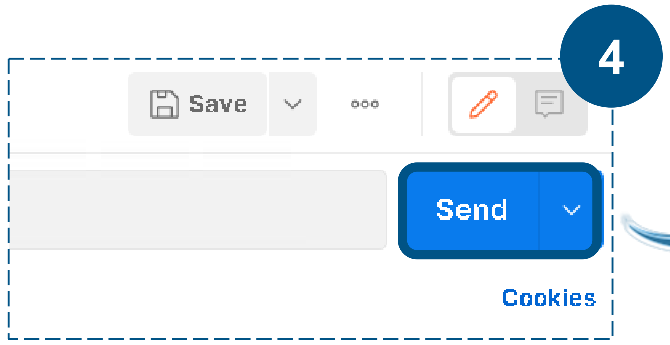
4. Click on the **Send** button, and observe the response.

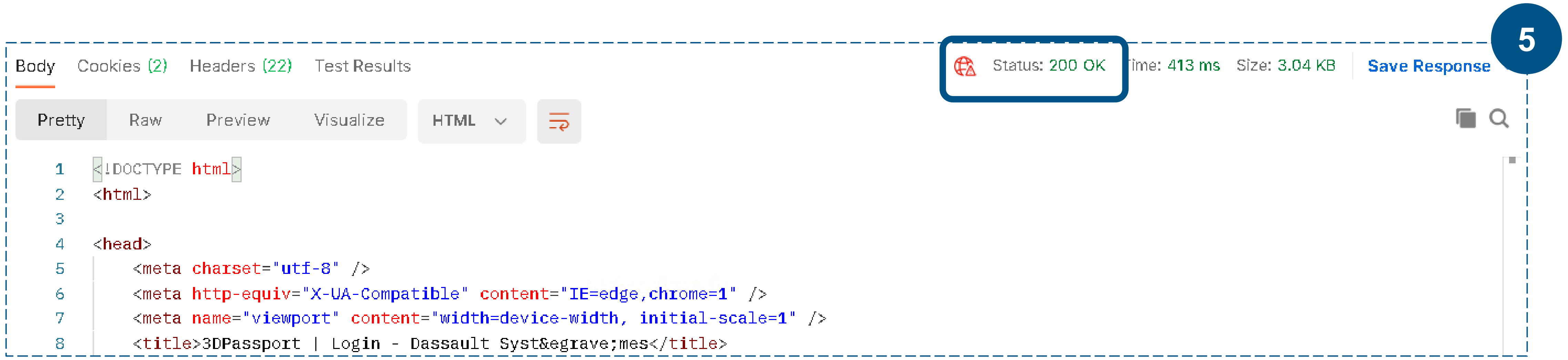
5. The **Status** should be 200, meaning a successful authentication of Username and Password.





	KEY	VALUE
✓	username	CDFJohn1
✓	password	3DS@ds00
✓	lt	LT-6-xvgNbqweSHwxiulSpnPjpy2ZCXVQrB





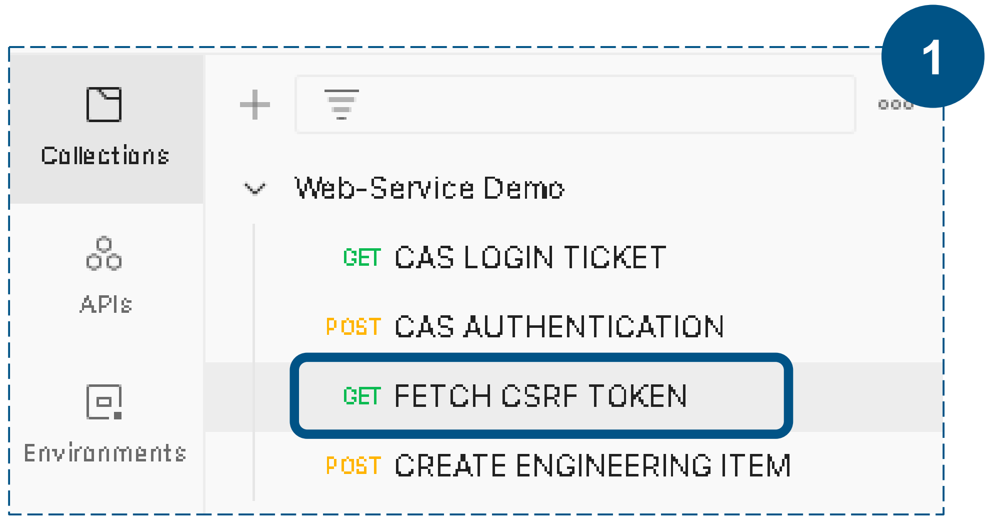
```
<!DOCTYPE html>
<html>

<head>
    <meta charset="utf-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge,chrome=1" />
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <title>3DPassport | Login - Dassault Syst&egrave;mes</title>
```

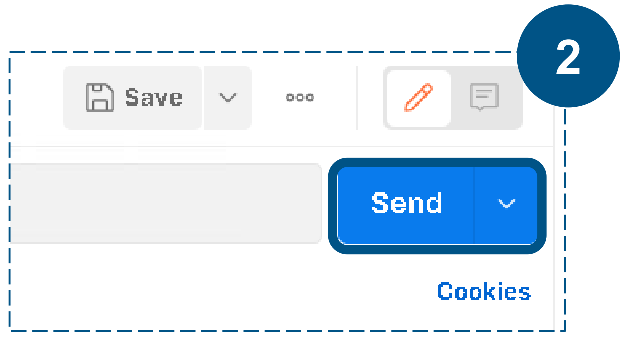
Create Engineering Item – Demonstration – Steps (10/14)

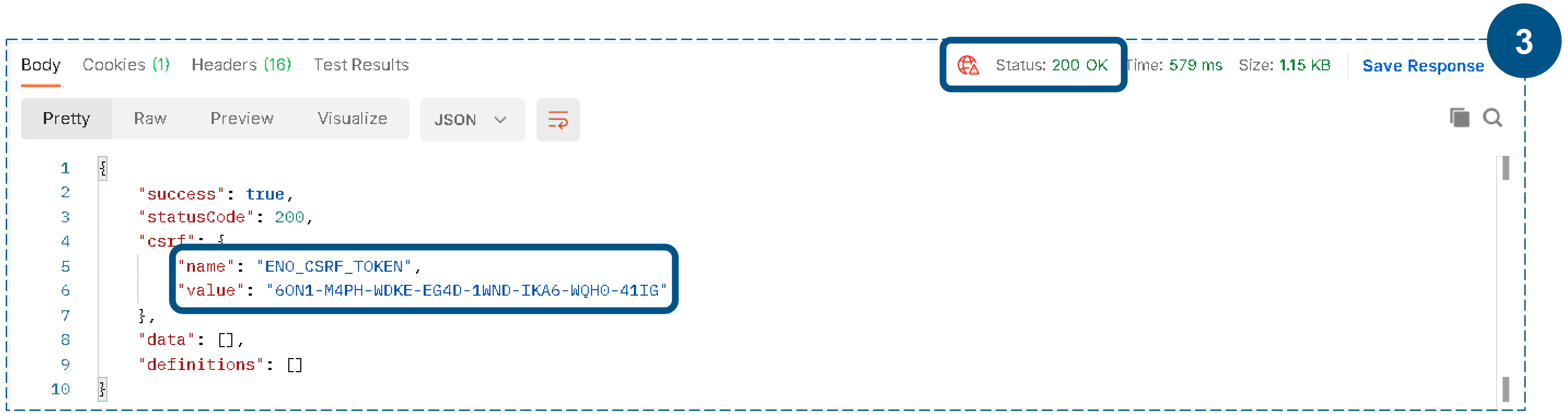
Execute Request for fetching CSRF Token:

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.







```
{
    "success": true,
    "statusCode": 200,
    "csrf": {
        "name": "ENO_CSRF_TOKEN",
        "value": "6ON1-M4PH-WDKE-EG4D-1WND-IKA6-WQH0-41IG"
    },
    "data": [],
    "definitions": []
}
```

Create Engineering Item – Demonstration – Steps (11/14)

Execute Request to Create Engineering Item:

