Db2 for z/OS Native REST Services and z/OS Connect EE Integration

Lab exercises

Created for Db2 REST for Hybrid Cloud Wildfire Workshop

Date: February 1, 2022

Version: V3.6

Authors:

Tadas Varaneckas

MacKenna Kelleher

Contents

Overview

IBM Db2 for z/OS versions 11 and 12 have the ability to be a REST (**Re**presentational **S**tate **T**ransfer) service provider. In this workshop, you will learn how quickly Db2 stored procedures and SQL statements can be enabled as REST services, and support REST interaction with your web, mobile and cloud applications. In addition, you will learn how to enhance Db2 REST services using IBM z/OS Connect Enterprise Edition 3.0 (z/OS Connect or zCEE) and transform them into REST APIs.

Db2 REST services are fully integrated into the Db2 distributed data facility (DDF). All Db2 REST services are managed natively within the Db2 subsystem/member. The Db2 native REST service solution leverages the existing DDF capabilities for authentication, authorization, client information management, service classification, system profiling, and service monitoring. Db2 defines a REST service as a package. Each package contains a single static SQL statement and is recorded in the SYSIBM.DSNSERVICE catalog table. The following SQL statements are supported: CALL, DELETE, INSERT, SELECT, TRUNCATE, UPDATE and WITH. Db2 provides REST APIs to create, discover and manage user-defined REST services in Db2. Db2 REST user-defined services are invoked by the POST method only. The zCEE API Editor can reassign this POST method. For example, in Lab 2 you will use the zCEE API Editor to map the Db2 REST service's POST method to the appropriate GET method.

IBM z/OS Connect Enterprise Edition provides a framework that enables z/OS based programs and data to participate fully in the new API economy for mobile and cloud applications. z/OS Connect provides RESTful access to z/OS subsystems such as Db2, CICS, IMS, WebSphere MQ, and Batch. z/OS Connect includes tools which can enhance Db2 REST services, by creating RESTful APIs from Db2 REST services.

The purpose of these labs is to provide the information necessary to create Db2 z/OS REST services and to integrate Db2 REST services into z/OS Connect RESTful APIs, which can be used by mobile and cloud applications.

Introduction

This introduction provides the system requirements and instructions to create Db2 REST services and APIs. The following labs are presented in a series of instructions using Db2 and the Db2 installation verification program's (IVP) "EMP" table created in job DSNTEJ1. The IVP EMP table should be available on customer systems to practice with local subsystems.

The following concepts are covered in this workshop:

- Create a Db2 REST services using an existing stored procedure and/or a SQL statement
- Define a Db2 REST service in z/OS Connect EE
- Create a Db2 REST API using the z/OS Connect EE API Editor and Swagger document

Software inventory used in this workshop to complete the exercises

1) IBM Db2 12 for z/OS
2) IBM z/OS Connect EE 3 – Version: 3
3) IBM Explorer for z/OS – Version 3 Aqua
4) IBM z/OS Connect EE API Editor
5) A REST API testing tool – the Postman Client will be used in the exercises

Icons

The following symbols appear in this document at places where additional guidance is available.

Icon	Purpose	Explanation
	Important!	This symbol calls attention to a particular step or command. For example, it might alert you to type a command carefully because it is case sensitive.
	Information	This symbol indicates information that might not be necessary to complete a step but is helpful or good to know.
	Troubleshooting	This symbol indicates that you can fix a specific problem by completing the associated troubleshooting information.

Lab 1 Db2 Native REST Services

This exercise covers creating a Db2 REST service using the Db2 BIND command using an update provided in Db2 PTF UI51748 and APAR PI98649 (PTF UI584231 or UI58425). The Db2 REST service will use a Db2 native SQL stored procedure or a SQL statement. When creating the service, you have the option to use an already existing stored procedure, or a simple SQL statement. If you would like to use both, two separate batch jobs must be submitted. A Db2 REST service is restricted to only one stored procedure or SQL statement. Customers are frequently interested in using Db2 REST services with stored procedures because they want to share existing enterprise business logic with mobile and cloud applications.

Db2 z/OS REST services do support HTTPS, but requires z/OS “Application Transparent – Transport Level Security” (AT-TLS).

Db2 REST services can be created and managed using IBM Data Studio and REST workstation development tools.

Note: Data Studio requires Db2 z/OS AT-TLS to be operational to create Db2 REST services.

Important!

Only one Db2 object (stored procedure or SQL statement) can be included in a Db2 REST service.

Important!

In this lab you have the option to create a Db2 Native REST service that uses a simple SQL statement or an existing stored procedure, or both.

If you would like to create both, you must send two separate batch jobs.

1.1 Review of the Db2 system and sample Db2 stored procedure

This section will provide the system information needed to access the Db2 subsystem from your virtual environment, as well as the stored procedure created for your use throughout the lab.

System information

The following Db2 information will be used to create the Db2 REST service:

- Db2 DNS name: wg31.washington.ibm.com
- Db2 TCP/IP port: 2446
- Db2 user name: USER1
- Db2 user password: USER1

Stored procedure

The Db2 Stored Procedure, **EMPL_DEPTS_NAT** has already been created using the SQL Processor Using File Input (SPUFI). The code of the Stored Procedure is provided below for your reference:

```
------------------------------------------------------------------------------------------------------------------------
CREATE PROCEDURE EMPL_DEPTS_NAT
     (IN WHICHQUERY INTEGER, IN DEPT1 CHARACTER(3), IN DEPT2
CHARACTER(3))
VERSION V1
           RESULT SETS 1
     LANGUAGE SQL
     ISOLATION LEVEL CS
     DISABLE DEBUG MODE
P1: BEGIN
DECLARE CURSOR1 CURSOR WITH RETURN FOR
     SELECT EMPLOYEE.EMPNO, EMPLOYEE.FIRSTNME, EMPLOYEE.MIDINIT,
     EMPLOYEE.LASTNAME,
     EMPLOYEE.WORKDEPT, EMPLOYEE.PHONENO
            FROM DSN81210.EMP AS EMPLOYEE
            WHERE EMPLOYEE.WORKDEPT=DEPT1
            ORDER BY EMPLOYEE.EMPNO ASC;
```

If WHICHQUERY=1, SELECT one department only – DEPT1 is the department

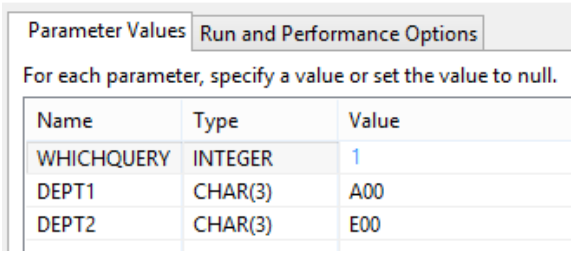
```
DECLARE CURSOR2 CURSOR WITH RETURN FOR
      SELECT EMPLOYEE.EMPNO, EMPLOYEE.FIRSTNME, EMPLOYEE.MIDINIT,
      EMPLOYEE.LASTNAME,
      EMPLOYEE.WORKDEPT, EMPLOYEE.PHONENO
         FROM DSN81210.EMP AS EMPLOYEE
         WHERE EMPLOYEE.WORKDEPT>=DEPT1 AND EMPLOYEE.WORKDEPT<=DEPT2
         ORDER BY EMPLOYEE.WORKDEPT ASC, EMPLOYEE.EMPNO ASC;
CASE WHICHQUERY
    WHEN 1 THEN
        OPEN CURSOR1;
    ELSE
        OPEN CURSOR2;
END CASE;
END P1#
```

If WHICHQUERY=2, SELECT multiple departments – from DEPT1 until DEPT2

NOTE: At your site, using this stored procedure format you would need to change the **SPUFI SQL TERMINATOR to #** for this statement to be created properly. For more information on how to use SPUFI in TSO, please visit the following page of the Executing SQL by Using SPUFI page of the IBM Knowledge Center.

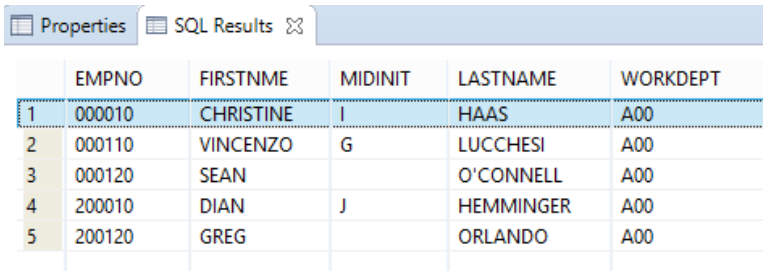
Below are the input parameters and sample output for this stored procedure:

Input parameters:



Name	Type	Value
WHICHQUERY	INTEGER	1
DEPT1	CHAR(3)	A00
DEPT2	CHAR(3)	E00

Sample stored procedure result set output:



	EMPNO	FIRSTNME	MIDINIT	LASTNAME	WORKDEPT
1	000010	CHRISTINE	I	HAAS	A00
2	000110	VINCENZO	G	LUCCHESI	A00
3	000120	SEAN		O'CONNELL	A00
4	200010	DIAN	J	HEMMINGER	A00
5	200120	GREG		ORLANDO	A00

Summary: In this section, we reviewed the Db2 System Information and sample Db2 stored procedure.

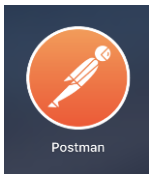
1.2 Lab Preparations and Discovering Services

Db2 REST services can be created and managed using IBM Data Studio and REST debugging tools. In this step, you will set up Postman for use in testing and executing the REST services that you will be creating using the Db2 BIND subcommand.

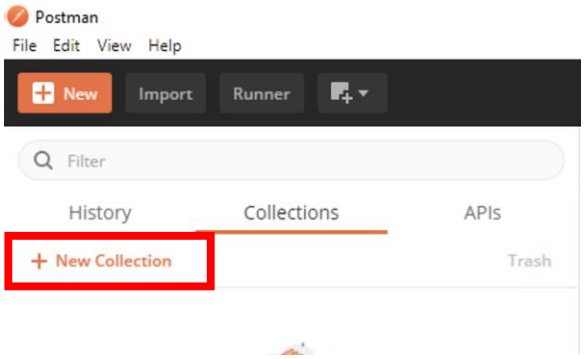
1.2.1 Launching the Postman tool and Authorization

In this exercise, you will use the tools available for internet download to work with REST APIs. In this section, you will use the Postman tool installed on the workstation, which allows you to issue REST requests to execute the Db2 REST Services.

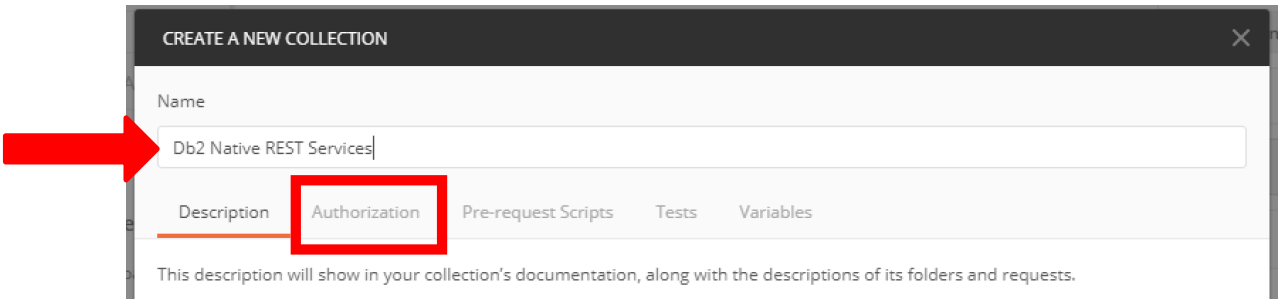
__1. Double click on the **Postman** icon:



__2. When Postman launches, click on **+ New Collection** in the side navigation bar to create a new folder for the requests that we are going to create.



__3. In the “Create a New Collection” window, name the collection “Db2 Native REST Services”, a unique name to identify the collection. Then click the **Authorization** tab, to set the authentication that will be used for all the REST requests that will be saved to the collection.

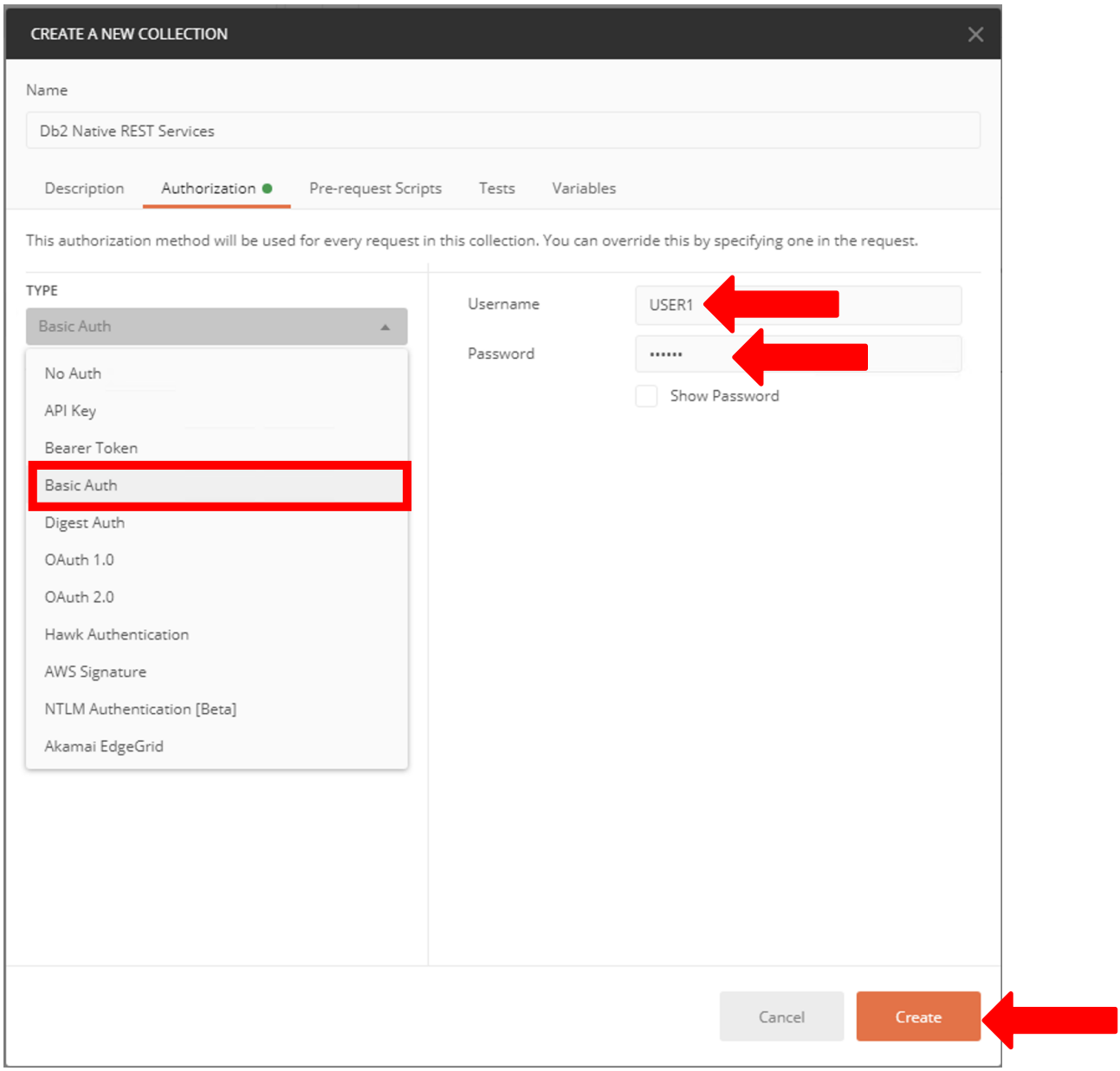


__4. Select **Basic Auth** from the dropdown menu under **Type** and enter the following information:

A. Username: USER1

B. Password: USER1

Then click **Create**.



__5. Once the collection is created, we can start creating and saving requests that will be useful in this lab.

Summary: In this first section, we launched Postman and authenticated our collection using Basic Authentication.

1.2.2 Discovering all Db2 REST services currently installed in the Db2 catalog

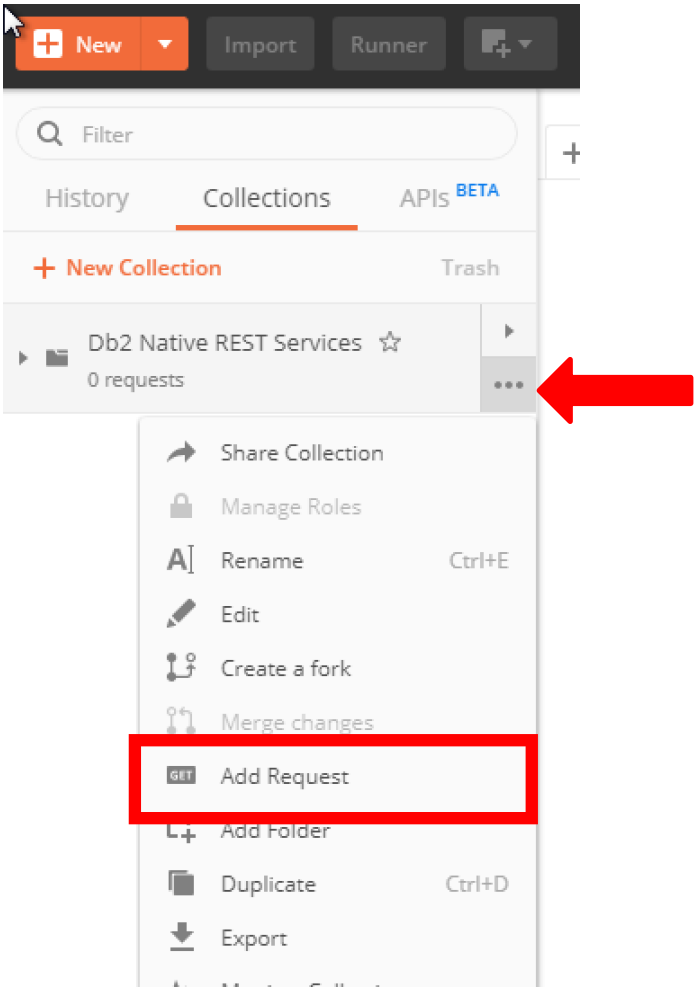
To start getting familiar with the Postman client, you will issue the Db2 REST discover request. In order to use this discover API, you must have the required authority. The response to this request will entail a list of all the native REST services available.

To ensure that you may invoke this request, BEFORE YOU BEGIN, you must have one of the following privileges or authorities to discover Db2 REST services:

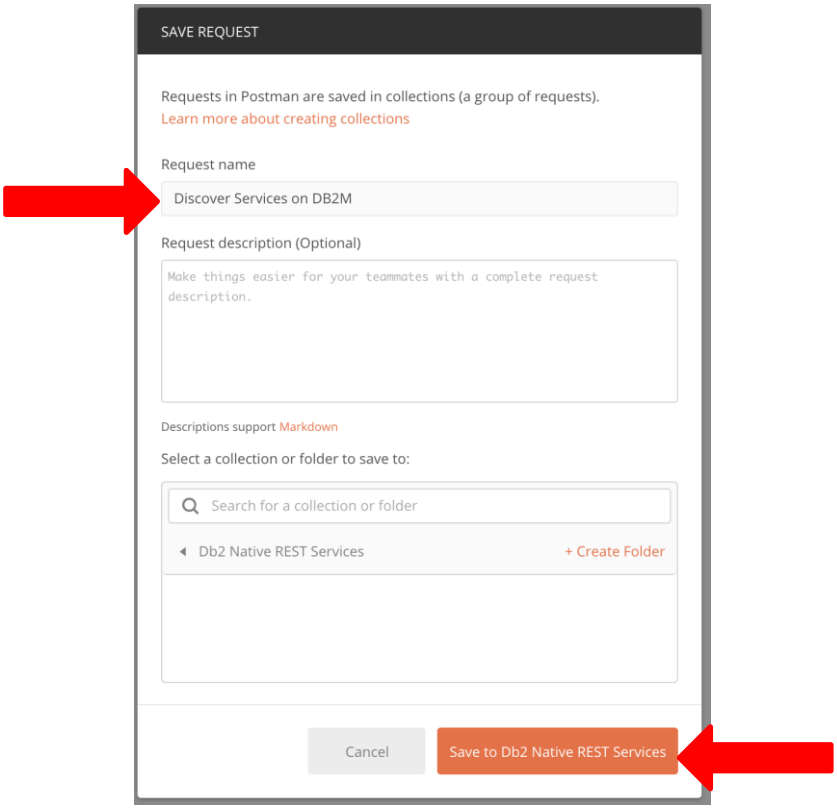
- Execute privilege on the package for the service
- Ownership of the service
- SYSADM or SYSCTRL authority
- System DBADM

__1. Opening a new request within the Collection

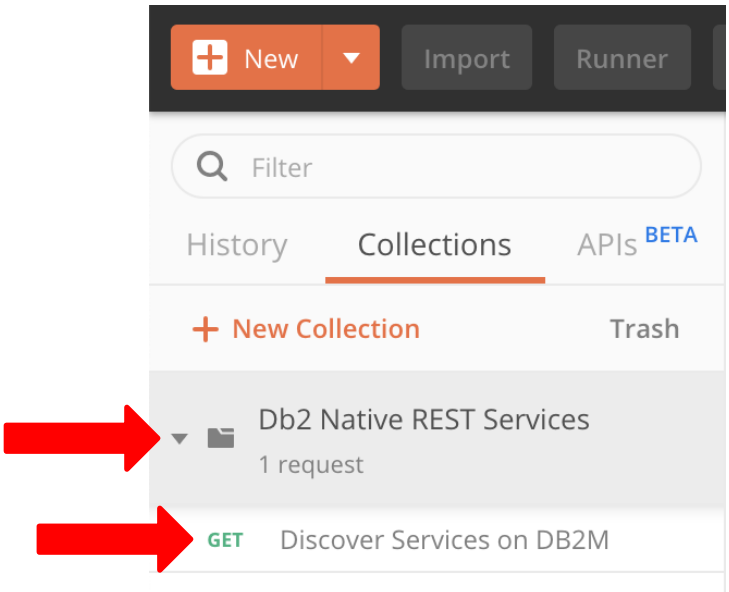
A. Click on the ellipses in the corner of the “Db2 Native REST Services” collection in the side navigation bar and select **Add Request** from the dropdown menu.



__2. By adding a new request directly to a Collection, Postman will first ask you to save the request. Since this request we will be invoking the discover API, name this request "Discover Services on DB2M" and click **Save to Db2 Native REST Services**.



__3. Open the newly created request by clicking on the "Db2 Native REST Services" Collection in the side navigation bar and then "Discover Services on DB2M" Request.



__4. Update the request with the information below to discover all Db2 REST services currently installed in the Db2 catalog.

A. Set the REST Method to: **GET**

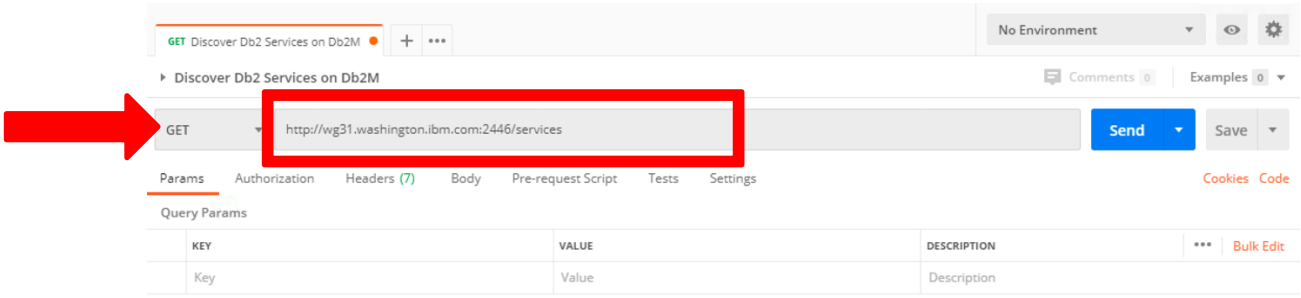
B. Add the Db2 discover API to the REST URL:

http://wg31.washington.ibm.com:2446/services

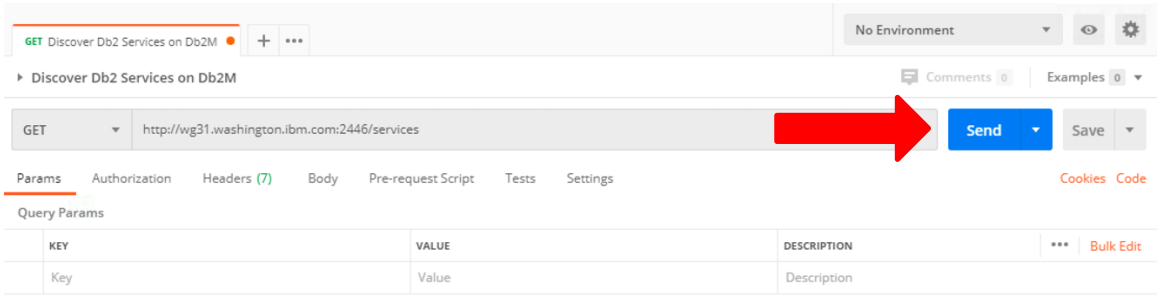
__i. Db2 DNS name = **wg31.washington.ibm.com**

__ii. Db2 port = **2446**

__iii. Db2 Discover URI = **/services**



__5. Click **SEND** to invoke the command.



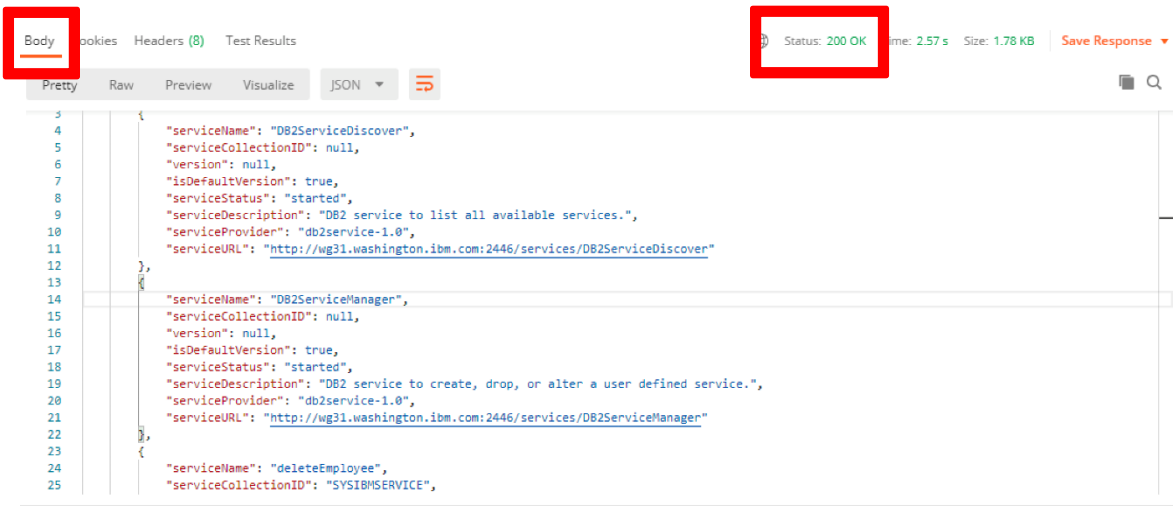
Information

You can also use any browser and the URL below to obtain a list of the Db2 REST services, providing the right authority. Enter the same username and password for USER1 when prompted.

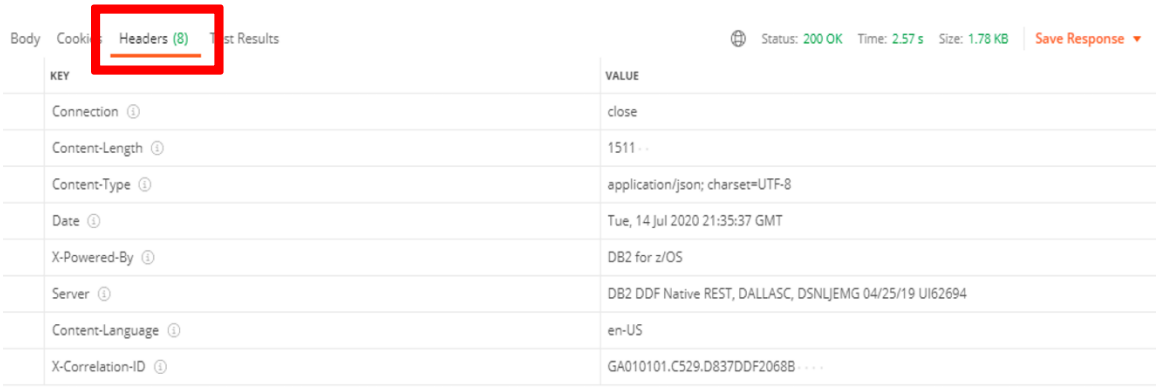
http://wg31.washington.ibm.com:2446/services

__6. Discover Response. Confirm the proper output below:

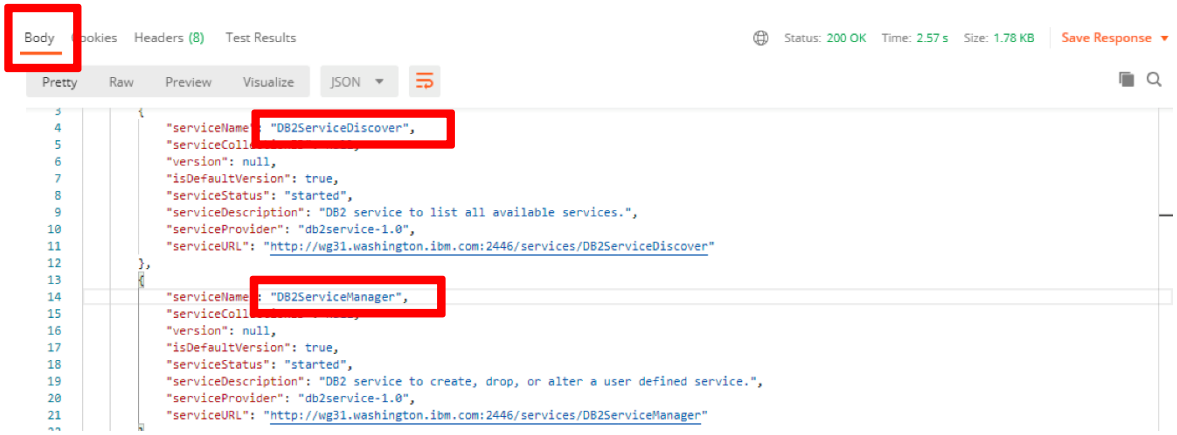
A. The first items displayed for review are response status, stats, and response **Body**. The status **200 OK** will be the normal successful return code for Db2 services.



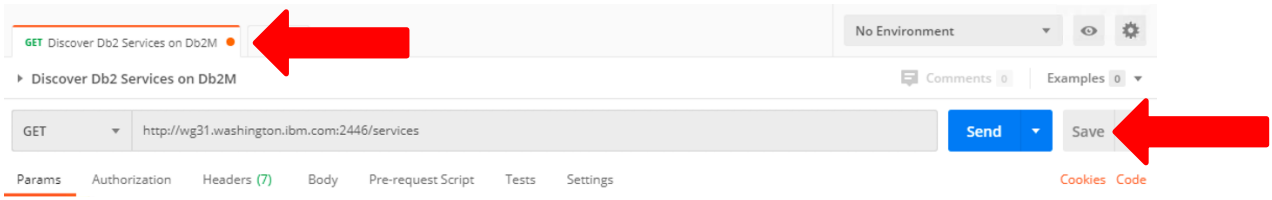
B. The response headers information may be viewed by clicking on the **Headers** tab for analysis.



C. Return to the response body by clicking the **Body** tab. Scroll down to see the Db2 services installed. Db2 services: “DB2ServiceDiscover” and “DB2ServiceManager” are for system management.



__7. **SAVE** the changes made to the request. You will know that the **SAVE** was successful when the orange dot on the “Discover Services on Db2M” tab disappears, and an “X” is in its place.



Information

Any time a change is made in a request, an orange dot appears in the corner of the request tab indicating the request was edited. If the request isn’t saved before closing the tab, Postman will prompt you if you want to save the request.

Be careful not to *override* a saved request with unwanted changes.

Information

Postman uses the request tabs for easy navigation purposes when working with multiple request tabs. You can close the tabs once you are done with the request or leave them open to navigate to in the future.

Summary: In this step, we used the REST discover API to confirm that the Db2 REST server is operational using the Postman Client, and to view the installed services on Db2M.

1.3 Creating a Db2 REST service

The following sections demonstrate how quickly a Db2 REST service can be created using a Db2 stored procedure and/or a SQL statement. Creating REST services from stored procedures allows existing business logic to be used by new applications in the API economy. The DB2 BIND subcommand will be used to create the Db2 user-defined REST service, and the REST service will be stored in the Db2 catalog (table DSNSERVICE) ready for use. Postman will be used to test the execution of the Db2 REST service.

When you create a service, Db2 identifies you – or the authorization ID that you use – as the default owner of the service. Therefore, you must have the required privileges to create a service and BIND the associated package into a collection. For example, you must be authorized to execute the SQL statement that is embedded in the service. See BIND PACKAGE (DSN) page in IBM Documentation for information regarding the privileges and authorities that you must have to create a package for a Db2 REST service.

To reiterate, in this section, you will be using a JCL batch job to create a Db2 Native REST Service using z/OS Explorer. For instructions and information on how to create these Db2 Native REST Services using the a REST request via the Postman Client, please see Appendix A at the end of this lab document.

Important!

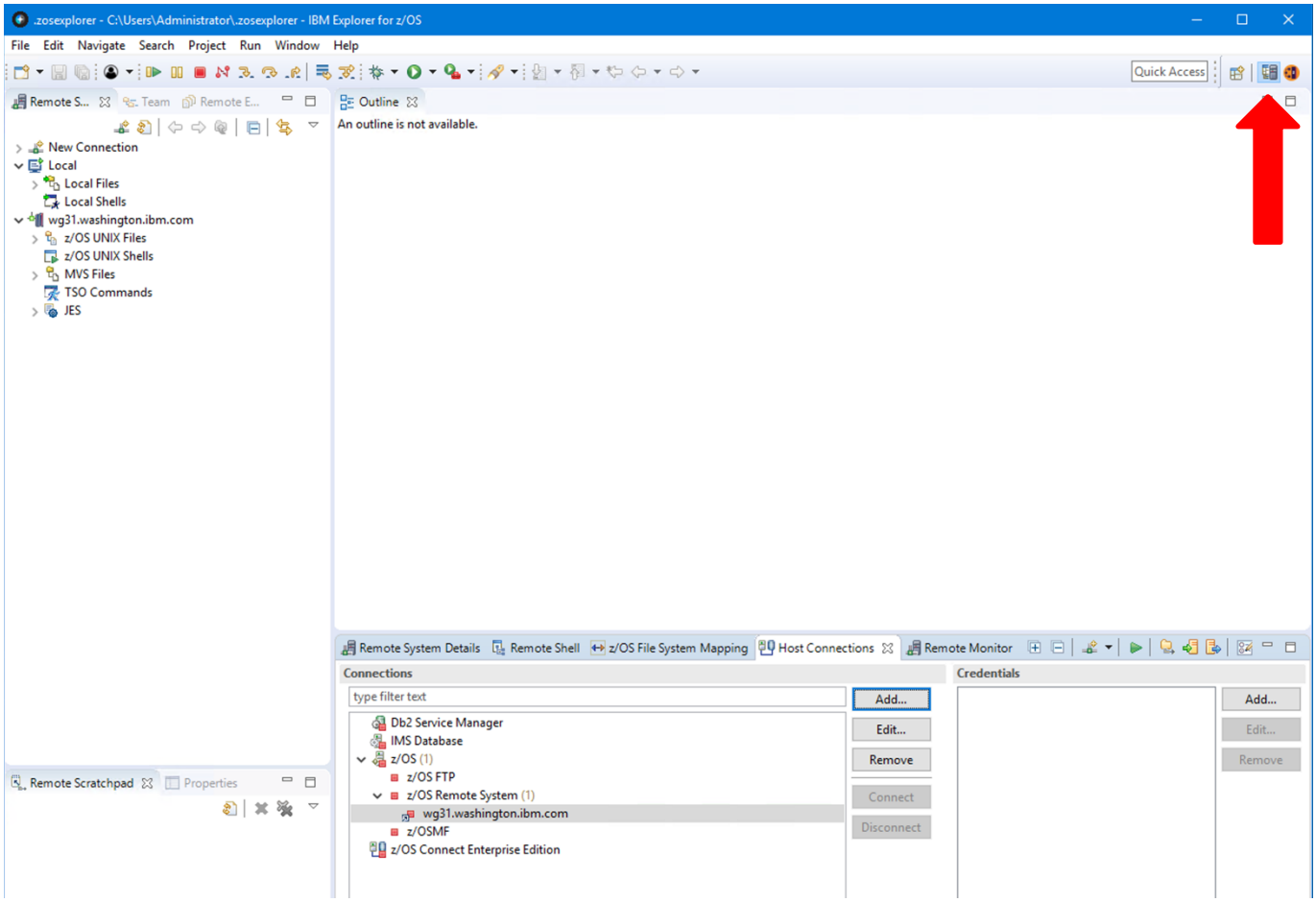
In this lab you have the option to create a Db2 Native REST service that uses an existing stored procedure, **or** a simple SQL statement, **or** both.

If you would like to create **both**, you must send two *separate* batch jobs by following and executing the steps below twice; once with the JCL calling the stored procedure, and the second time with the JCL for the simple SQL statement.

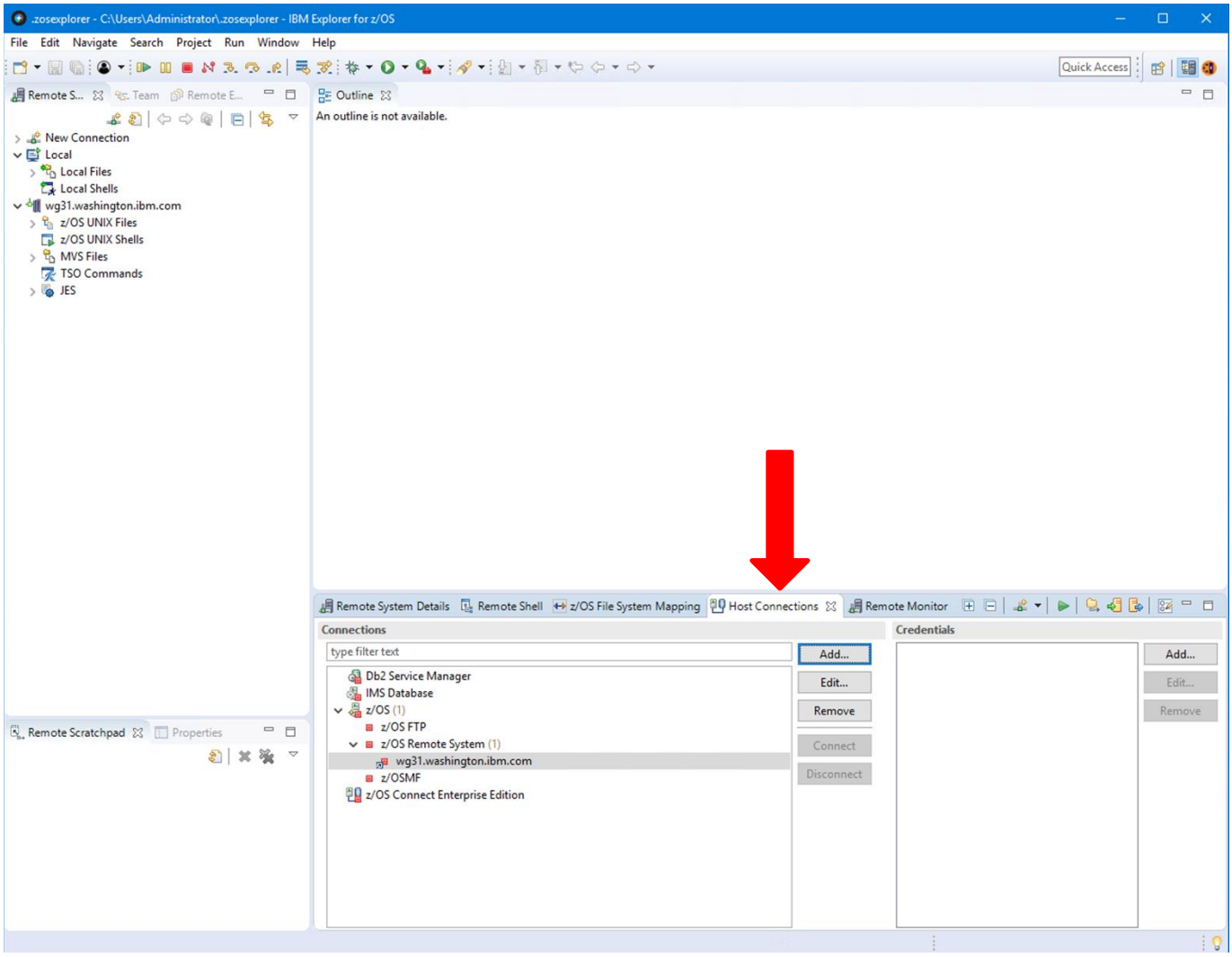
The following steps show you how to complete this using z/OS Explorer though using the provided JCL in a TSO 3270 emulator or green screen would also successfully create the services.