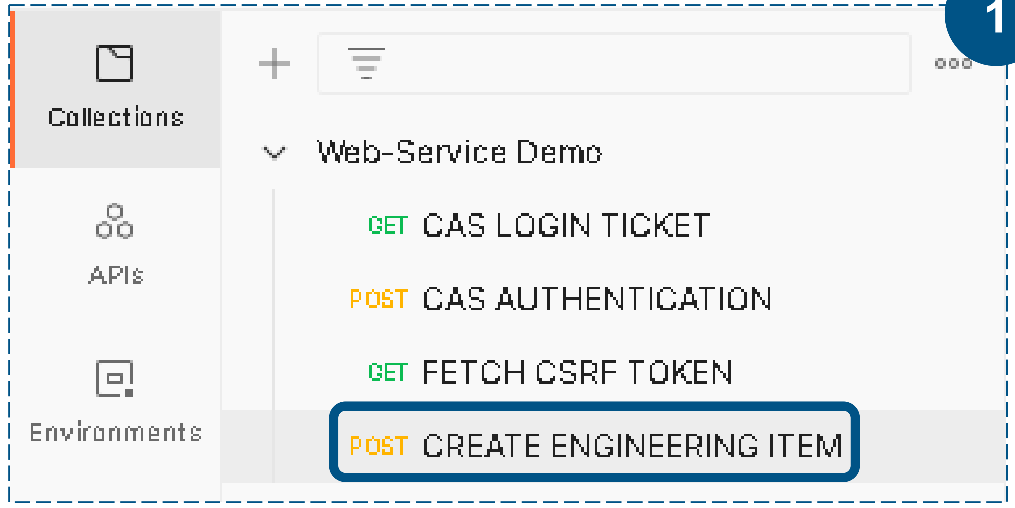
1. From the *Collections* tab, open the **CREATE ENGINEERING ITEM** web-service.

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**.

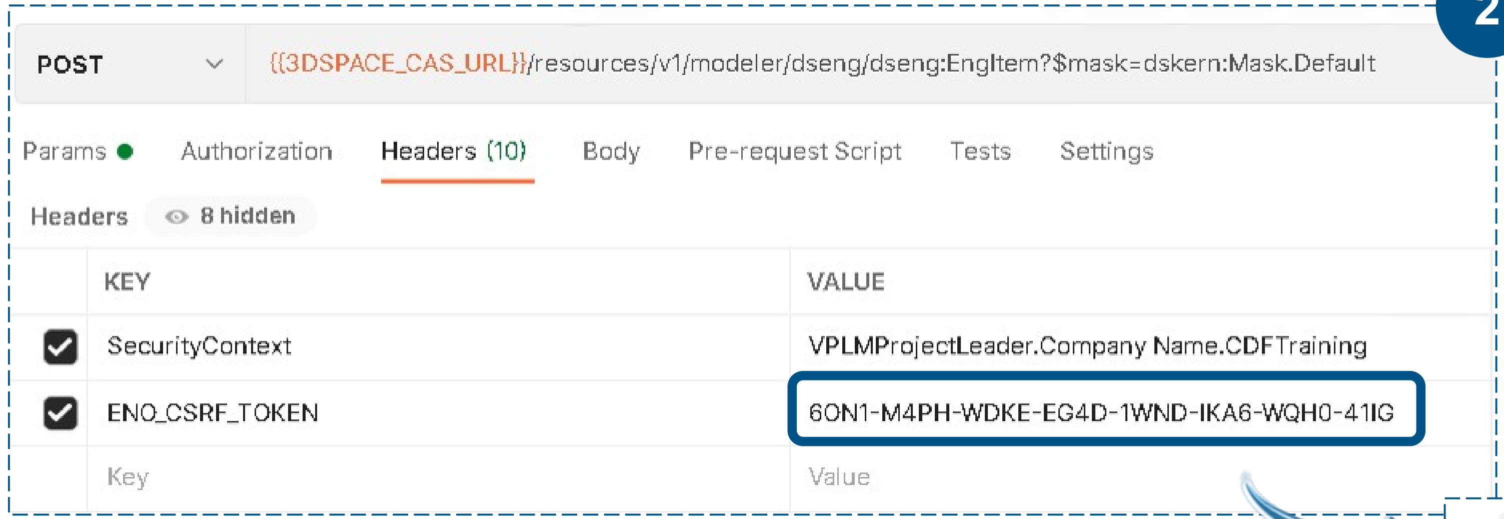
3. **Save** the changes. Click on the **Send** button, and observe the response.

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.

1



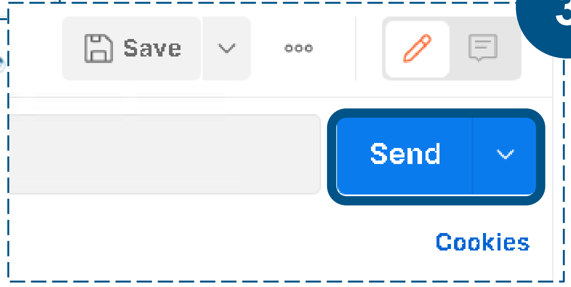
2



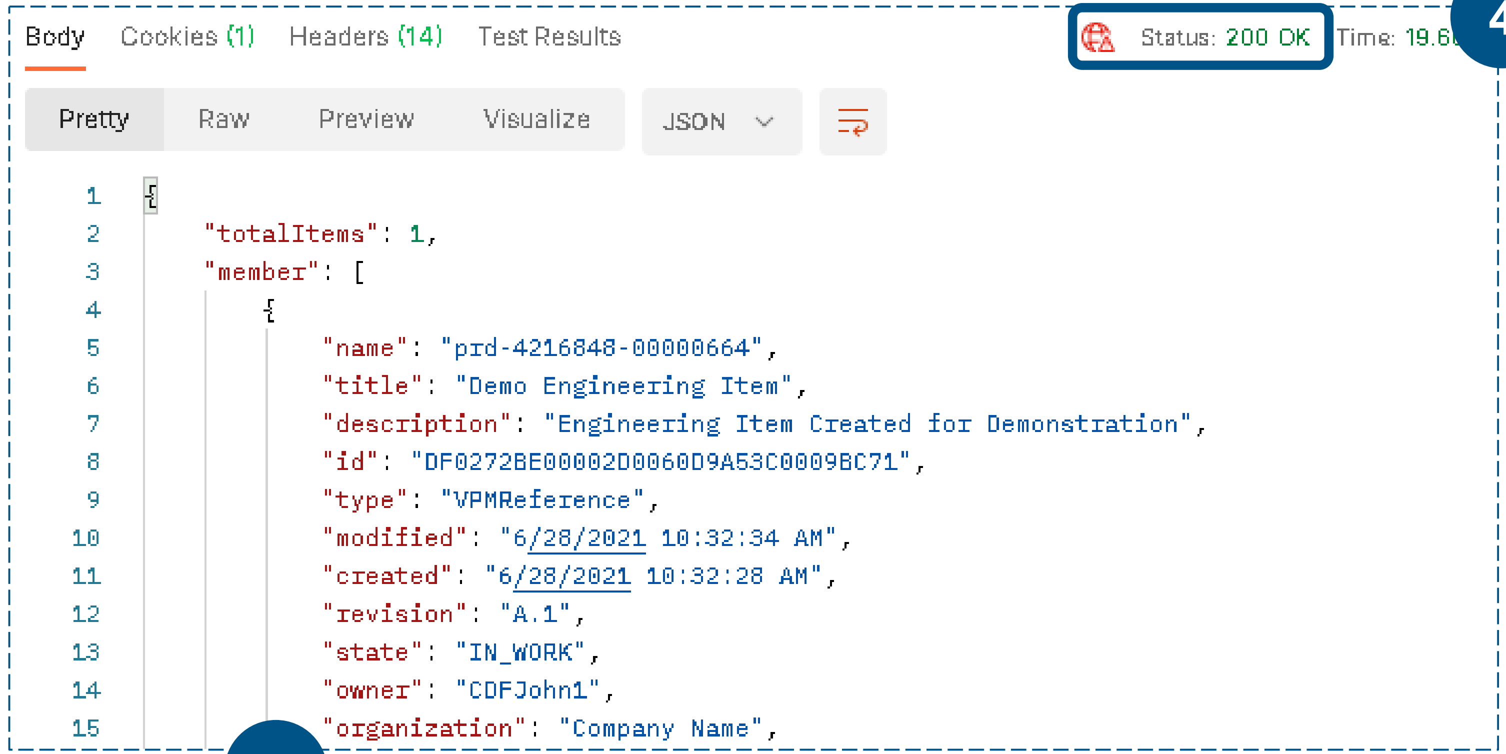
POST {{3DSPACE_CAS_URL}}/resources/v1/modeler/dseng/dseng:EngItem?$mask=dskern:Mask.Default