__1. Launch IBM z/OS Explorer.

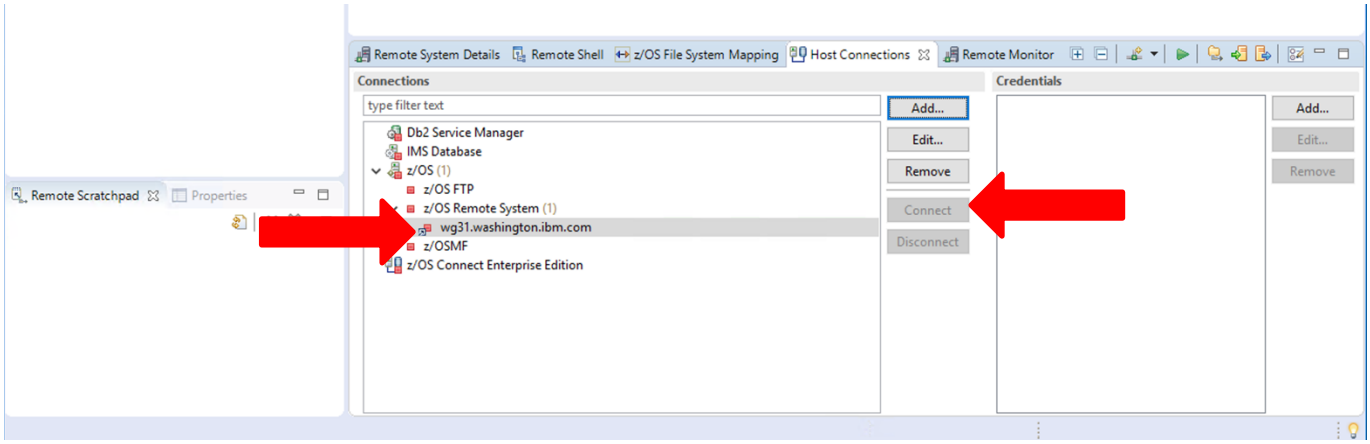
__2. When the IBM z/OS Explorer session launches, close the welcome window and switch to the Remote System Explorer perspective by clicking on the correct icon in the top right-hand corner of the window.



__3. In the bottom right-hand window, click on the Host Connections tab and connect to the host system to access the jobs needed to create the REST Services.



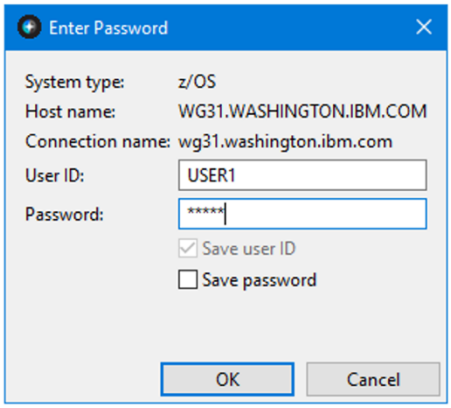
__4. Under z/OS Remote System, click on **wg31.washington.ibm.com** then click **Connect**



__6. In the Enter Password pop up window enter in the following credentials:

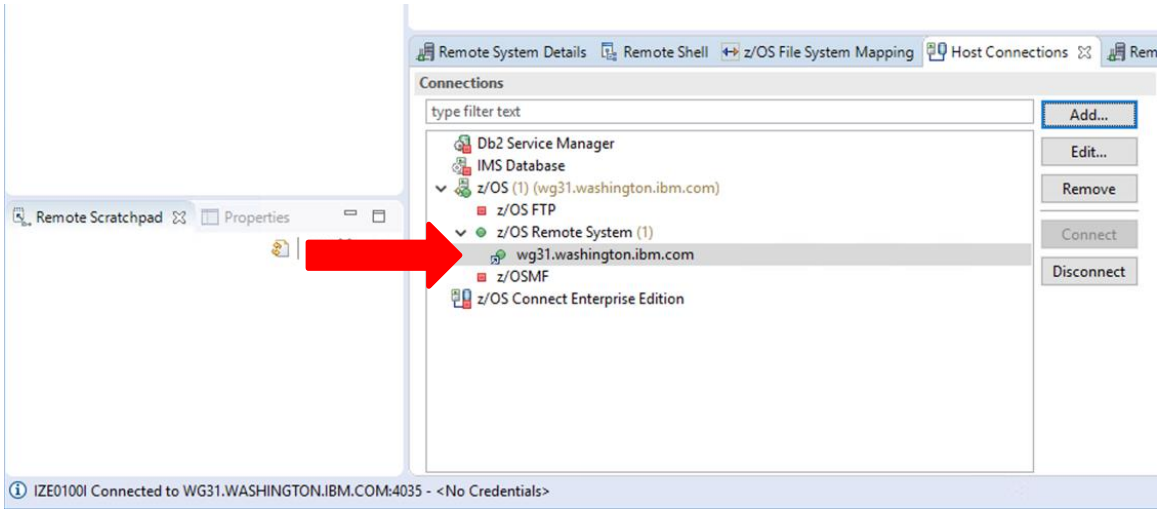
User ID: USER1

Password: USER1

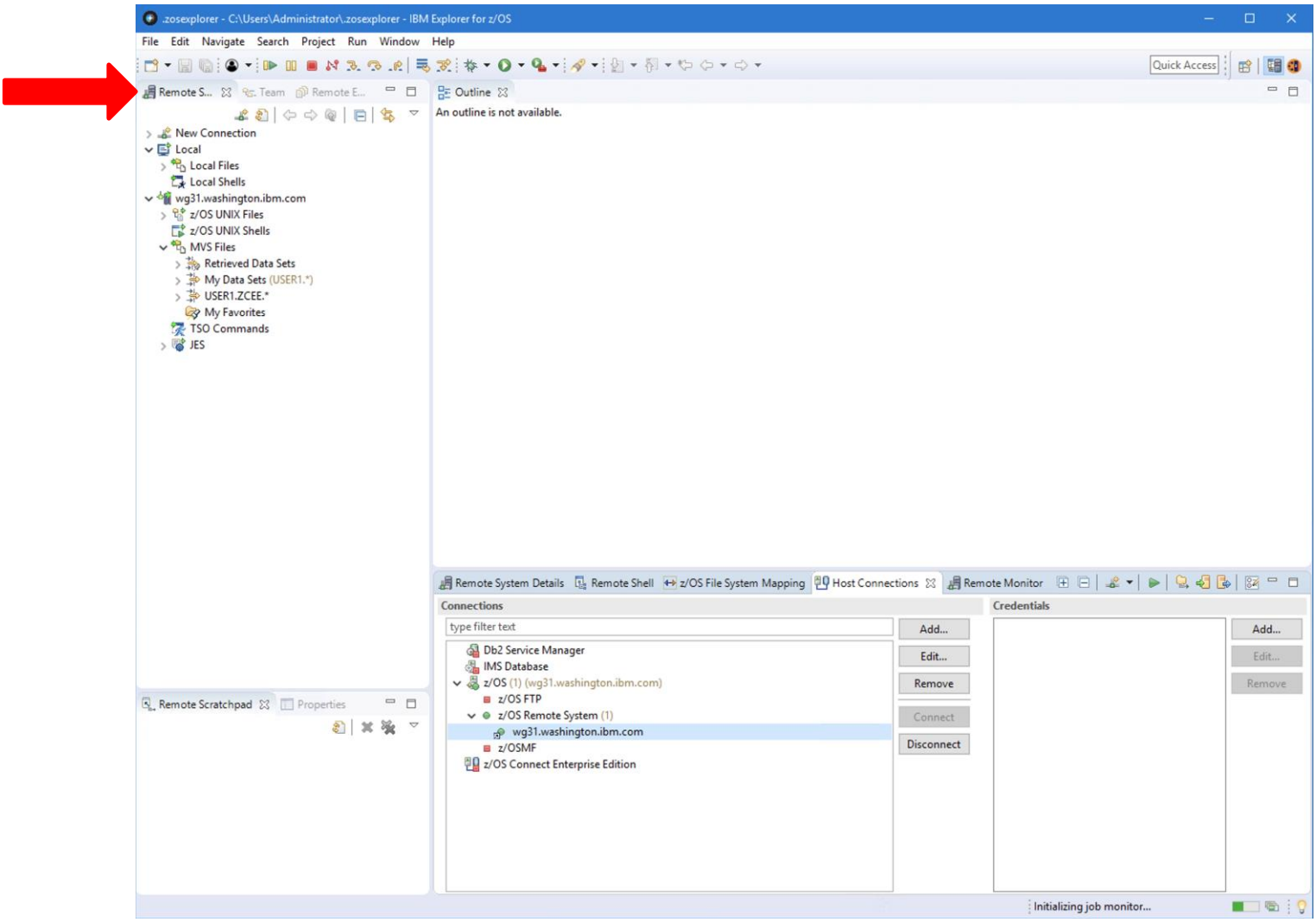


__7. Then click **OK**

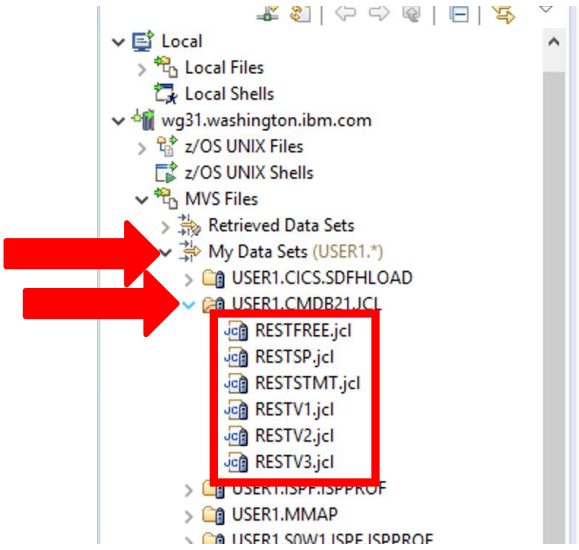
__8. Confirm that successful connection to the Remote System is indicated by the replacement of the red square by the green circle next to the host name in the Host Connections tab.



__9. Locate the **Remote Systems** tab in the upper left hand window of the z/OS Explorer.



__10. Under **MVS Files** expand **My Data Sets** and then the data set *USER1.CMDB21.JCL* to locate the jobs for creating the different Db2 REST Services.



__11. Now you can create any of the services using the provided JCL. To create a service calling the **stored procedure** Right click on RESTSP.jcl in the Remote Systems tab and select Submit. To create a service using a **SQL statement**, Right click on RESTSTMT.jcl in the Remote Systems tab and select Submit. Each of the jobs are listed below for your reference.

A. RESTSP.jcl**: selectByDeptSP**

```
//RESTSP   JOB MSGCLASS=H,CLASS=A,NOTIFY=&SYSUID,REGION=0M
//BIND EXEC PGM=IKJEFT01,DYNAMNBR=20
//STEPLIB  DD DSN=DSN1210.DB2.SDSNEXIT,DISP=SHR
//         DD DSN=DSN1210.DB2.SDSNLOAD,DISP=SHR
//SYSTSPRT DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//DSNSTMT  DD *
CALL EMPL_DEPTS_NAT(:whichQuery,:department1,:department2)
//SYSTSIN  DD *
DSN SYSTEM(DSN2)
BIND SERVICE("SYSIBMService") -
NAME("selectByDeptSP") -
SQLENCODING(1047) -
DESCRIPTION('Select employees by departments or department range')
/*
```

B. RESTSTMT.jcl: **selectByDeptSTMT**

```
//RESTSTMT JOB MSGCLASS=H,CLASS=A,NOTIFY=&SYSUID,REGION=0M
//BIND EXEC PGM=IKJEFT01,DYNAMNBR=20
//STEPLIB  DD DSN=DSN1210.DB2.SDSNEXIT,DISP=SHR
//         DD DSN=DSN1210.DB2.SDSNLOAD,DISP=SHR
//SYSTSPRT DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//DSNSTMT  DD *
 SELECT FIRSTNME,LASTNAME,PHONENO,WORKDEPT FROM
 DSN81210.EMP where WORKDEPT = :INDEPTNO
//SYSTSIN  DD *
DSN SYSTEM(DSN2)
BIND SERVICE("SYSIBMService") -
NAME("selectByDeptSTMT") -
SQLENCODING(1047) -
DESCRIPTION('Select employee based on department number.')
```

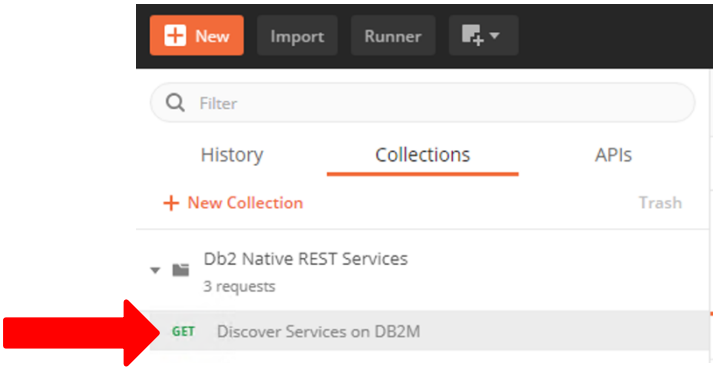
Expect a return code of 0 for successful completion.

1.3.1 *(Optional) Confirm the creation of the Db2 REST Services*

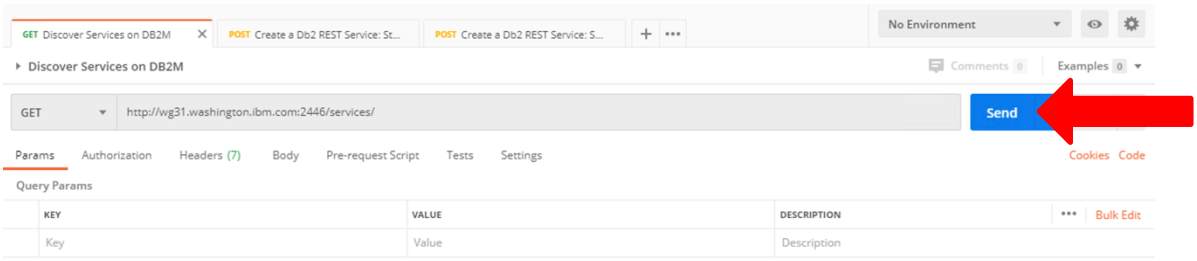
This section is optional because receiving a **200 OK** response code confirms the creation of your service. However, it is useful to "re-discover" the services installed on DB2M to see any updates made to the subsystem.

__1. Use the "Discover Services on DB2M" request again to confirm the creation of the Db2 service.

A. Open the "Discover Services on DB2M" request again in the "Db2 Native REST Services" collection from side navigation bar if you had previously closed the request.



B. Click **SEND** to invoke the command.



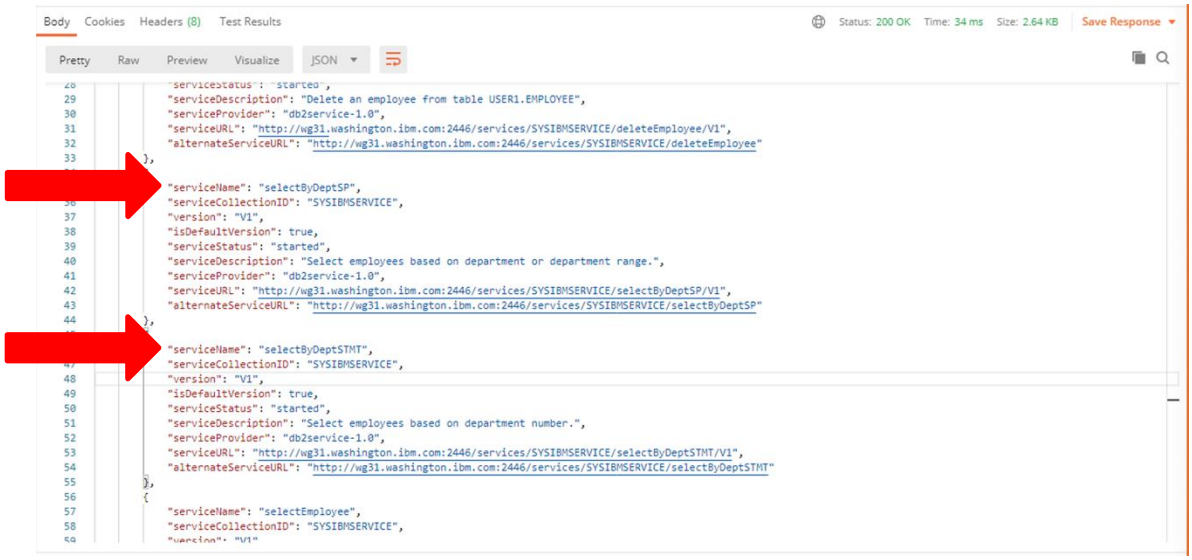
Troubleshooting

If the request fails, returning a status code of anything other than **200 OK**, go back to **section 1.2.2** and confirm that the request is filled out correctly; namely the correct REST Method and URI.

Also, read the StatusCode and StatusDescription to troubleshoot and debug the error.

Be careful when saving any changes made to the request.

C. Scroll down and confirm that the “selectByDeptSP” and/or “selectByDeptSTMT” is in the discover response output body.



The ServiceURL listed below (as well as above in the JSON) contains the URL used to display the JSON request and response schemas and execute the Db2 REST service. The service URL will be used section 1.4 to display the Db2 services’ metadata. You may enter the URL in Firefox (not as RESTClient) to view the schema.

Stored Procedure:

- http://wg31.washington.ibm.com:2446/services/SYSIBMService/selectByDeptSP
- The Uniform Resource Identifier (URI) is: /services/SYSIBMServiceselectByDeptSP

SQL Statement:

- http://wg31.washington.ibm.com:2446/services/SYSIBMService/selectByDeptSTMT
- The Uniform Resource Identifier (URI) is: /services/SYSIBMServiceselectByDeptSTMT

Summary: In this section, we confirmed the creation of the REST service using our previous “Discover Services on DB2M” request.

1.4 Execute the Db2 REST service

The first thing to do before executing the Db2 REST service is to review the JSON request and response schema information. The JSON request schema provides the information needed to execute the Db2 REST service, and the response schema provides the information being sent from Db2.