	KEY	VALUE
✓	SecurityContext	VPLMProjectLeader.Company Name.CDFTraining
✓	ENO_CSRF_TOKEN	6ON1-M4PH-WDKE-EG4D-1WND-IKA6-WQH0-41IG
	Key	Value

3



4



Status: 200 OK Time: 19.6

```
{
    "totalItems": 1,
    "member": [
        {
            "name": "prd-4216848-00000664",
            "title": "Demo Engineering Item",
            "description": "Engineering Item Created for Demonstration",
            "id": "DF0272BE00002D0060D9A53C0009BC71",
            "type": "VPMReference",
            "modified": "6/28/2021 10:32:34 AM",
            "created": "6/28/2021 10:32:28 AM",
            "revision": "A.1",
            "state": "IN_WORK",
            "owner": "CDFJohn1",
            "organization": "Company Name",
```

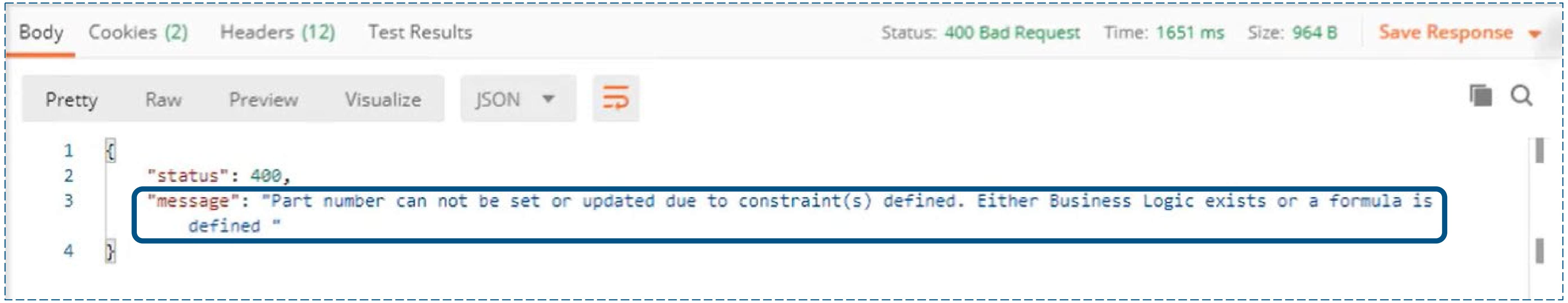
Create Engineering Item – Demonstration – Steps (12/14)

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.



```
{
    "status": 400,
    "message": "Part number can not be set or updated due to constraint(s) defined. Either Business Logic exists or a formula is
        defined "
}
```

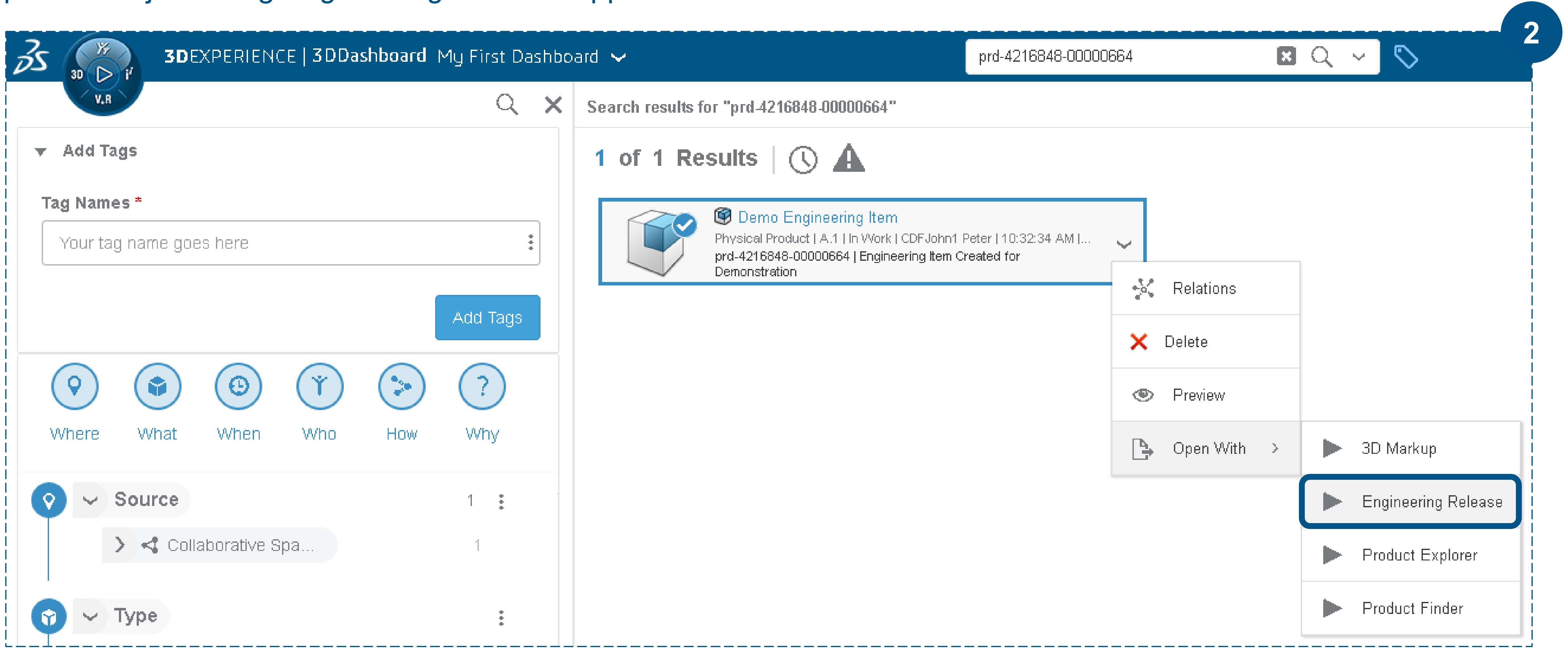
- In this case, the **partNumber** parameter should be removed from the request body of the web-service call. This would allow the Business Rule to set the Enterprise Item Number automatically.

Create Engineering Item – Demonstration – Steps (13/14)

Verify the creation of Engineering Item 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.

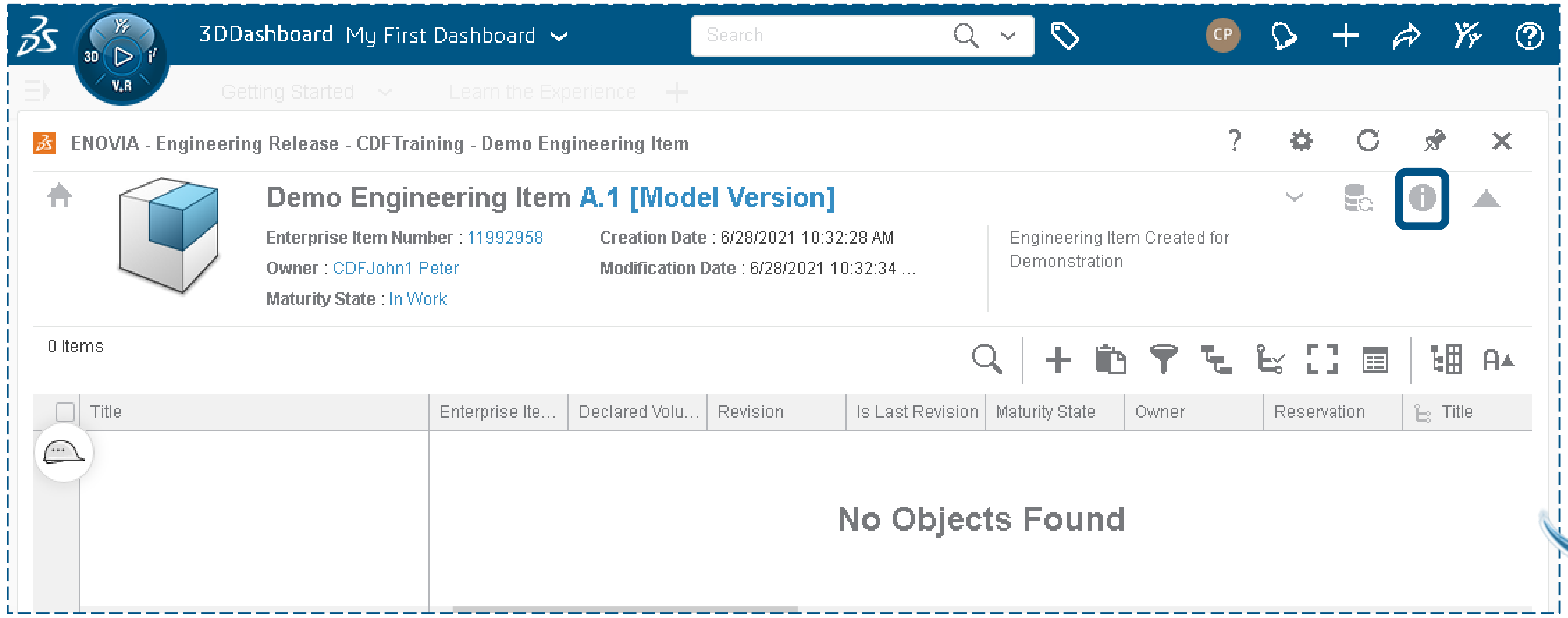


To be able to search the Engineering Item object, it needs to be indexed first. Please allow sufficient time for indexation to happen, before searching for object.

Create Engineering Item – Demonstration – Steps (14/14)

Verify the creation of Engineering Item 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.



ENOVIA - Engineering Release - CDFTraining - Demo Engineering Item

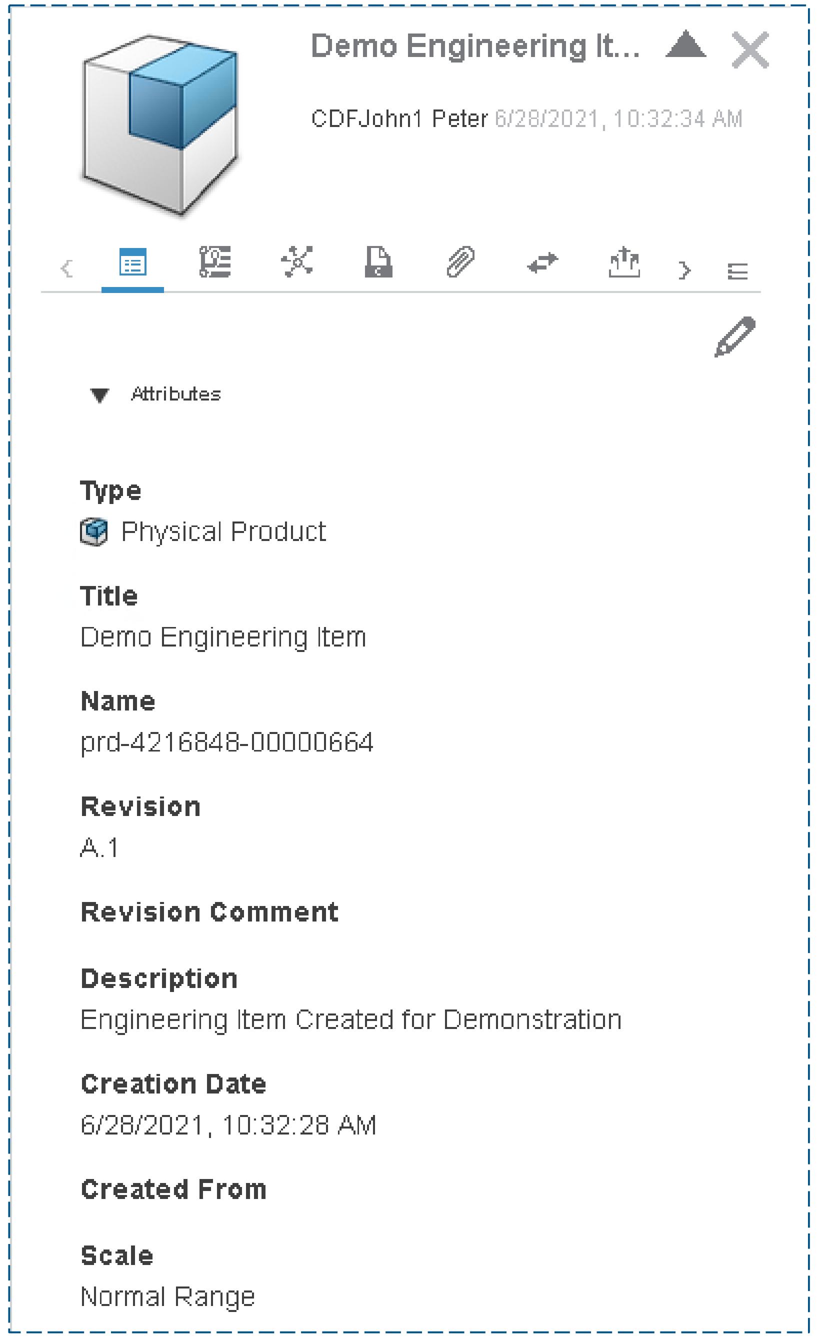
Demo Engineering Item A.1 [Model Version]

Enterprise Item Number : 11992958
Owner : CDFJohn1 Peter
Maturity State : In Work
Creation Date : 6/28/2021 10:32:28 AM
Modification Date : 6/28/2021 10:32:34 ...
Engineering Item Created for Demonstration

0 Items

Title	Enterprise Ite...	Declared Volu...	Revision	Is Last Revision	Maturity State	Owner	Reservation	Title

No Objects Found



Demo Engineering It...