For the service using the stored procedure – "selectByDeptSP" – it is important to take note of a property of the *response* schema. A Db2 REST service that calls a stored procedure, returns a result set that Db2 calls a "dynamic anonymous result set". The reason there is an "anonymous result set" is because the Db2 stored procedure can provide various output results, which is determined based on runtime input. The term "anonymous result sets" indicates that in order to determine what the result set output will look like, the Db2 service must be executed and tested to definitively know the result set. You will view this property in **step 6A step ii** below. In contrast, you will not see this property in the *response* schema for the service using a simple SQL statement because the result set output is guaranteed based on the runtime input parameters.

Db2 SQL statements and stored procedures with output parameters will have their JSON response fields included in the JSON response schema, because that information is included in the Db2 catalog.

1.4.1 Display the Db2 Native REST service JSON schema information

In this section, regardless of which service you created – "selectByDeptSP" and/or "selectByDeptSTMT" – you will only create one request that can be used for each option. The request information required only differs in the **Db2 service URI**, which is detailed in **step 4B** below.

__1. Create a new request in the "Db2 Native REST Services" Collection, as detailed before in **section 1.2.2 step 1**.

__1. Name the request "Display service JSON schema" and click **Save to Db2 Native REST Services**.

__2. Open the newly created request by clicking on the "Display service JSON schema" Request in the side navigation bar under the "Db2 Native REST Services" Collection.

__3. Update the request with the information below to view the created service's JSON schema.

A. Set the REST Method to: **GET**

B. To view the JSON schema information for a specific service, the appropriate corresponding Db2 service URI must be used.

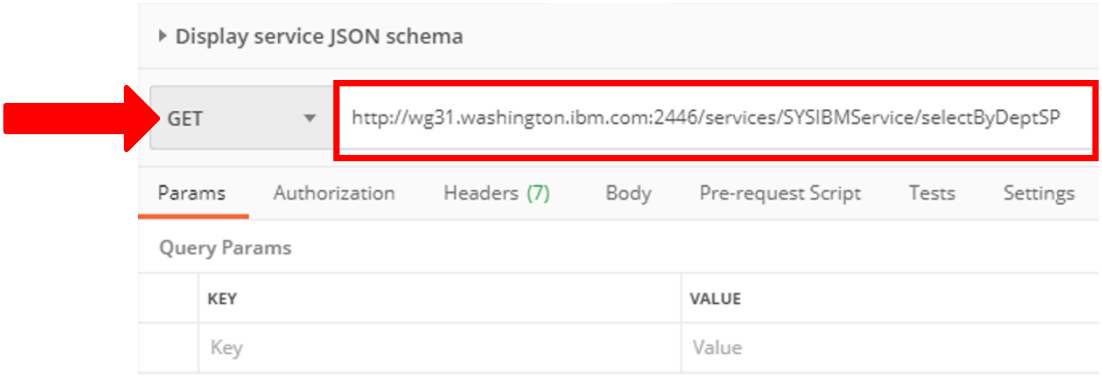
To view the JSON schema for the service calling a **Stored Procedure**, fill in the request body information with the URI found in **step i** below.
To view the JSON schema for the service calling a **SQL Statement**, fill in the request body information with the URI found in **step ii** below.

__i. **FOR STORED PROCEDURE**
http://wg31.washington.ibm.com:2446/services/SYSIBMService/selectByDeptSP

(1) Db2 DNS name = **wg31.washington.ibm.com**

(2) Db2 port = **2446**

(3) Db2 service URI = **/services/SYSIBMService/selectByDeptSP**

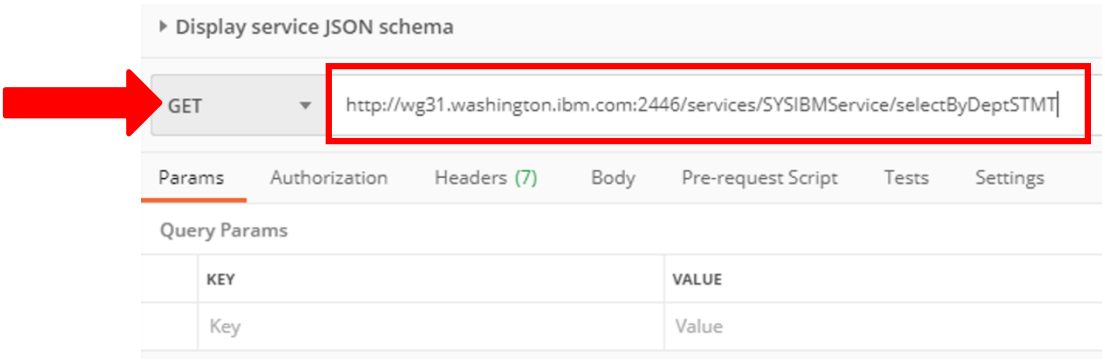


__ii. **FOR SQL STATEMENT**
http://wg31.washington.ibm.com:2446/services/SYSIBMService/selectByDeptSTMT

(1) Db2 DNS name = **wg31.washington.ibm.com**

(2) Db2 port = **2446**

(3) Db2 service URI = **/services/SYSIBMService/selectByDeptSTMT**



__4. Click **SEND** to invoke the command. If you created both services and would like to view both schemas, send one request, view the response. Then update the Db2 service URI with the other URI, and send the request again to view the response.

__5. The Db2 REST service schema information will be displayed in the response. Confirm the following output below for the services that you created.

The output for the service using the **Stored Procedure** is found in **step A** below.
The output for the service using the **SQL Statement** is found in **step B** below.

A. **FOR STORED PROCEDURE:** “selectByDeptSP” schema information

__i. Listed below are highlighted sections with the ***request*** schema fields and data types for the **stored procedure**. In order to send a successful REST request to Db2, this schema indicates what must be included in the request body in order to execute the service. Review the required fields and their expected corresponding data types below. You will use this information later to execute the service.

(1) WHICHQUERY = integer

(2) DEPT1 = string

(3) DEPT2 = string

From this information we can imagine what a request body should entail. For example, here is a sample JSON request that we could use to invoke the service:

{

"whichQuery": 2,

"department1":"A00",

"department2":"E00"

}

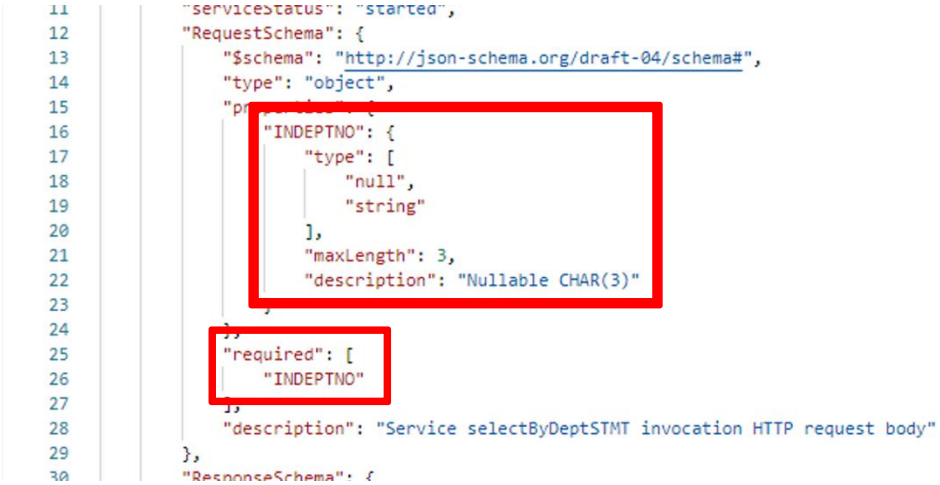
__ii. Scroll down further in the schema and notice the JSON ***response*** schema result set type, “Anonymous ResultSets”. As described at the beginning of the section, this indicates that the number of results sets to be returned is ambiguous, due to the fact that the results are dependent on the input parameters the stored procedure.

__iii. Reviewing other sections of the ***response*** schema, you will see what data will be returned from Db2 and how you will receive that data. Even though we know that the number of results sets will vary, due to the “Anonymous ResultSet” property, the result set output will be returned as an array of items that will be of the type “object”.

B. **SQL STATEMENT:** “selectByDeptSTMT” schema information

__i. Listed below are highlighted sections with the ***request*** schema field names and data types for the **SQL statement**. In order to send a successful REST request to the Db2, this schema indicates what must be included in the request body in order to execute the service. Review the required fields and their expected corresponding data types below. You will use this information later in the lab to execute the service.

(1) INDEPTNO = string



From this information we can imagine what a request body should entail. For example, here is a sample JSON request that we could use to invoke the service:

{"INDEPTNO": ”A00”}

__ii. Scroll down further in the schema and notice the JSON ***response*** schema result set type, **DOES NOT** include the type “Anonymous ResultSets”. As described at the beginning of the section, this is because this service uses a simple SQL statement and the result set output is known. Instead, the schema indicates that the requested set of data will be returned.

__iii. Reviewing other sections of the ***response*** schema, you will see what data will be returned from Db2 and how you will receive that data. Selecting an employee by department will return an array of object types that indicate the employee's first and last name, phone number, and work department.

Information

In Lab 2 you will see how to manipulate the Result Set Output for a service using the API editor when mapping the Response Output.

For example, you may want to omit sensitive employee information from the results.

__6. Upon reviewing the schema information for the service(s), you now know what is required to execute the service(s) successfully, and what the result set output should look like when it is returned.

__7. **SAVE** the changes made to the request

Summary: In this section, we viewed the JSON schema of the service(s) that we created in section 1.3.

1.4.2 Executing the Db2 REST service

In this exercise you will execute the Db2 REST service(s) that has(have) been created in **section 1.3**. If you created both services and would like to execute them both in this section, for organization and clarity, execute the following steps twice. Only at step 5 should you follow the specific instructions for your request.

For the create request, you will add two custom headers to the request. The headers indicate that the REST service is using JSON for the information transmitted. The Header Fields:

- The **Accept** field will be set to **application/json**
  - Defines that JSON will be used for the response
- The **Content-Type** field will be set to **application/json**
  - Defines that JSON will be used for the POST body

__1. Create a new request in the “Db2 Native REST Services” Collection, as detailed before in **section 1.2.2 step 1**.

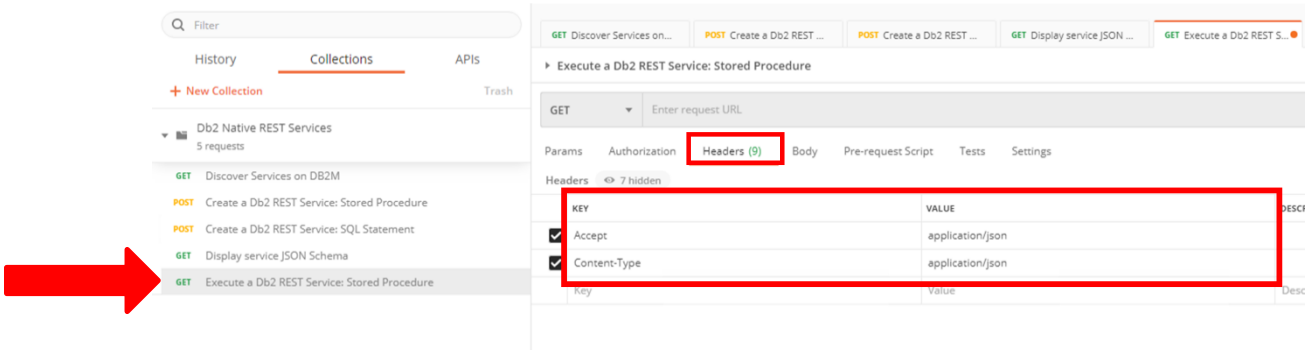
__2. For stored procedure, name the request “Execute a Db2 REST Service: Stored Procedure”.
For SQL statement, name the request “Execute a Db2 REST Service: SQL Statement”.

__3. Click **Save to Db2 Native REST Services**.

__4. Open the new request from the collection in the side navigation bar and navigate to the **Headers** tab to create the appropriate REST Header fields.

A. In the **Key** column type “Accept”, and in its corresponding **Value** column type “application/json”.

B. Add another header by typing “Content-Type” in the **Key** column, and in its corresponding **Value** column type “application/json”.



__5. If the service was created using a **stored procedure**, follow the steps in **step A**.
If the service was created using a **SQL statement**, follow the steps in **step B**.

A. **FOR STORED PROCEDURE:**
Update the request with the information below to execute the Db2 service we created in section 1.3. The required input parameters we reviewed in **section 1.4.1** resides in the JSON body.

__i. Set the REST Method to: **POST**

__ii. Add the Db2 REST URL:

http://wg31.washington.ibm.com:2446/services/SYSIBMService/selectByDeptSP

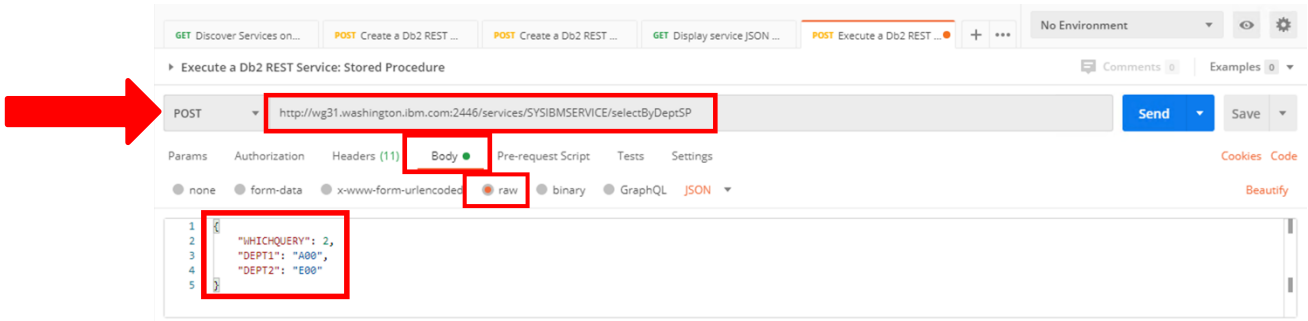
(1) Db2 DNS name = **wg31.washington.ibm.com**

(2) Db2 port = **2446**

(3) Db2 Service URI = **/services/SYSIBMService/selectByDeptSP**

__iii. Create the request JSON body by entering all the fields needed to execute the Db2 Service we created in section 1.3 (“selectByDeptSP”), provided below. Click on the **Body** tab and then the **raw** radio button.

```
{
"whichQuery": 2,
"department1": "A00",
"department2": "E00"
}
```



(1) The whichQuery variable is an integer, so no quotation marks are present.

(2) The fields department1 and department2 have string variables and include quotation marks.

Information

We know this is the information that the REST Service is expecting because we reviewed the JSON schema for **selectByDeptSP** in **section 1.4.1**.

An application developer would use the same process in order to use the service in their application.

B. **FOR SQL STATEMENT:**
Update the request with the information below to execute the Db2 service we created in section 1.3. The required input parameters we reviewed in **section 1.4.1** resides in the JSON body.

__i. Set the REST Method to: **POST**

__ii. Add the Db2 REST URL:

http://wg31.washington.ibm.com:2446/services/SYSIBMService/selectByDeptSTMT

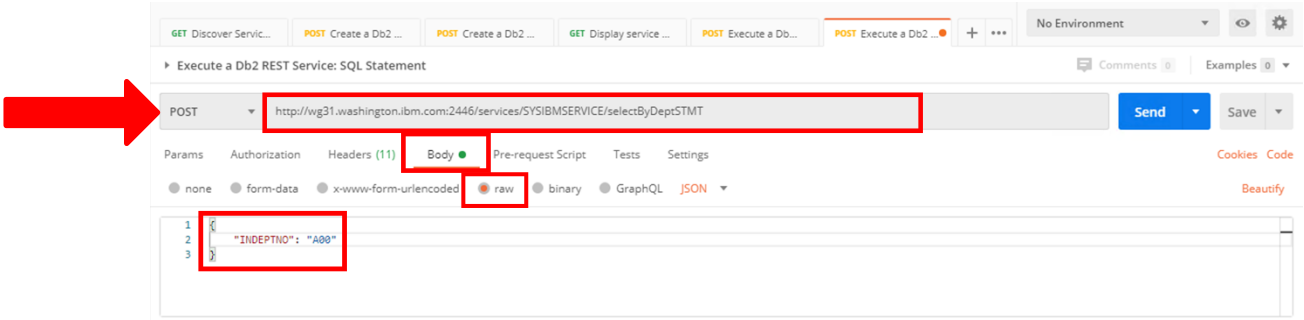
(1) Db2 DNS name = **wg31.washington.ibm.com**

(2) Db2 port = **2446**

(3) Db2 Service URI = **/services/SYSIBMService/selectByDeptSTMT**

__iii. Create the request JSON body by entering all the fields needed to execute the Db2 Service we created in section 1.3 (“selectByDeptSTMT”), provided below. Click on the **Body** tab and then the **raw** radio button.

```
{
"INDEPTNO": "A00"
}
```



(1) The INDEPTNO variable is a character string that maps to WORKDEPT.

Information

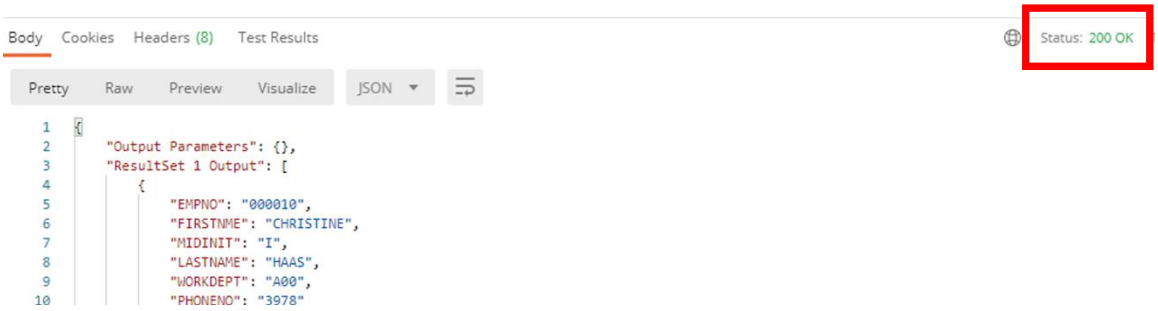
We know this is the information that the REST Service is expecting because we reviewed the JSON schema for **selectByDeptSTMT** in section 1.4.1.

An application developer would use the same process in order to use the service in their application.

__6. Whether you are executing the request using the provided stored procedure or a simple SQL statement, click **SEND** to invoke the command. In step 7 below, review the result set output corresponding to your service.

__7. Execute Service Response. Confirm the proper output below:

A. The first item for review is the status code should be **200 OK**. The status code **200 OK** will be the normal successful return code for Db2 services.

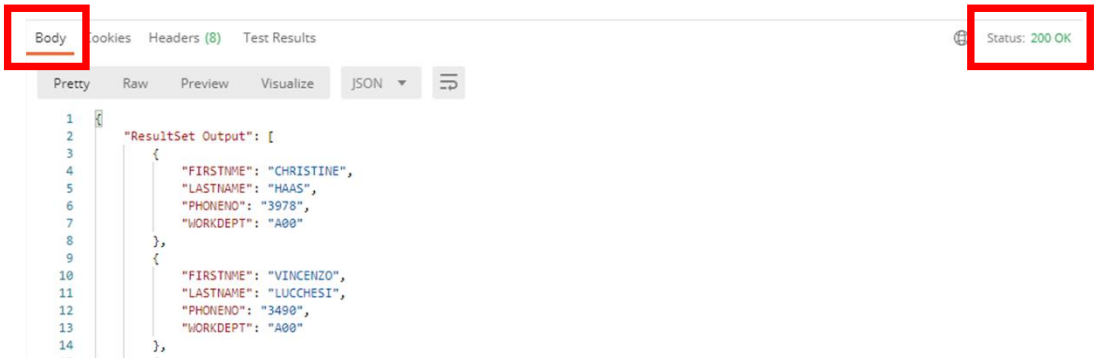


__i. The next item for review is the result set output that contains the employee information available in the response **Body** tab.

(1) **FOR STORED PROCEDURE**

- Scrolling down to review the entire response body, we can see that the employees belonging to the work departments within the range A00 to E00 have been retuned, which corresponds to the input parameters that we sent in the request body when we executed the service “selectByDeptSP”.

(2) **FOR SQL STATEMENT**



- Scrolling down to review the entire response body, we can see that the employees belonging to the work department A00 have been retuned, which corresponds to the input parameters that we sent in the request body when we executed the service “selectByDeptSTMT”.

__8. **SAVE** the changes to your request(s).

Summary: In this section we executed the REST Service(s) that we created in step 1.3.

In order to execute it properly, we viewed the JSON schema corresponding to the service created (selectByDeptSP and/or selectByDeptSTMT) and then executed the service in a request.

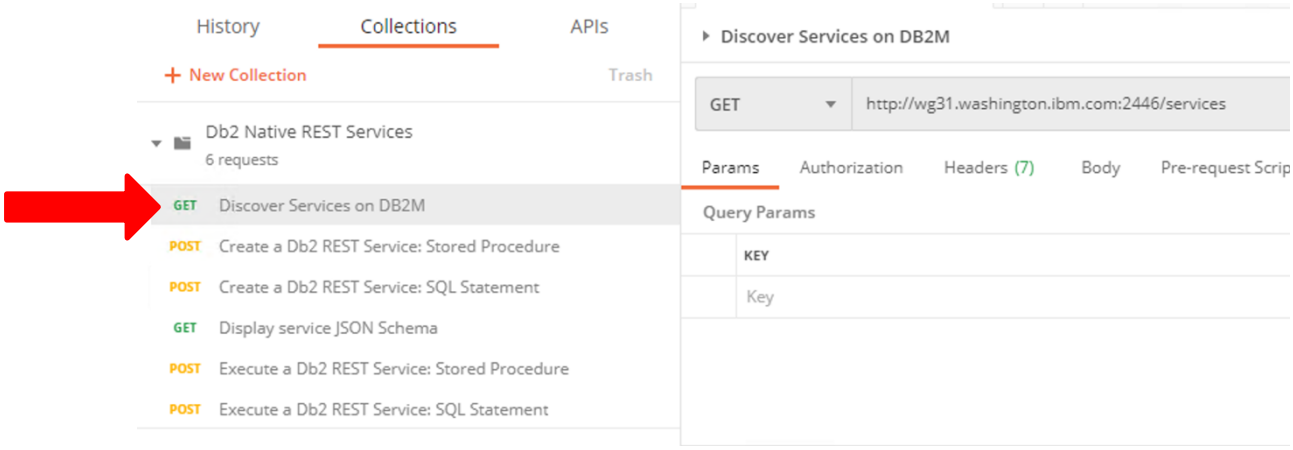
1.5 Versioning Db2 REST Services

In this section of the lab, we will explore a Db2 REST Service versioning example. A versioned Db2 Native REST Service – using a SQL statement – has been provided for you, which you will view and then create two new versions an examine the different output result sets. To understand the service, we will view the service's schema, execute the service, and then create the new services.

1.5.1 Discover the Versioned Service

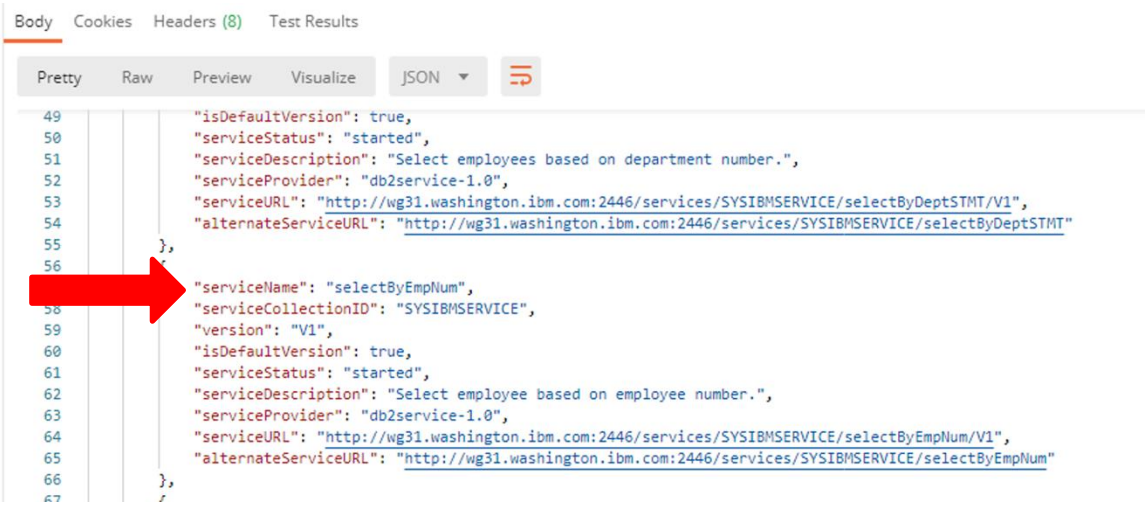
__1. Use the "Discover Services on DB2M" request again to discover the versioned service.

A. Open the "Discover Services on DB2M" request again in the "Db2 Native REST Services" collection from side navigation bar if you had previously closed the request.



B. Click **SEND** to invoke the command.

C. Scroll down and find the versioned service "selectByEmpNum" in the discover response output body. Review the version number and different parameters, next we will view the service schema located at the serviceURL.

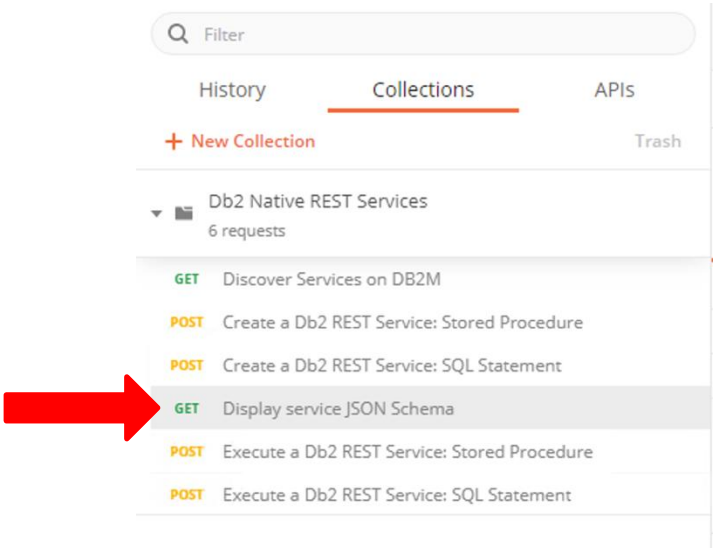


Summary: Here we quickly discovered the information for the provided service "selectByEmpNum".

1.5.2 View the JSON Schema of the Service

__1. Use the “Display service JSON schema” request again to learn more about the versioned service “selectByEmpNum”.

A. Open the “Display Services on DB2M” request again in the “Db2 Native REST Services” collection from side navigation bar if you had previously closed the request.



__2. Update the request with the information below to view the versioned service’s JSON schema.

A. The REST Method should remain: **GET**

B. Add the “selectByEmpNum” serviceURL:

http://wg31.washington.ibm.com:2446/services/SYSIBMService/selectByEmpNum/V1

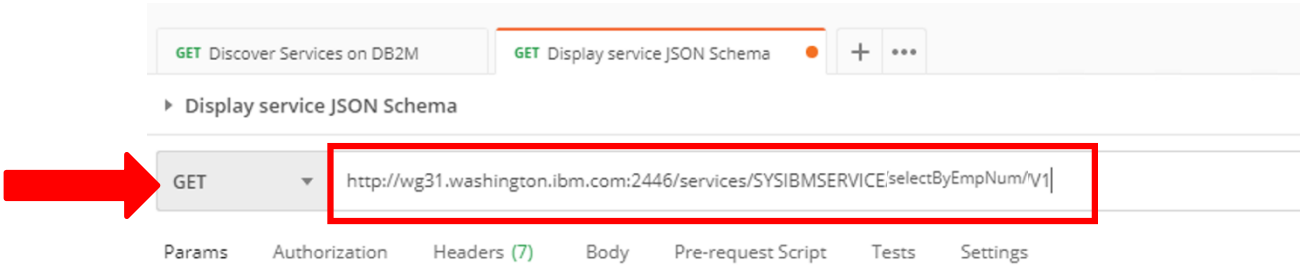
__i. Db2 DNS name = **wg31.washington.ibm.com**

__ii. Db2 port = **2446**

__iii. Db2 service URL = **/services/SYSIBMService/selectByEmpNum/**

__iv. Service version = **V1**

Notice that the version number – **V1** – is included. If this parameter is not included in the URL, then the default version of the service will be used, which is indicated by the “**isDefaultVersion**” parameter from the Discover result set output in section 1.5.1 step 1C.

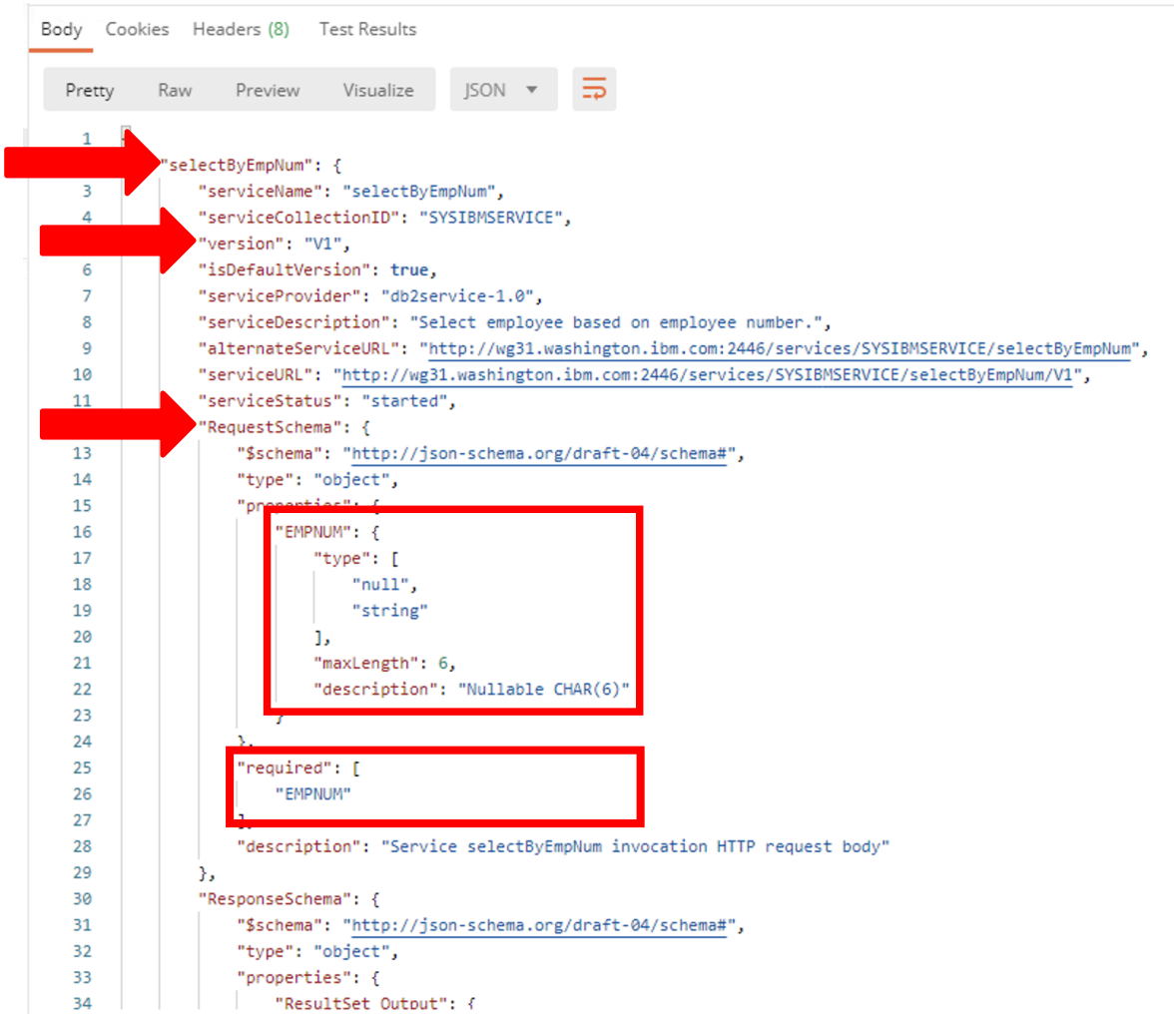


__3. Click **SEND** to invoke the command.

__4. The Db2 REST service schema information will be displayed in the response. Confirm the following output below for the services that you created.

A. Listed below are highlighted sections with the ***request*** schema fields and data types for the **selectByEmpNum/V1** service. In order to send a successful REST request to Db2, this schema indicates what must be included in the request body in order to execute the service. Review the required fields and their expected corresponding data types below. You will use this information later to execute the service.

__i. EMPNUM = string



From this information we can imagine what a request body should entail. For example, here is a sample JSON request that we could use to invoke the service:

{

"EMPNUM": “000010”

}

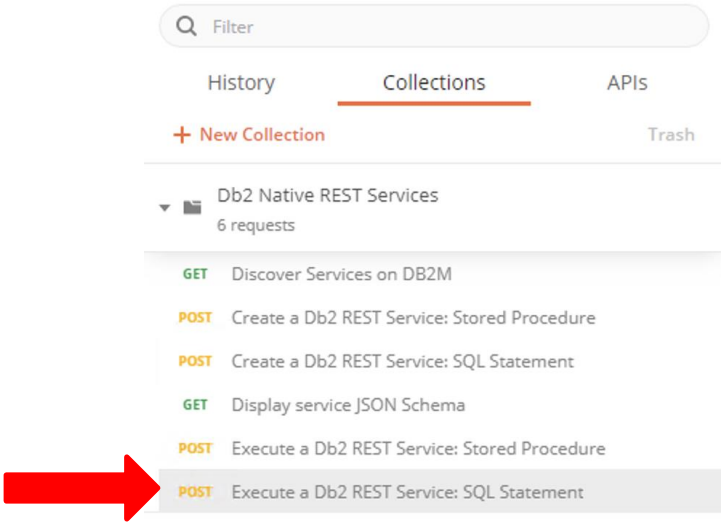
B. Reviewing other sections of the ***response*** schema, you will see what data will be returned from Db2 and how you will receive that data.

Summary: In this section, we reviewed the JSON schema for the service selectByEmpNum, to understand how to execute the service and see how the data will be returned in the response.

1.5.3 Execute the Versioned Service

__1. To execute the versioned service, you may use either of the “Execute a Db2 REST Service: …” requests – Stored Procedure or SQL Statement.

A. Open one of the “Execute a Db2 REST Service: …” requests again in the “Db2 Native REST Services” collection from side navigation bar if you had previously closed the request.



__2. Update the request with the information below to execute the versioned service.

A. The REST Method should remain: **POST**

B. Add the “selectByEmpNum” serviceURL:

http://wg31.washington.ibm.com:2446/services/SYSIBMService/selectByEmpNum/V1

__i. Db2 DNS name = **wg31.washington.ibm.com**

__ii. Db2 port = **2446**

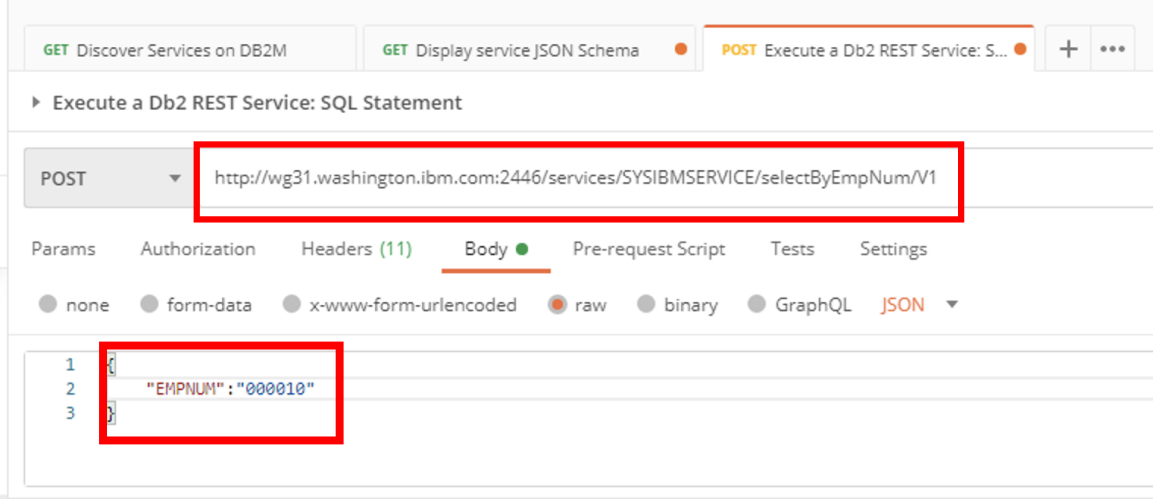
__iii. Db2 service URL = **/services/SYSIBMService/selectByEmpNum/**

__iv. Service version = **V1**

Notice that the version number – **V1** – is included. If this parameter is not included in the URL, then the default version of the service will be used, which is indicated by the “**isDefaultVersion**” parameter from the Discover result set output in **section 1.5.1 step 1C**.

C. Add the service’s required parameters to the JSON Body of the request:

```
{
"EMPNUM": "000010"
}
```

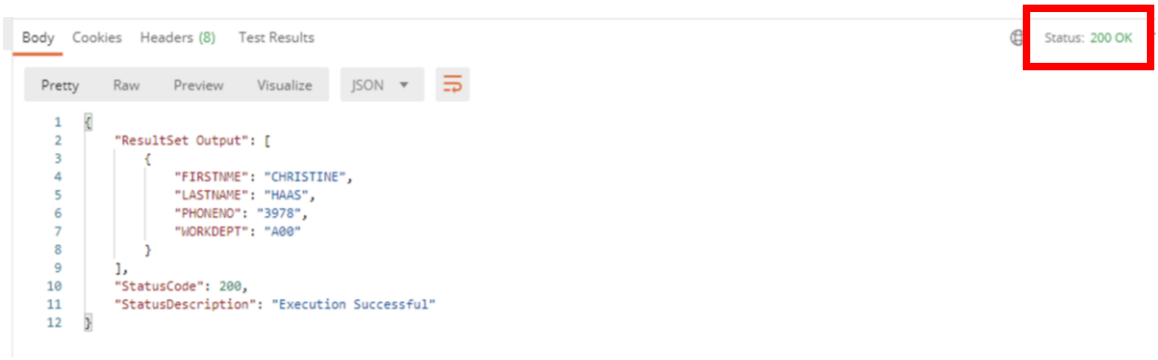


(1) The EMPNUM variable is a character string that maps to EMPNO.

__3. Click **SEND** to invoke the request.

__4. Execute Service Response. Confirm the proper output below:

A. The first item for review is the status code should be **200 OK**. The status code **200 OK** will be the normal successful return code for Db2 services.



Troubleshooting

If the request fails, returning a status code of anything other than **200 OK**, confirm that the request is filled out correctly; namely the correct Method, serviceURL, JSON body, and Header information.

The header information should have been saved from completing **section 1.4.2**.

Also, read the StatusCode and StatusDescription to troubleshoot and debug the error.

B. The next item for review is the result set output that contains the employee information available in the response **Body** tab.

__i. Upon executing the service we can see that employee information pertaining to the employee number that was sent to Db2 with the REST request was returned. In the following section, we will create new versions of this same service such that more information will be returned, by selecting data with more complex SQL statements.