CDFJohn1 Peter 6/28/2021, 10:32:34 AM

Attributes

Type
Physical Product

Title
Demo Engineering Item

Name
prd-4216848-00000664

Revision
A.1

Revision Comment

Description
Engineering Item Created for Demonstration

Creation Date
6/28/2021, 10:32:28 AM

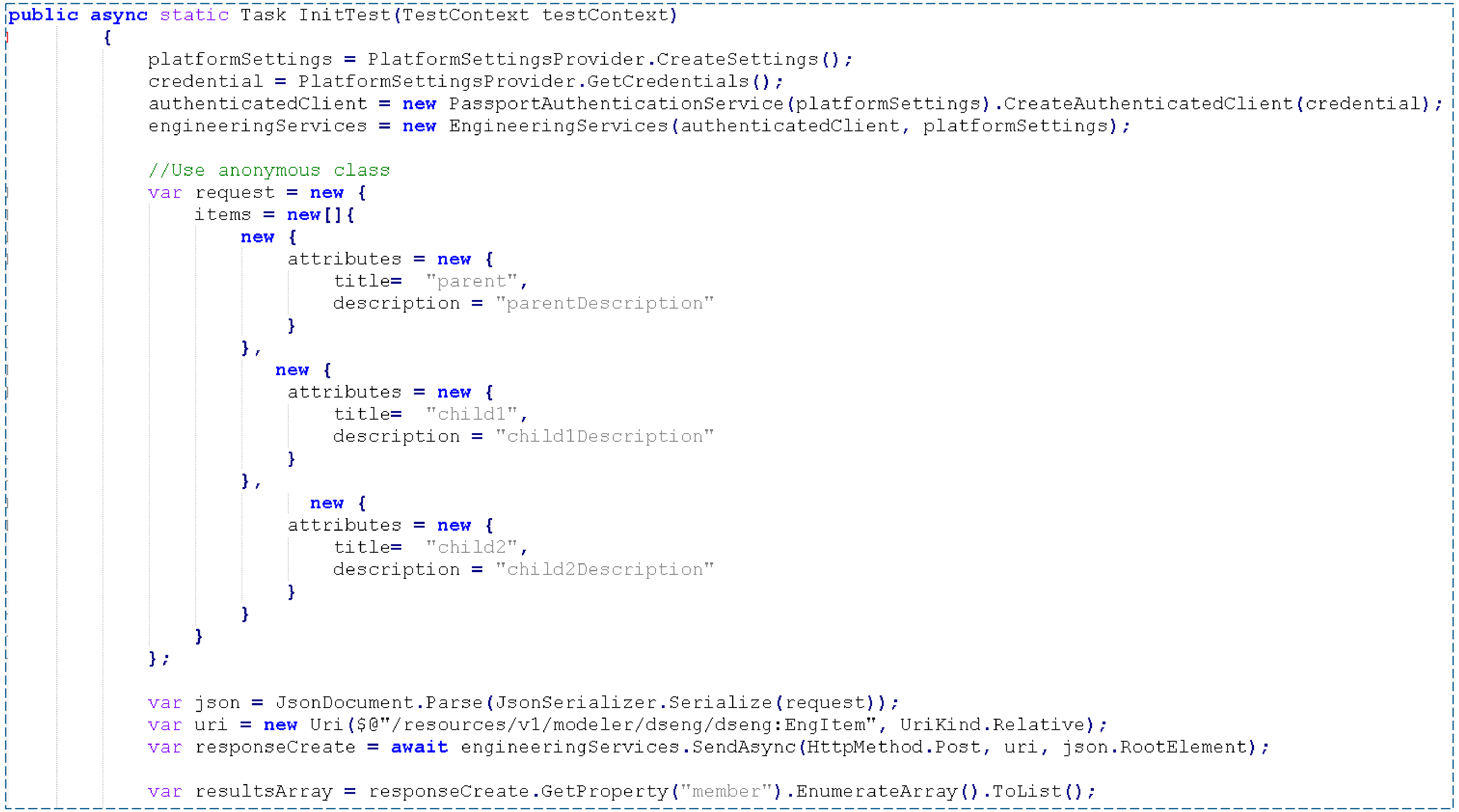
Created From

Scale
Normal Range

Create Engineering Item – Automation

1. 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.

2. 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.

Additional Topics

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

- Agent Management
- Event Publishing
- Enterprise IP Integration Management

Next Generation Integration Framework

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

Here are the topics to be covered:

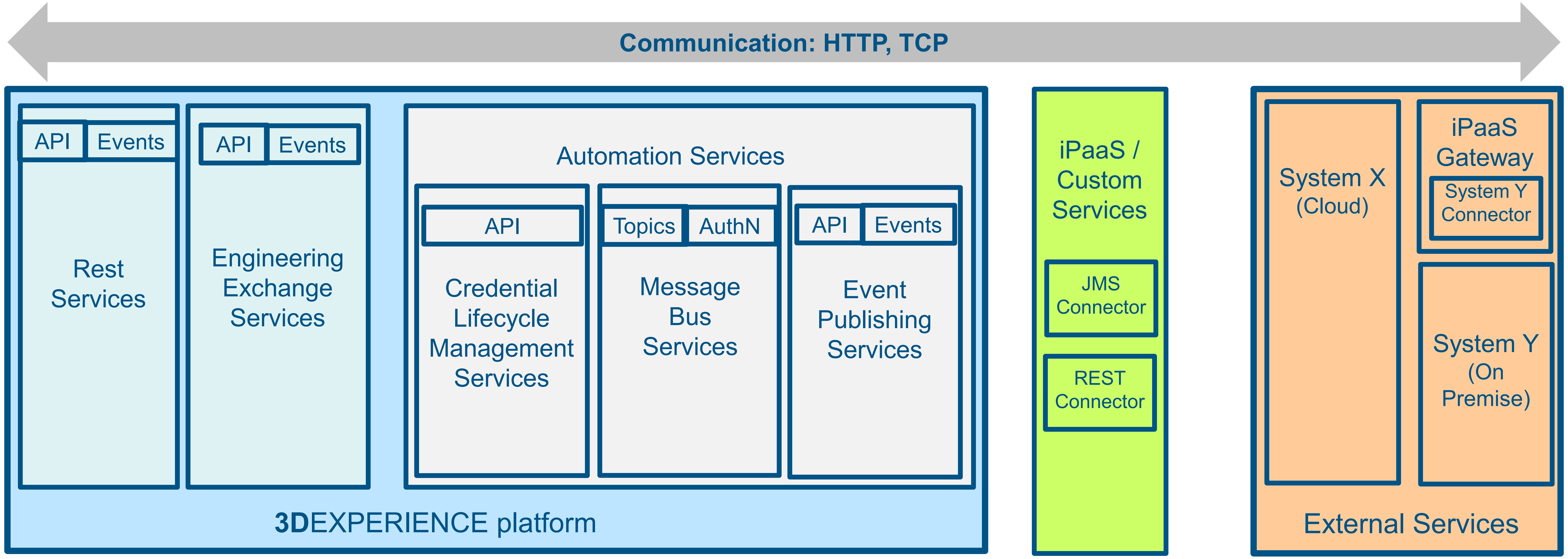
1. Next Generation Integration Framework

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.

    - 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.

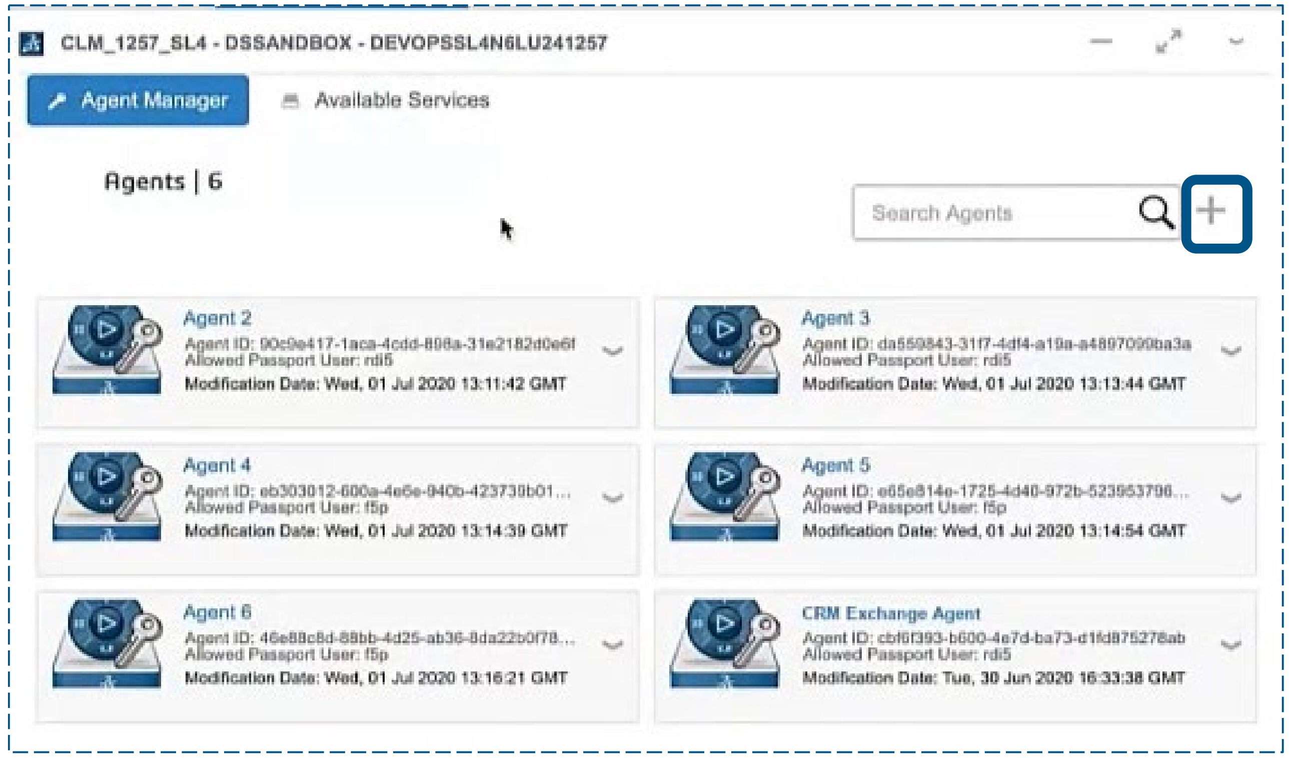
    - These services provides JMS connector and REST connector to achieve integration between other enterprise applications and **3D**EXPERIENCE platform.