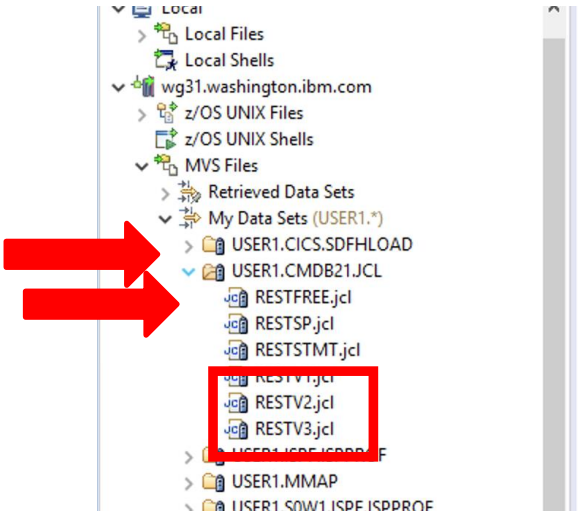
Summary: In this section we executed the service that was provided – selectByEmpNum – to understand the service's default version behavior, and result set output. In the following section we will create two new versions, that will expand upon this result set.

1.5.4 Create new Versions of the Service

__1. To create a new version of the “selectByEmpNum” service, use the provided JCL to submit batch jobs in the same way you completed section 1.3.

A. Navigate back to IBM z/OS Explorer

B. Locate the JCL for the versioned services in the Remote Systems Tab.



__2. Right click on RESTV2.jcl and RESTV3.jcl and click Submit to create the services. Review the JCL before submitting to understand the differences between each version. The JCL has been provided below for reference.

A. RESTV2.jcl: **selectByEmplNum.V2**

```
//RESTV2 JOB MSGCLASS=H,CLASS=A,NOTIFY=&SYSUID,REGION=0M
//BIND EXEC PGM=IKJEFT01,DYNAMNBR=20
//STEPLIB  DD DSN=DSN1210.DB2.SDSNEXIT,DISP=SHR
//         DD DSN=DSN1210.DB2.SDSNLOAD,DISP=SHR
//SYSTSPRT DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//DSNSTMT  DD *
SELECT E.FIRSTNME, E.LASTNAME, E.PHONENO, E.WORKDEPT,
  M.LASTNAME AS MANAGER FROM DSN81210.EMP E, DSN81210.EMP M,
  DSN81210.DEPT D WHERE E.EMPNO = :EMPNUM and E.WORKDEPT = D.DEPTNO
  and D.MGRNO = M.EMPNO
//SYSTSIN  DD *
DSN SYSTEM(DSN2)
BIND SERVICE("SYSIBMService" ) -
NAME("selectByEmpNum") -
SQLENCODING(1047) -
VERSION(V2) -
DESCRIPTION('Select employee based on employee number, includes the emp-
loyee''s manager.')
```

B. RESTV3.jcl: **selectByEmpNum.V3**

```
----+----1----+----2----+----3----+----4----+----5----+----6----+---
//RESTV3 JOB MSGCLASS=H,CLASS=A,NOTIFY=&SYSUID,REGION=0M
//BIND EXEC PGM=IKJEFT01,DYNAMNBR=20
//STEPLIB  DD DSN=DSN1210.DB2.SDSNEXIT,DISP=SHR
//         DD DSN=DSN1210.DB2.SDSNLOAD,DISP=SHR
//SYSTSPRT DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//DSNSTMT  DD *
SELECT E.FIRSTNME, E.LASTNAME, E.PHONENO, E.WORKDEPT,
  M.LASTNAME AS MANAGER, M.PHONENO AS MGRPHONE FROM DSN81210.EMP
  E, DSN81210.EMP M, DSN81210.DEPT D WHERE E.EMPNO = :EMPNUM AND
  E.WORKDEPT = D.DEPTNO and D.MGRNO = M.EMPNO
//SYSTSIN  DD *
DSN SYSTEM(DSN2)
BIND SERVICE("SYSIBMService" ) -
NAME("selectByEmpNum") -
SQLENCODING(1047) -
VERSION(V3) -
DESCRIPTION('Select employee based on employee number, includes the emp-
loyee''s manager.')
```

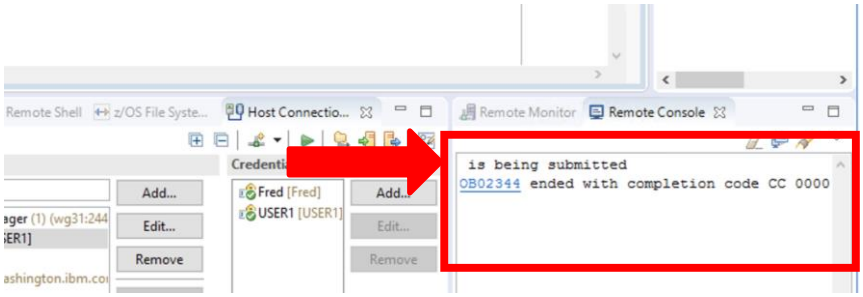
wg31

Information

Notice that the serviceName is **the same as** the provided REST service “selectByEmpNum”. If versioning was NOT enabled, the service name would have to be UNIQUE in order for successful creation of the service.

To demonstrate the versioning capabilities of Db2 REST services, make sure that the serviceName is the same and that the **version parameter** is **unique**.

__3. A completion code of 0 is expected for successful submission.

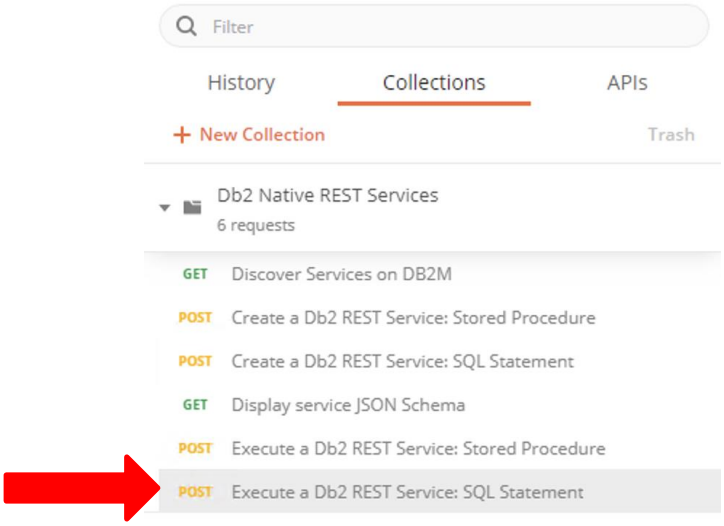


Summary: In this section we created two new versions of the existing service “selectByEmpNum” using the provided JCL. Version 2 returns the employee’s manager in the result set, and version 3 includes the employee’s manager and the manager’s phone number in the result set. In the following section we will see how these services compare when executed.

1.5.5 Execute the new service versions.

__1. To execute the versioned services, you may use either of the “Execute a Db2 REST Service: …” requests – Stored Procedure or SQL Statement.

A. Open one of the “Execute a Db2 REST Service: …” requests again in the “Db2 Native REST Services” collection from side navigation bar if you had previously closed the request.



__2. Update the request with the information below to execute the second version of the “selectByEmpNum” service that includes the employee’s manager in the response.

A. The REST Method should remain: **POST**

B. Add the “selectByEmpNum” serviceURL:

http://wg31.washington.ibm.com:2446/services/SYSIBMService/selectByEmpNum/V2

__i. Db2 DNS name = **wg31.washington.ibm.com**

__ii. Db2 port = **2446**

__iii. Db2 service URL = **/services/SYSIBMService/selectByEmpNum/**

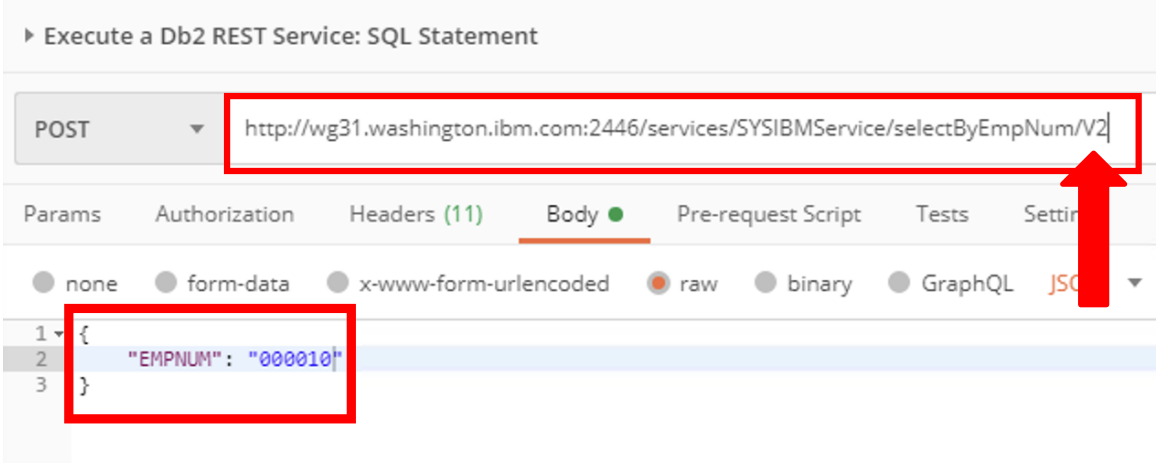
__iv. Service version = **V2**

*****DO NOT FORGET TO UPDATE THE VERSION NUMBER*****

If this parameter is not included in the URL, then the default version of the service will be executed, which is indicated by the “**isDefaultVersion**” parameter from the Discover result set output in section 1.5.1 step 1C.

C. Add the service’s required parameters to the JSON Body of the request:

```
{
"EMPNUM": "000010"
}
```



(1) The EMPNUM variable is a character string that maps to EMPNO.

__3. Click **SEND** to invoke the request.

__4. Execute Service Response. Confirm the proper output below:

A. The first item for review is the status code should be **200 OK**. The status code **200 OK** will be the normal successful return code for Db2 services.

B. The next item for review is the result set output that contains the employee information available in the response **Body** tab.

__i. Upon executing the service, we can see that now the manager's name is included in the result set output when compared to the execution of the original service V1.

__5. Upon successful execution of the second version of the service, execute the third version of the service that includes the manager's phone number in the response. In the "Execute a Db2 REST Service: SQL Statement" request, update the fields with the following information:

A. The REST Method should remain: **POST**

B. Add the "selectByEmpNum" serviceURL:

http://wg31.washington.ibm.com:2446/services/SYSIBMService/selectByEmpNum/V3

__i. Db2 DNS name = **wg31.washington.ibm.com**

__ii. Db2 port = **2446**

__iii. Db2 service URL = **/services/SYSIBMService/selectByEmpNum/**

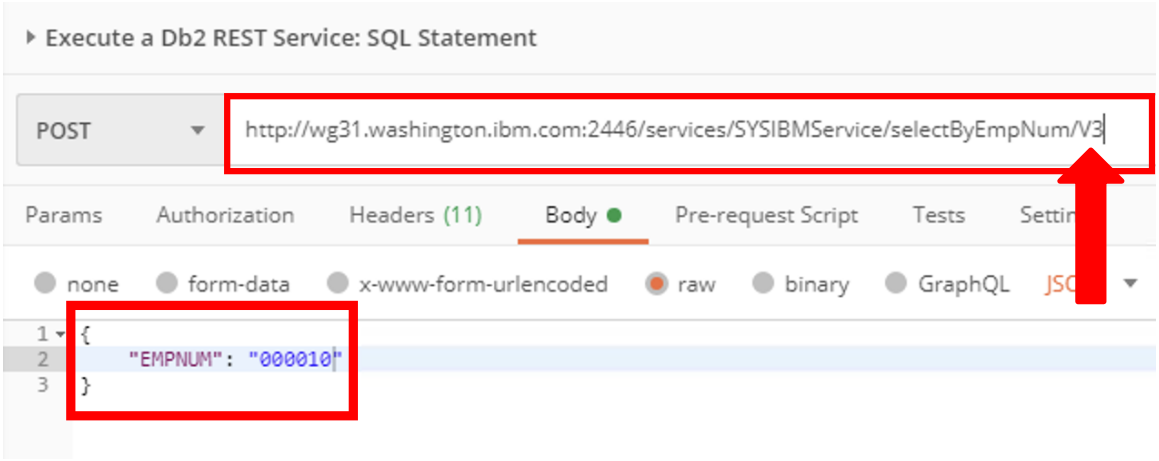
__iv. Service version = **V3**

*****DO NOT FORGET TO UPDATE THE VERSION NUMBER*****

If this parameter is not included in the URL, then the default version of the service will be executed, which is indicated by the "**isDefaultVersion**" parameter from the Discover result set output in **section 1.5.1 step 1C**.

C. Add the service's required parameters to the JSON Body of the request:

```
{
"EMPNUM": "000010"
}
```



(1) The EMPNUM variable is a character string that maps to EMPNO.

__6. Click **SEND** to invoke the request.

__7. Execute Service Response. Confirm the proper output below:

A. The first item for review is the status code should be **200 OK**. The status code **200 OK** will be the normal successful return code for Db2 services.

B. The next item for review is the result set output that contains the employee information available in the response **Body** tab.

__i. Upon executing the service, we can see that now the manager's name and phone number is included in the result set output when compared to the execution of the original service V1.

Summary: In this section we executed the newly created versions of the service that was provided – selectByEmpNum. By employing this technique, there can be multiple services with the same name that perform similar tasks depending on what the desired output is.

1.6 Summary

During this lab, you learned how quickly a stored procedure and/or a SQL statement could be used to create a Native Db2 REST Service. Upon reviewing the Db2 system information and the sample Db2 Stored Procedure, you accomplished the following using the REST client – Postman – to manage, view, and execute the Native REST Services:

1. Used the discover API to view the installed services on DB2M.
2. Created a service using a stored procedure and/or a SQL statement.
3. Viewed the service(s) schema(s) to understand the required parameters needed to execute the service(s), as well as understand how the service would behave when called.
4. Executed the service using the information available in the service(s) schema(s).
5. Finally, explore the versioning capability of Db2 REST Services.

Lab 2 Creating a Db2 REST API in z/OS Connect EE

In this exercise, you will use z/OS Connect EE to create a Db2 REST API from one of the Db2 REST services that you created in Lab 1. If you did not complete Lab 1, please go back in order to move forward. There are four steps to create a Db2 REST API in z/OS Connect EE, listed below is a summary of the steps covered in the following exercises:

a) Import the Db2 service from the Db2Service Manager.

b) The z/OS Connect EE includes a tool called “z/OS Connect Build Toolkit”, which is used to create a service archive file (SAR). You will use the z/OS Connect Build Toolkit (zconbt) to create a service archive file (SAR) from the Db2 service’s JSON schemas.

c) Define the Db2 REST service to z/OS Connect EE.

d) Use the API Editor to create a Db2 REST API and deploy the new REST API to z/OS Connect EE. This step will also create the swagger file, which helps with application development.

e) Test the new Db2 REST API.

2.1 Create the Db2 REST API and deploy it to z/OS Connect EE

User defined Db2 z/OS native REST services are invoked using **only** the POST method. The sample Db2 REST service selectEmployee only supports reading data, and you will use the zCEE API Editor to map the Db2 POST method to a more appropriate GET method. You will also learn how to map input data in the REST call to the Db2 REST service, avoiding the need for a JSON request body. zCEE does not use JSON request body, that is used for native REST only.

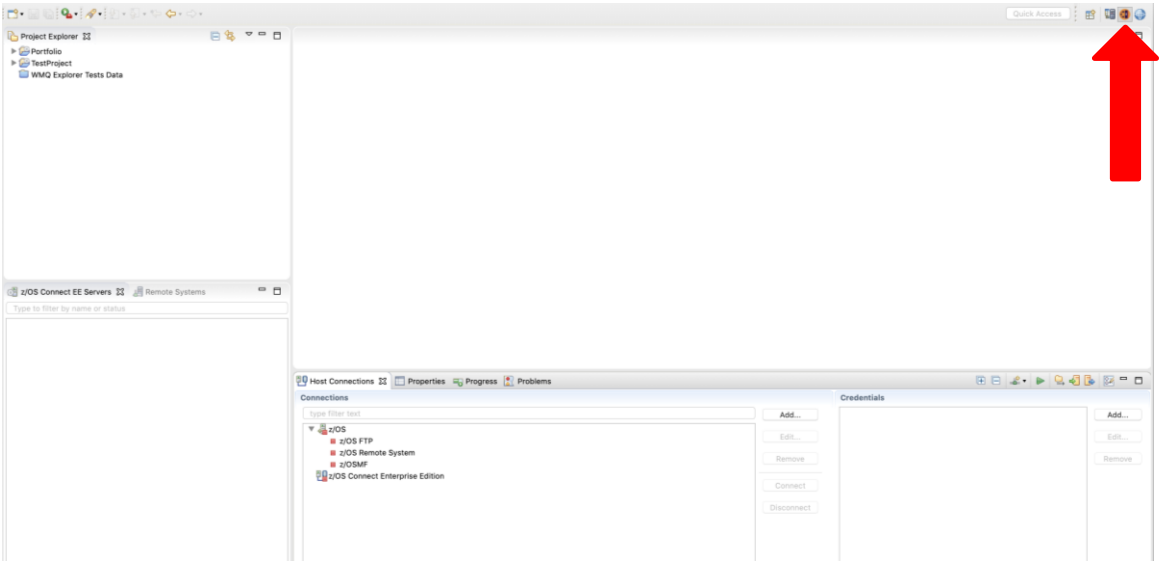
This exercise covers the following topics:

- Create a Db2 REST Service Project
- Create a Service Archive File (.sar)
- Deploy .sar file
- Create a Db2 REST API project
- Map a POST method to a GET method
- Use the Request Mapping Editor to define the JSON request schema fields
  - Define input variable mapping to the JSON request schema
  - Configure field translation
- Deploy selectemployee REST API to z/OS Connect EE
- Test the selectemployee Db2 REST API

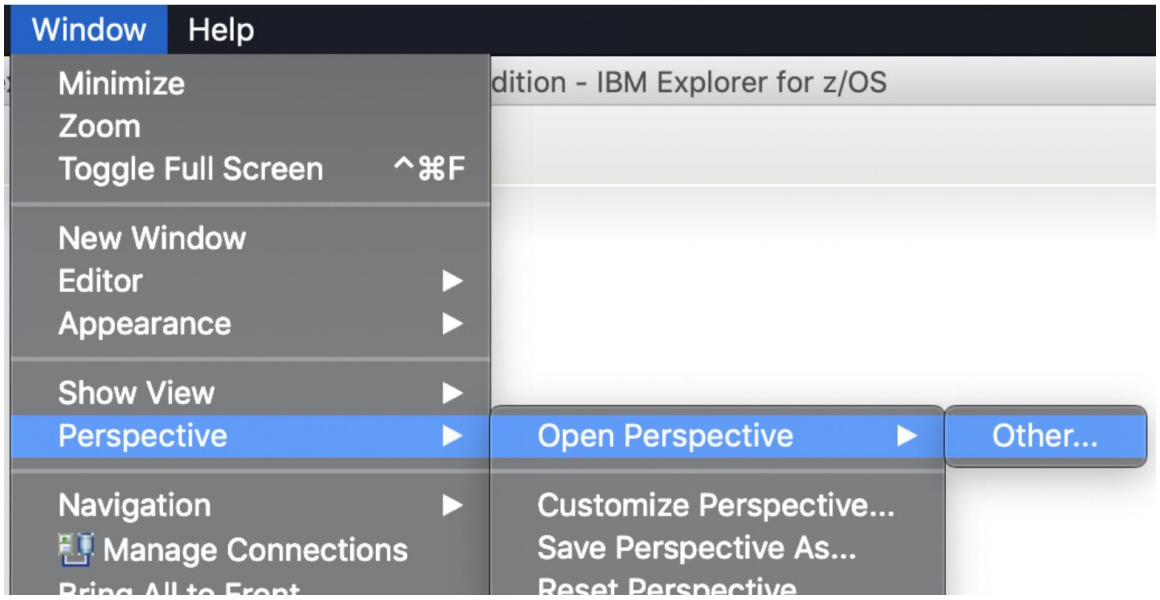
2.1.1 Add a zCEE Connection, and a Db2 Service Manager Connection

Your copy of z/OS Explore includes the “z/OS Connect Enterprise Edition” and the “API Editor”. This step will add a zCEE Connection to z/OS Explorer and Create the Db2 REST API project. The screen format is similar to the one used in Data Studio and Workload Query Tuner.