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

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



Agent 2
Agent ID: 90c9e417-1aca-4cdd-808a-31e2182d0e6f
Allowed Passport User: rdi5
Modification Date: Wed, 01 Jul 2020 13:11:42 GMT

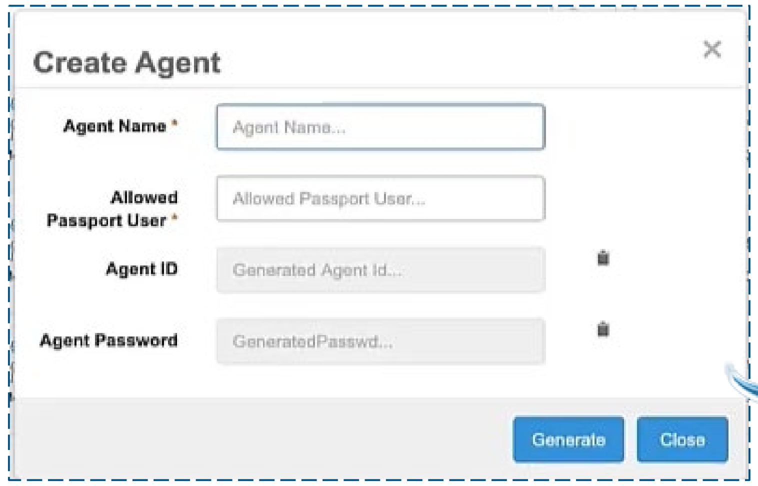
Agent 3
Agent ID: da559843-31f7-4df4-a19a-a4897090ba3a
Allowed Passport User: rdi5
Modification Date: Wed, 01 Jul 2020 13:13:44 GMT

Agent 4
Agent ID: eb303012-600a-4e6e-940b-423738b01...
Allowed Passport User: f5p
Modification Date: Wed, 01 Jul 2020 13:14:39 GMT

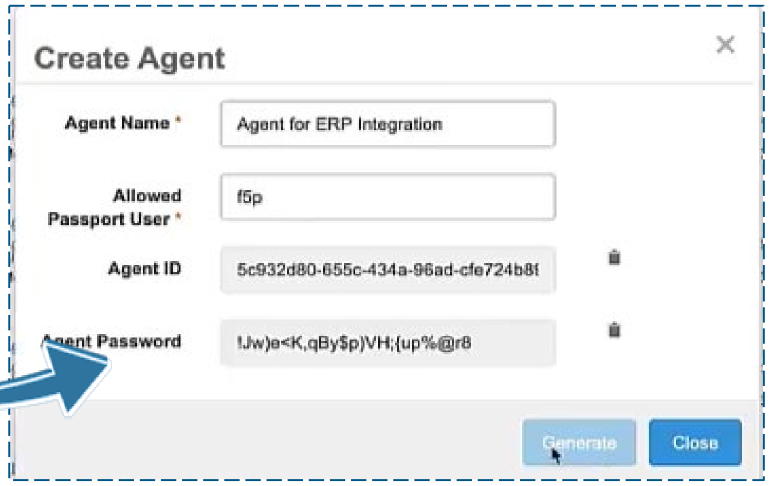
Agent 5
Agent ID: e65e814e-1725-4d40-972b-523953798...
Allowed Passport User: f5p
Modification Date: Wed, 01 Jul 2020 13:14:54 GMT

Agent 6
Agent ID: 46e88c8d-88bb-4d25-ab36-8da22b0f78...
Allowed Passport User: f5p
Modification Date: Wed, 01 Jul 2020 13:16:21 GMT

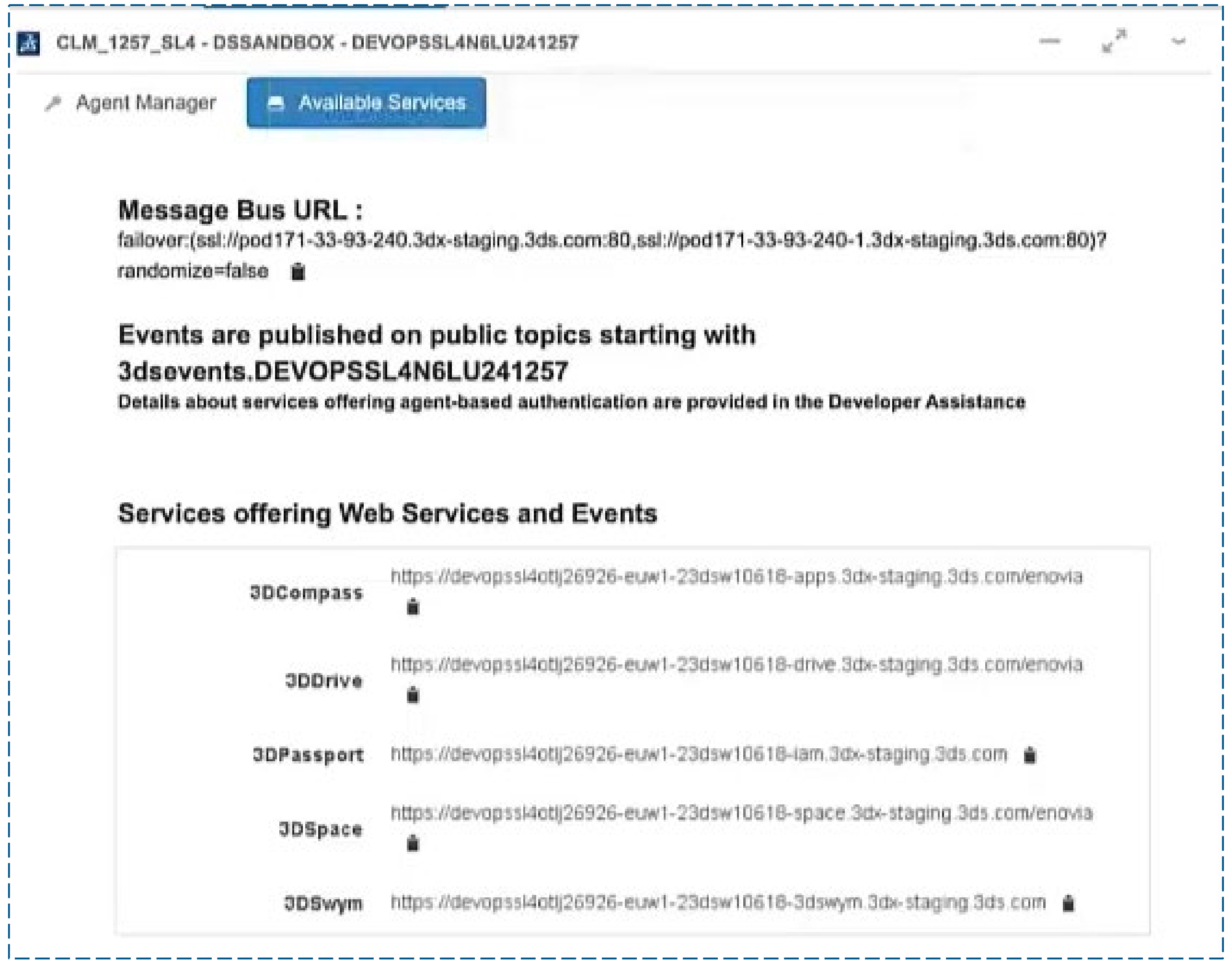
CRM Exchange Agent
Agent ID: cbf6f393-b600-4e7d-ba73-d1fd875278ab
Allowed Passport User: rdi5
Modification Date: Tue, 30 Jun 2020 16:33:38 GMT



Agent Name *	Agent Name...
Allowed Passport User *	Allowed Passport User...
Agent ID	Generated Agent Id...
Agent Password	GeneratedPasswd...



Agent Name *	Agent for ERP Integration
Allowed Passport User *	f5p
Agent ID	5c932d80-655c-434a-96ad-cfe724b8
Agent Password	!Jw)e<K,qBy$p)VH;{up%@r8



Message Bus URL :
failover:(ssl://pod171-33-93-240.3dx-staging.3ds.com:80,ssl://pod171-33-93-240-1.3dx-staging.3ds.com:80)?randomize=false

Events are published on public topics starting with 3dsevents.DEVOPSSL4N6LU241257
Details about services offering agent-based authentication are provided in the Developer Assistance

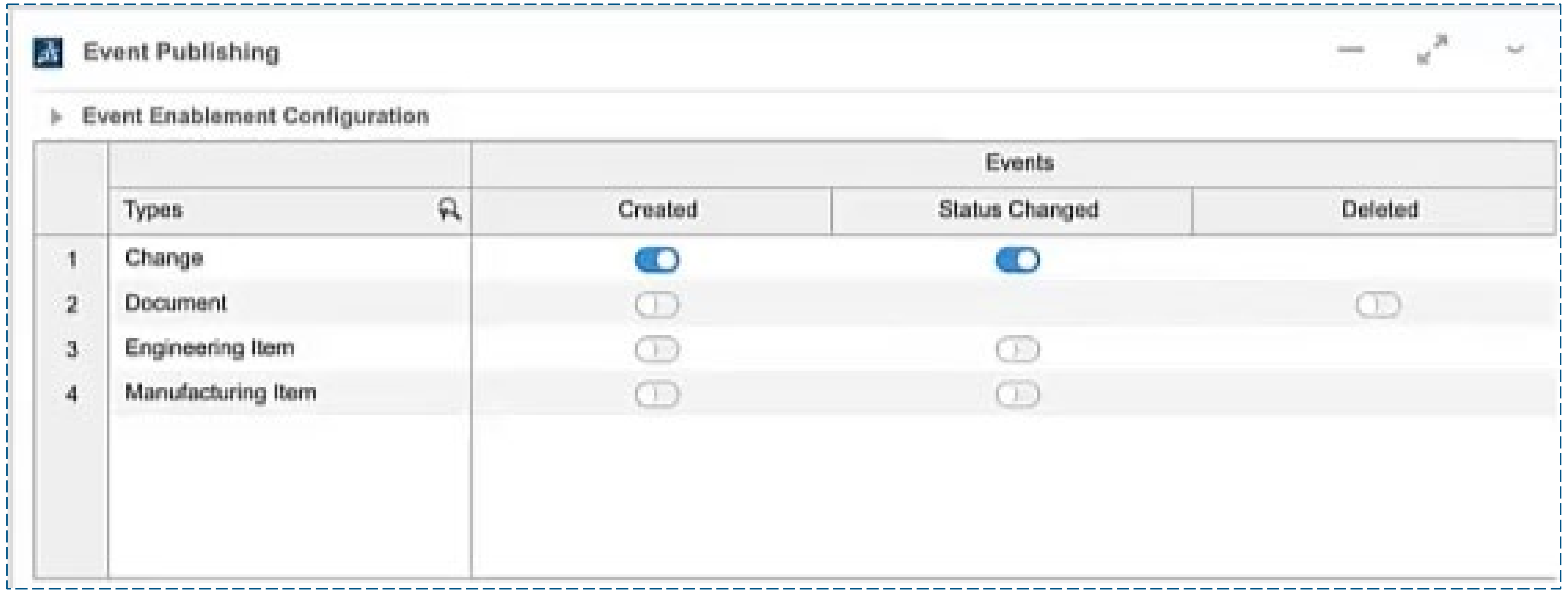
Services offering Web Services and Events

3DCompass	https://devopssl4otlj26926-euw1-23dsw10618-apps.3dx-staging.3ds.com/enovia
3DDrive	https://devopssl4otlj26926-euw1-23dsw10618-drive.3dx-staging.3ds.com/enovia
3DPassport	https://devopssl4otlj26926-euw1-23dsw10618-iam.3dx-staging.3ds.com
3DSpace	https://devopssl4otlj26926-euw1-23dsw10618-space.3dx-staging.3ds.com/enovia
3DSwym	https://devopssl4otlj26926-euw1-23dsw10618-3dswym.3dx-staging.3ds.com

Enterprise Integration Architect Role (2/2)

Event Publishing

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



Event Publishing

Event Enablement Configuration

		Events		
	Types	Created	Status Changed	Deleted
1	Change	●	●	
2	Document	○		○
3	Engineering Item	○	○	
4	Manufacturing Item	○	○	

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.

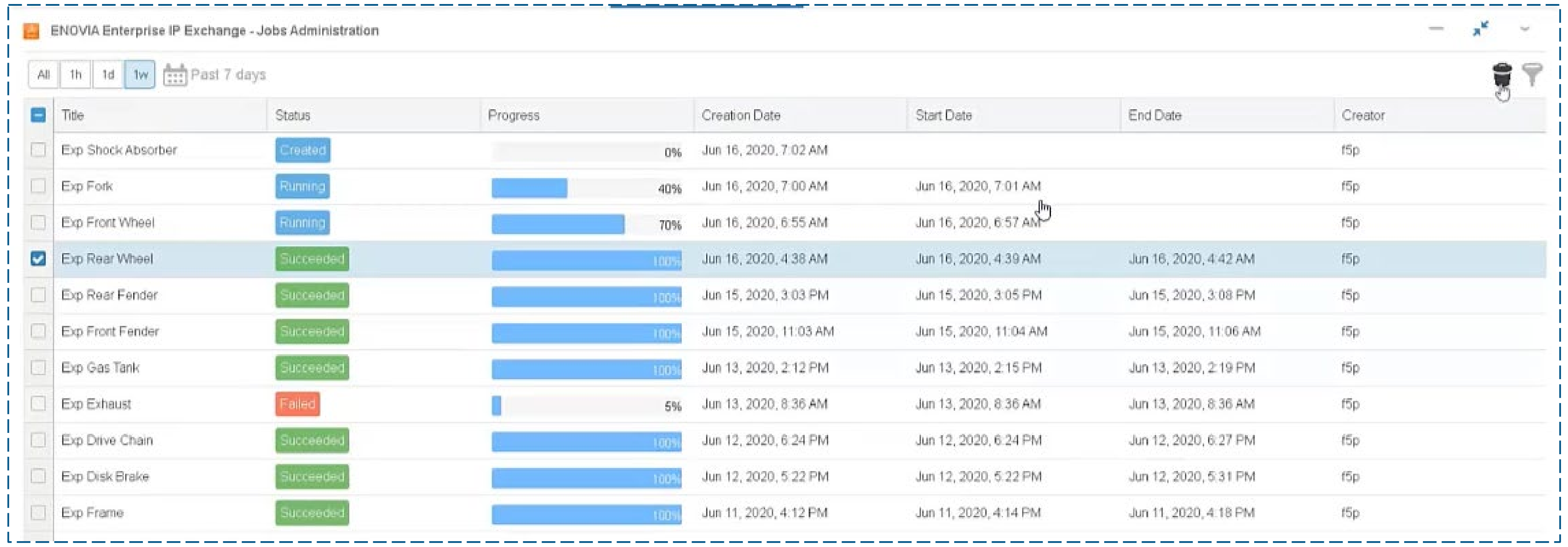
    Also, this role allows access to **Enterprise IP Integration Management** app to monitor the IP Exchange jobs.

  - **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)

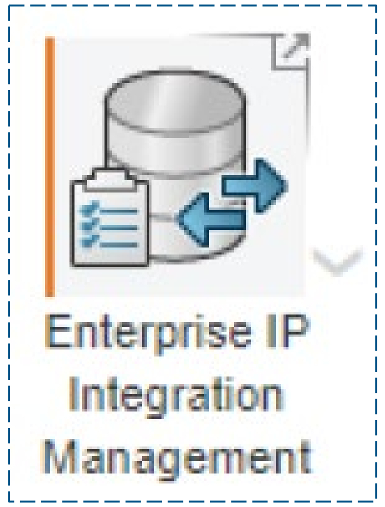
- The **Enterprise IP Integration Management** app allows the *IP Exchange Administrator* to monitor IP exchange jobs, and generate reports after their execution.