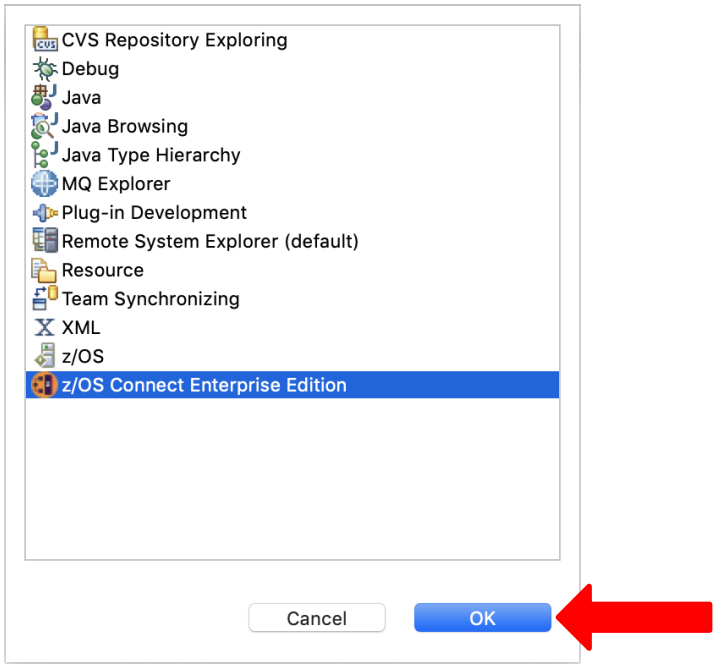
__1. Change z/OS Explorer’s Eclipse perspective from “Remote System Explorer” to “z/OS Connect Enterprise Edition”.



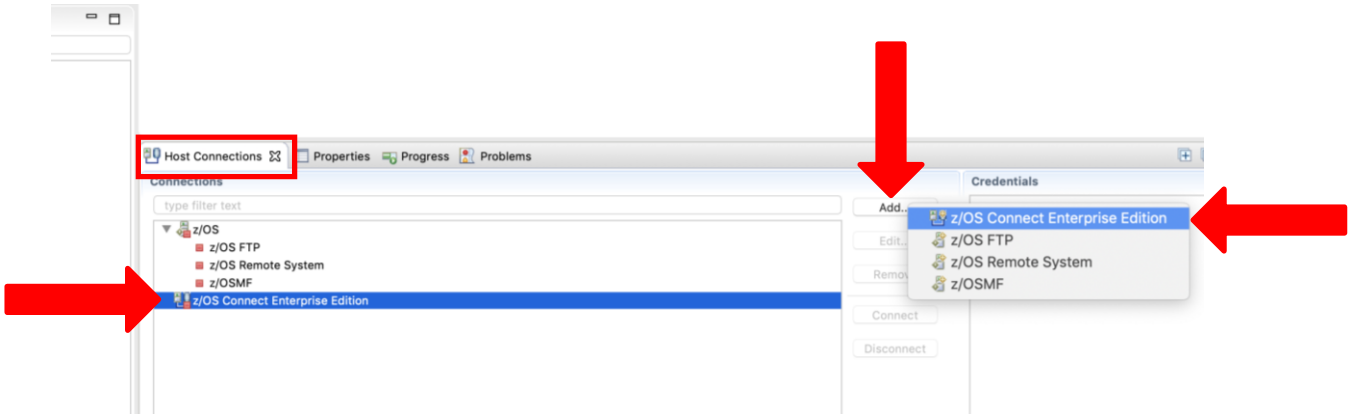
A. If you do not see the z/OS Connect Enterprise Edition perspective button, using the menu bar, select “Window” and navigate to “Perspective” > “Open Perspective” > “Other”.



B. Select “z/OS Connect Enterprise Edition” and click **OK**.



__2. We will first establish a connection for zCEE. Add your connection by clicking on “Host connections”, then “z/OS Connect Enterprise Edition”, the **Add** button, and then “z/OS Connect Enterprise Edition”.



__3. Adding a z/OS Connect Enterprise Edition Connection

A. First you will be prompted for the Host Name, which is the same as the LPAR address. In the “Add z/OS Connect Enterprise Edition Connection” Window that popped up enter the following information:

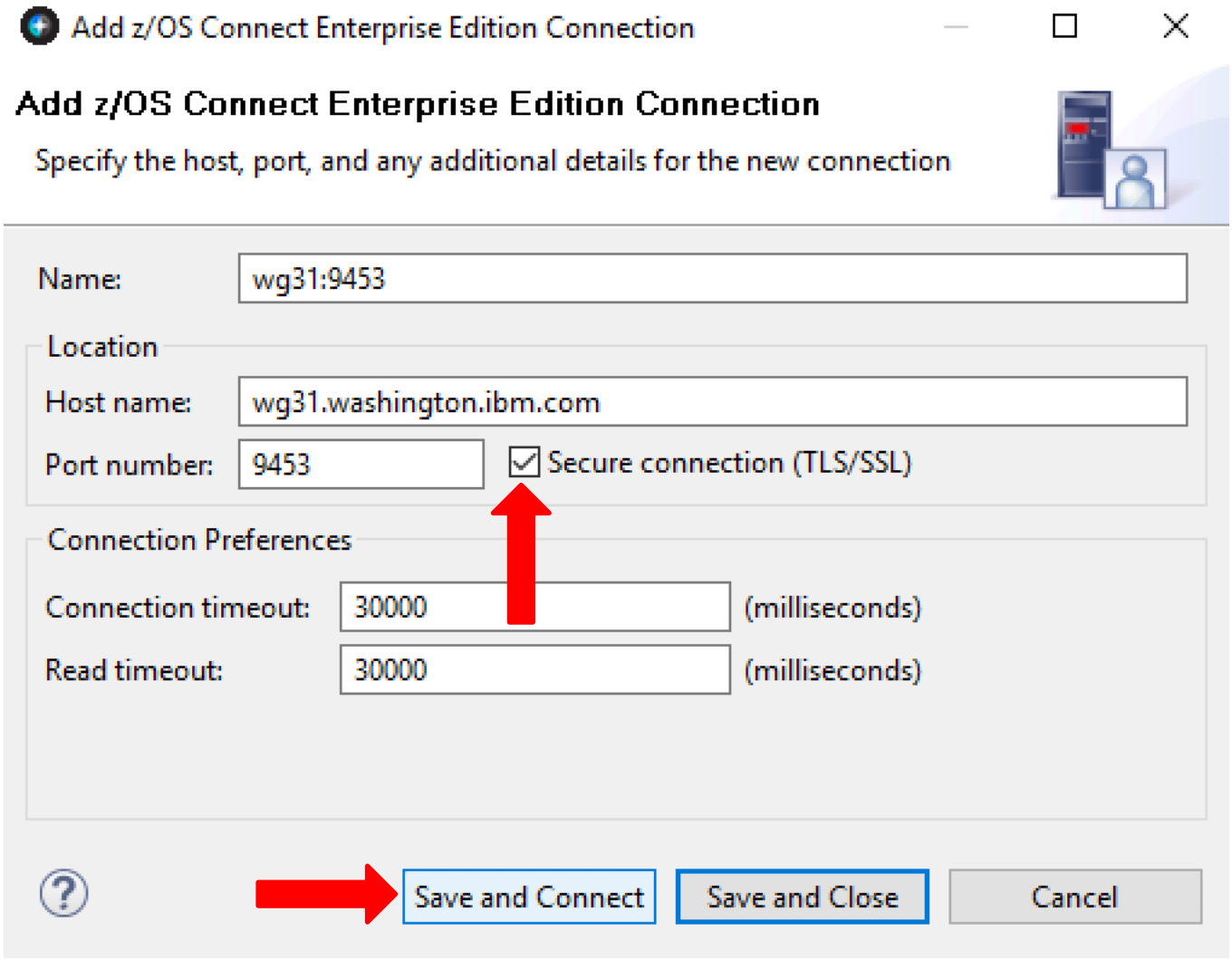
__i. Host name: **wg31.washington.ibm.com**

__ii. Port number: **9453**

__iii. Name: **wg31:9453**

__**iv. Check the checkbox “Secure connection (TLS/SSL)”**

If this checkbox is NOT checked, your API WILL NOT WORK



__4. Finally, click **Save and Connect**

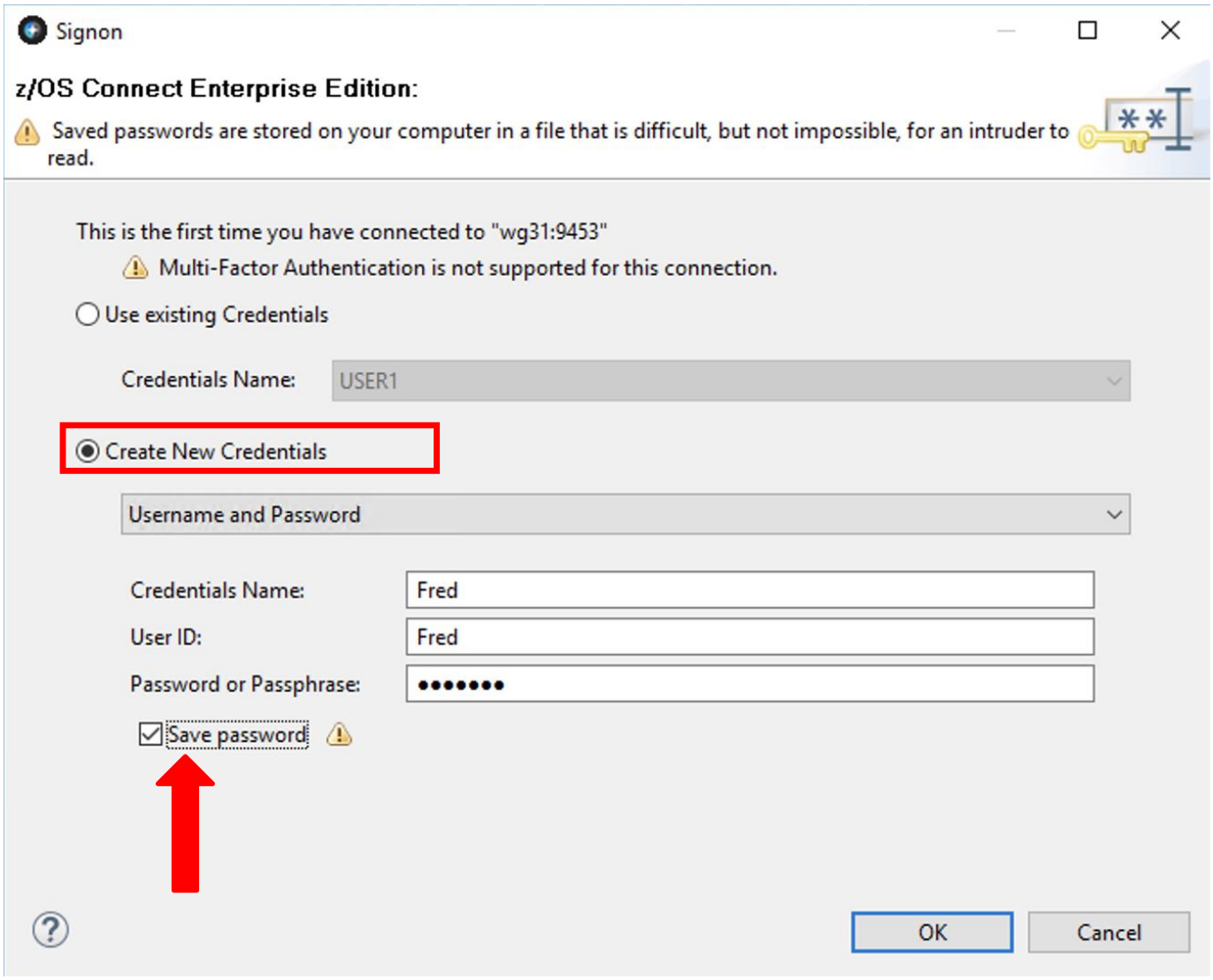
__5. Signing in and saving credentials

A. After clicking **Save and Connect** on the “Add z/OS Connect Enterprise Edition” window, a new “Signon” window pops up. Enter the following information:

__i. User ID: **Fred**

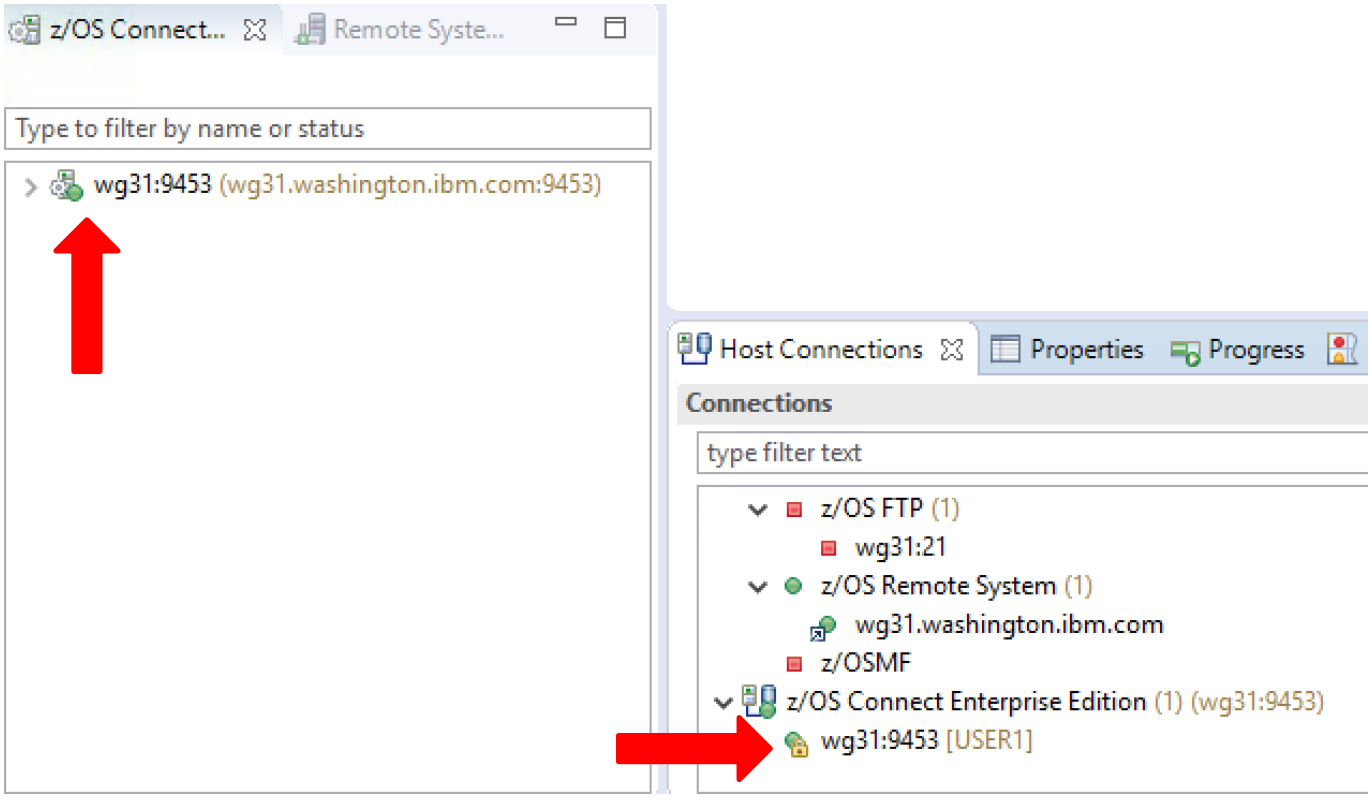
__ii. Password or Passphrase: **fredpwd (case sensitive)**

__iii. Check the “Save password” checkbox to make future connections simple.

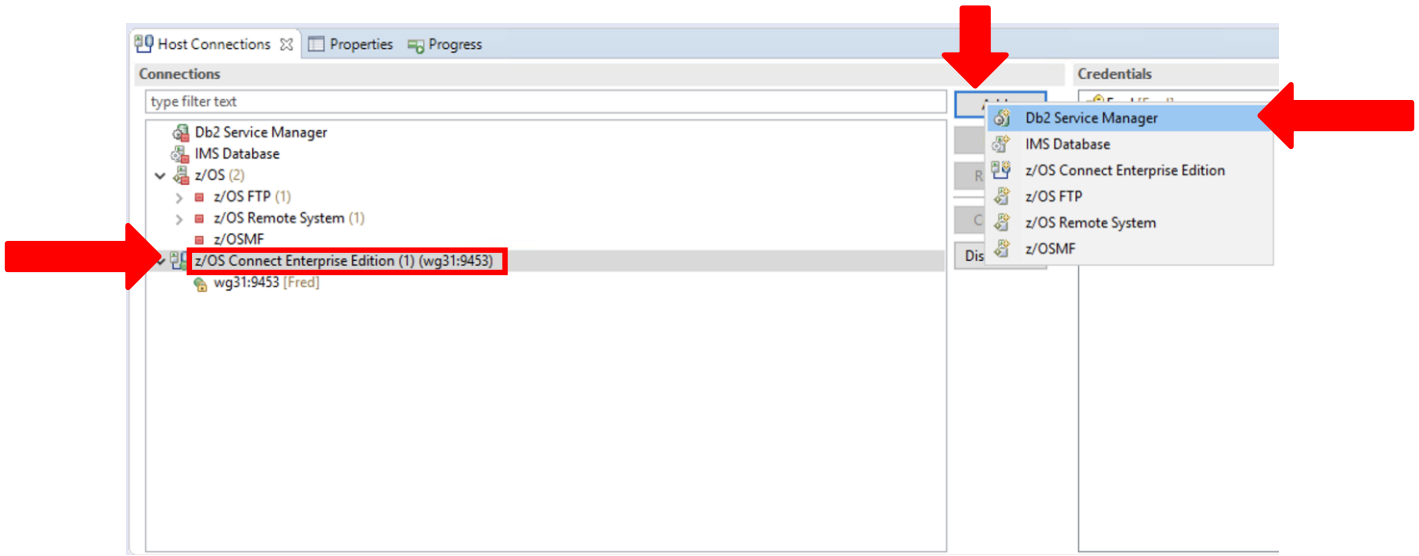


__6. Click **OK.**

__7. Successful connection is indicated in the “Host Connections” and “z/OS Connect EE Servers” tabs by the **red square** icons changing to **green circle** icons. As well as at the bottom of the z/OS Explorer window.



__8. Next, we will establish a connection for the Db2 Service Manager. Add your connection by clicking on “Host connections”, then “z/OS Connect Enterprise Edition”, the **Add** button, and then “Db2 Service Manager”.