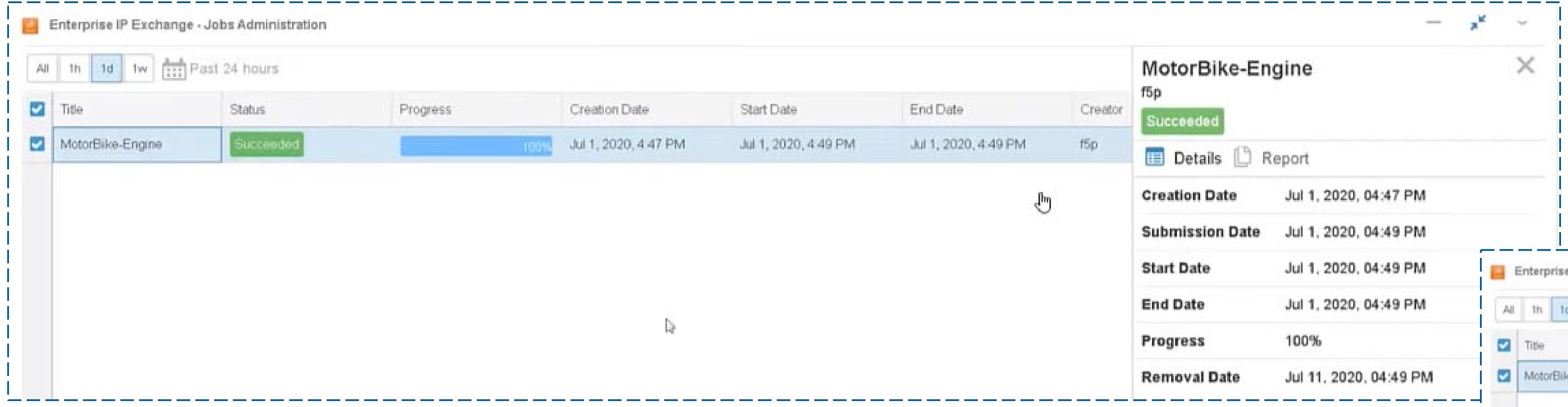


ENOVIA Enterprise IP Exchange - Jobs Administration

All | 1h | 1d | 1w | Past 7 days

Title	Status	Progress	Creation Date	Start Date	End Date	Creator
Exp Shock Absorber	Created	0%	Jun 16, 2020, 7:02 AM			f5p
Exp Fork	Running	40%	Jun 16, 2020, 7:00 AM	Jun 16, 2020, 7:01 AM		f5p
Exp Front Wheel	Running	70%	Jun 16, 2020, 6:55 AM	Jun 16, 2020, 6:57 AM		f5p
Exp Rear Wheel	Succeeded	100%	Jun 16, 2020, 4:38 AM	Jun 16, 2020, 4:39 AM	Jun 16, 2020, 4:42 AM	f5p
Exp Rear Fender	Succeeded	100%	Jun 15, 2020, 3:03 PM	Jun 15, 2020, 3:05 PM	Jun 15, 2020, 3:08 PM	f5p
Exp Front Fender	Succeeded	100%	Jun 15, 2020, 11:03 AM	Jun 15, 2020, 11:04 AM	Jun 15, 2020, 11:06 AM	f5p
Exp Gas Tank	Succeeded	100%	Jun 13, 2020, 2:12 PM	Jun 13, 2020, 2:15 PM	Jun 13, 2020, 2:19 PM	f5p
Exp Exhaust	Failed	5%	Jun 13, 2020, 8:36 AM	Jun 13, 2020, 8:36 AM	Jun 13, 2020, 8:36 AM	f5p
Exp Drive Chain	Succeeded	100%	Jun 12, 2020, 6:24 PM	Jun 12, 2020, 6:24 PM	Jun 12, 2020, 6:27 PM	f5p
Exp Disk Brake	Succeeded	100%	Jun 12, 2020, 5:22 PM	Jun 12, 2020, 5:22 PM	Jun 12, 2020, 5:31 PM	f5p
Exp Frame	Succeeded	100%	Jun 11, 2020, 4:12 PM	Jun 11, 2020, 4:14 PM	Jun 11, 2020, 4:18 PM	f5p





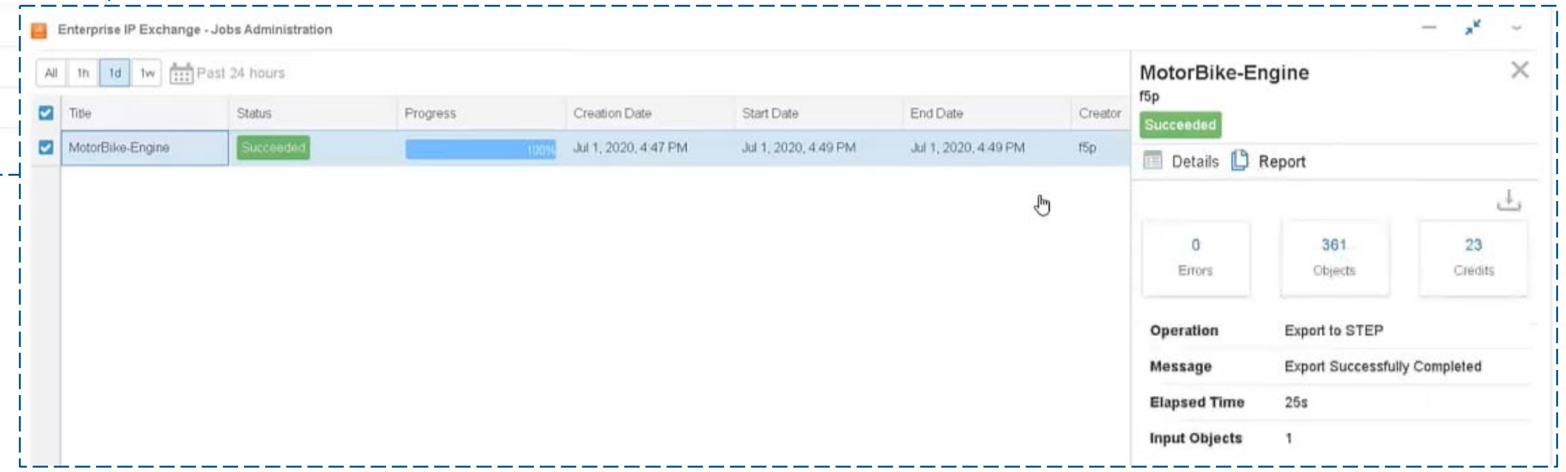
Enterprise IP Exchange - Jobs Administration

All | 1h | 1d | 1w | Past 24 hours

Title	Status	Progress	Creation Date	Start Date	End Date	Creator
MotorBike-Engine	Succeeded	100%	Jul 1, 2020, 4:47 PM	Jul 1, 2020, 4:49 PM	Jul 1, 2020, 4:49 PM	f5p

MotorBike-Engine
f5p
Succeeded
Details | Report

Creation Date	Jul 1, 2020, 04:47 PM
Submission Date	Jul 1, 2020, 04:49 PM
Start Date	Jul 1, 2020, 04:49 PM
End Date	Jul 1, 2020, 04:49 PM
Progress	100%
Removal Date	Jul 11, 2020, 04:49 PM



Enterprise IP Exchange - Jobs Administration

All | 1h | 1d | 1w | Past 24 hours

Title	Status	Progress	Creation Date	Start Date	End Date	Creator
MotorBike-Engine	Succeeded	100%	Jul 1, 2020, 4:47 PM	Jul 1, 2020, 4:49 PM	Jul 1, 2020, 4:49 PM	f5p

MotorBike-Engine
f5p
Succeeded
Details | Report

0	361	23
Errors	Objects	Credits

Operation	Export to STEP
Message	Export Successfully Completed
Elapsed Time	25s
Input Objects	1

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.

  - **Step 4**: Based on the topics that are subscribed to in *Step2*, and using Integration Agent created in *Step 1*, IP Exchange jobs can be created.

IF WE™