__9. Adding a Db2 Service Manager Connection

A. First you will be prompted for the Host Name, which is the same as the LPAR address. In the “Add Db2 Service Manager Connection” Window that popped up, enter the following information:

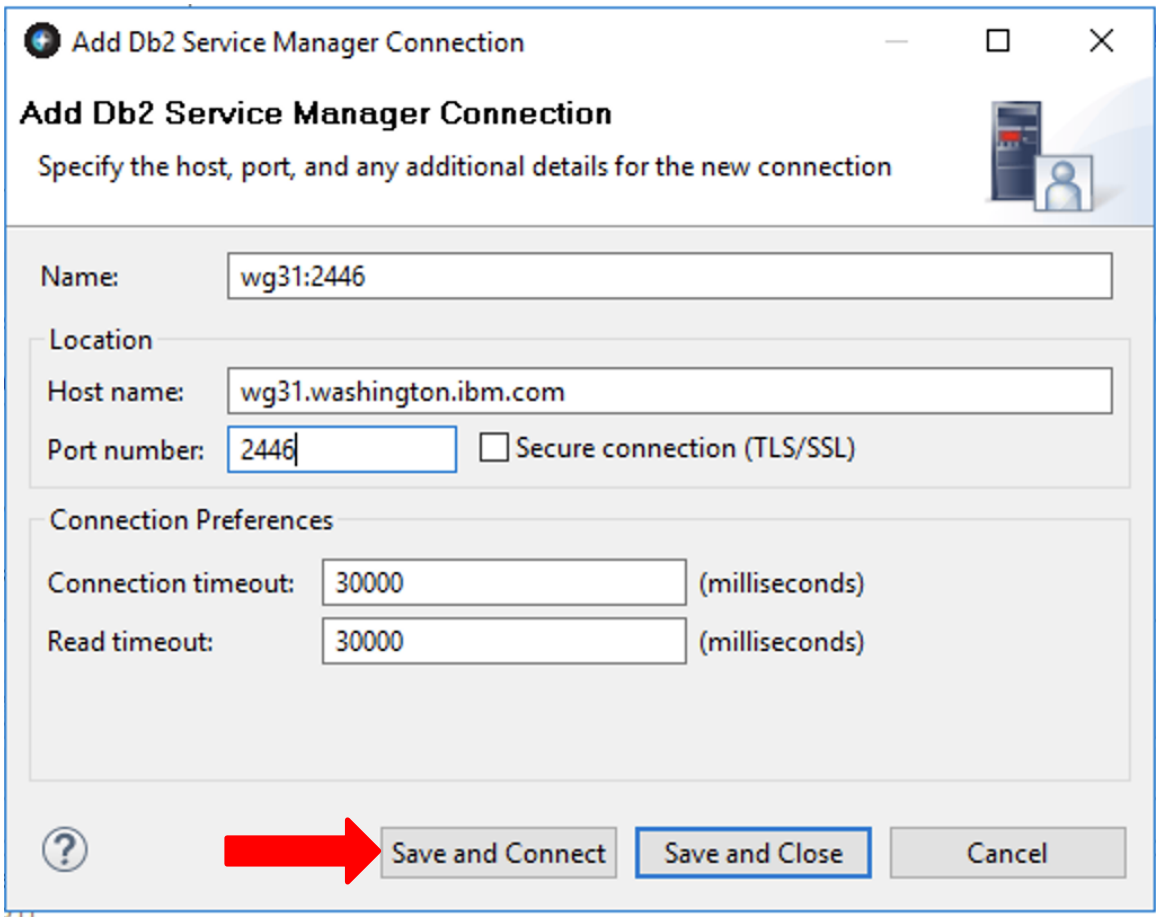
__i. Host name: **wg31.washington.ibm.com**

__ii. Port number: **2446**

__iii. Name: **wg31:2446**

__**iv. DO NOT check the checkbox “Secure connection (TLS/SSL)”**

Unlike the zCEE Connection DO NOT check the secure connection checkbox



__10. Finally, click **Save and Connect**

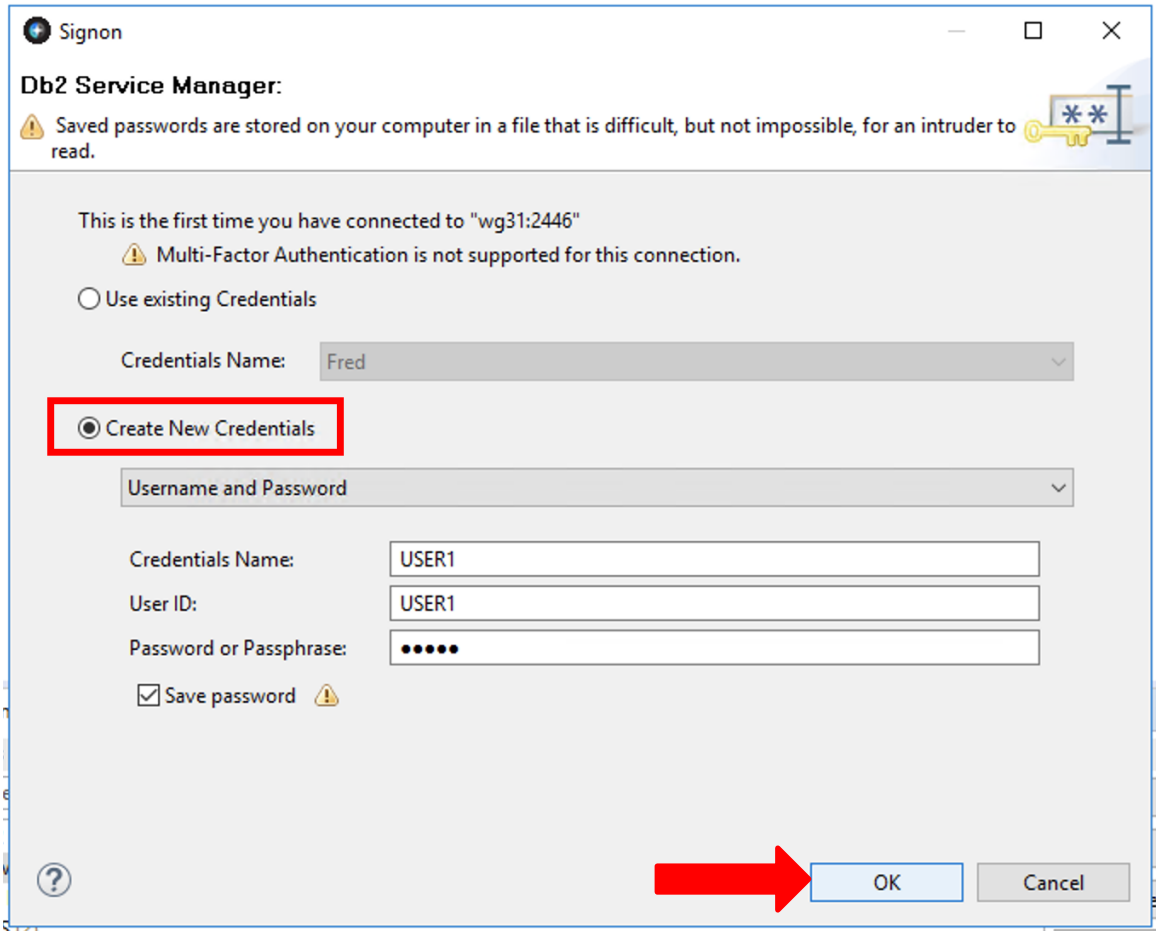
__11. Signing in and saving credentials

A. After clicking **Save and Connect** on the “Add z/OS Connect Enterprise Edition” window, a new “Signon” window pops up. Enter the following information:

__i. User ID: **USER1**

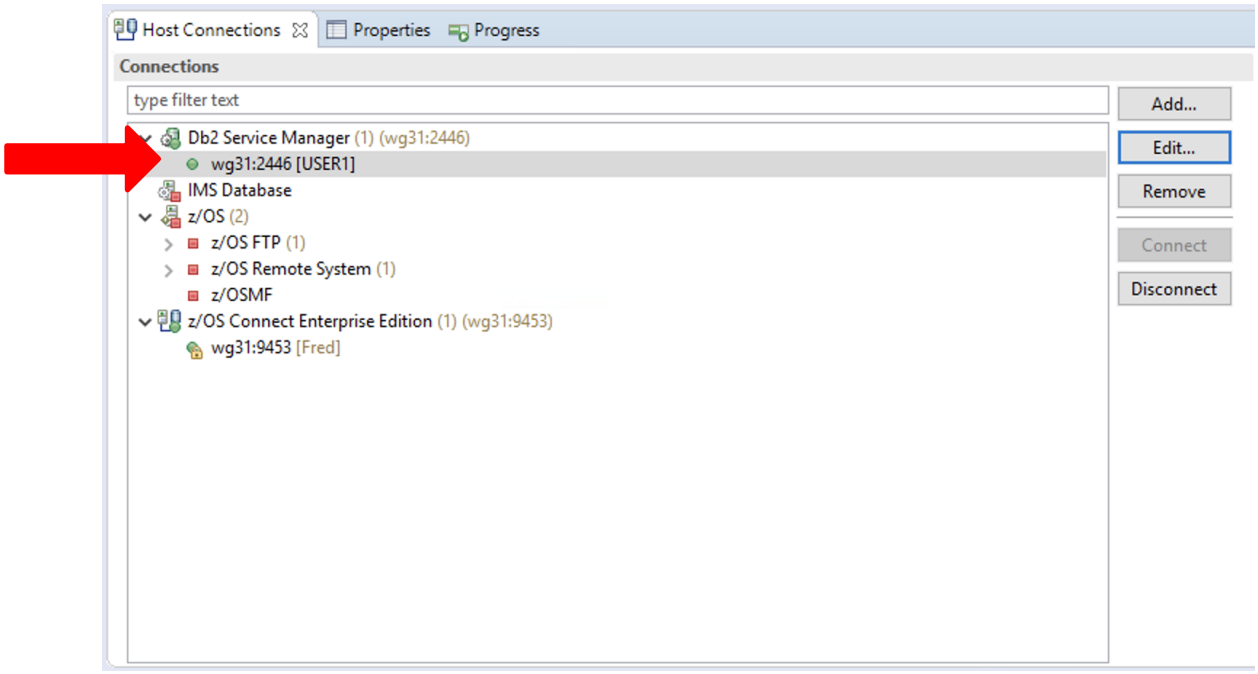
__ii. Password or Passphrase: **USER1**

__iii. Check the “Save password” checkbox to make future connections simple.



__12. Click **OK**.

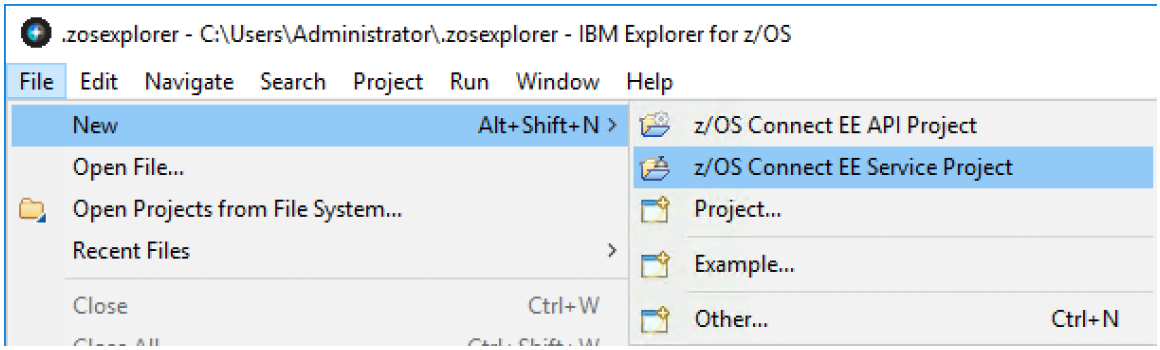
__13. Successful connection is indicated in the “Host Connections” tab by the Db2 Service Manager **red square** icons changing to **green circle** icons. As well as at the bottom of the z/OS Explorer window.



2.1.2 Create the Service Archive File (SAR)

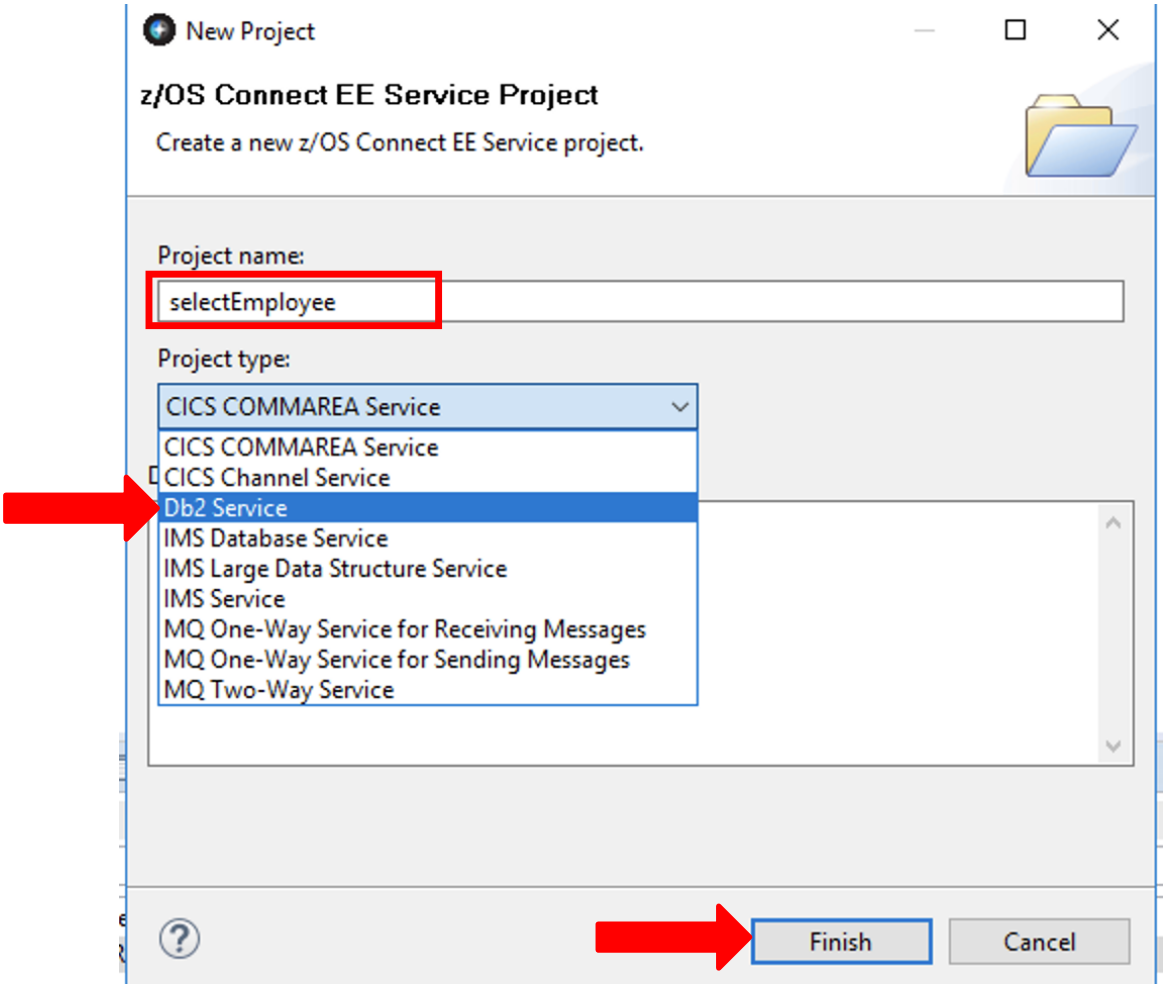
__1. Create a z/OS Connect EE Service Projects.

A. Navigate to z/OS Connect EE Service Project by clicking on File > New > z/OS Connect EE Service Project.



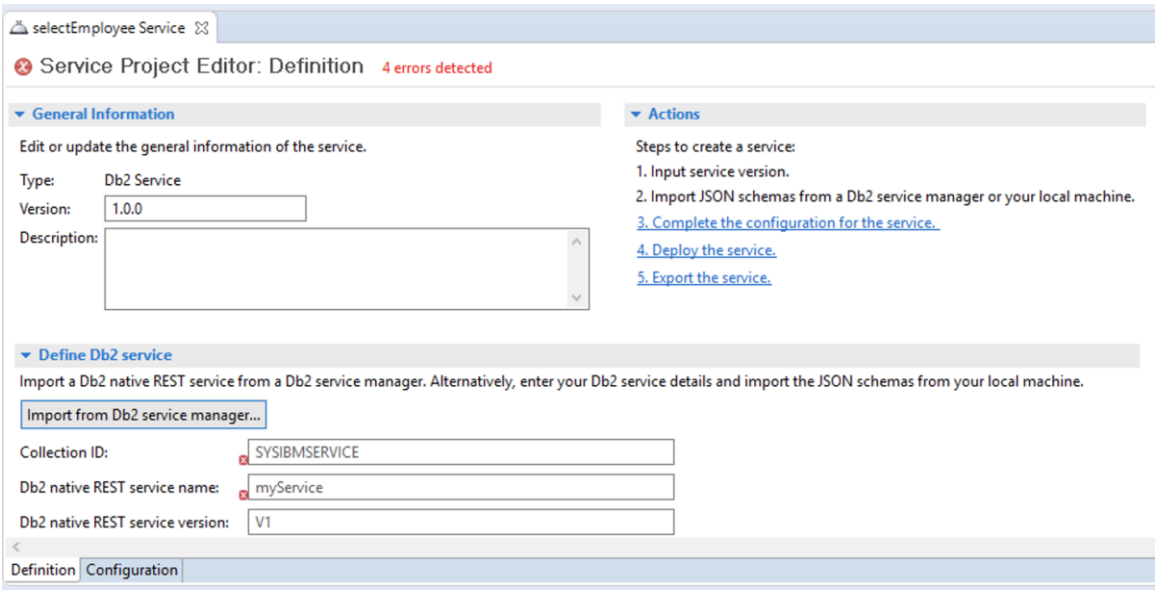
B. Enter "selectEmployee" as the Project Name.

C. Change Project Type in the drop-down menu to Db2 Service.

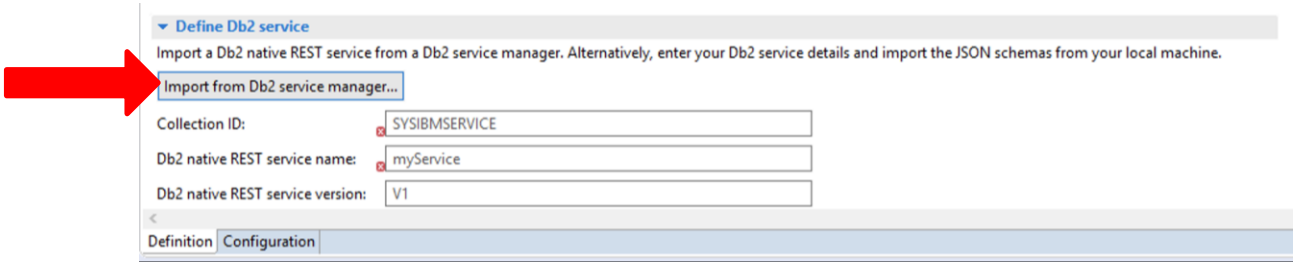


D. Click **Finish**.

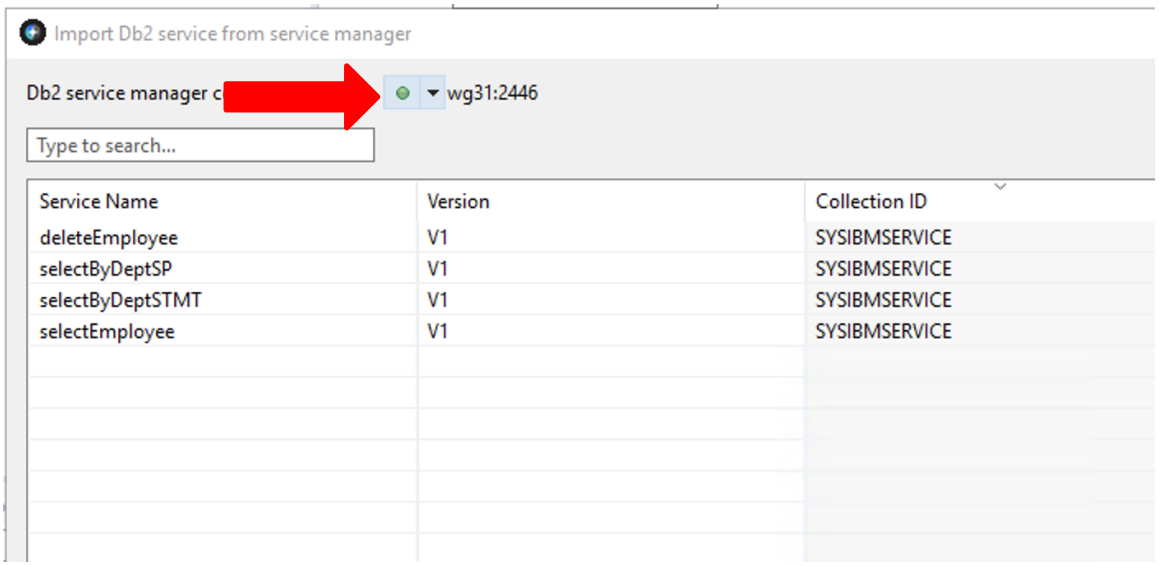
__2. This will open the “Service Project Editor: Definition” window for the selectEmployee service. For now, disregard the message about the 4 errors detected, they will be addressed shortly.



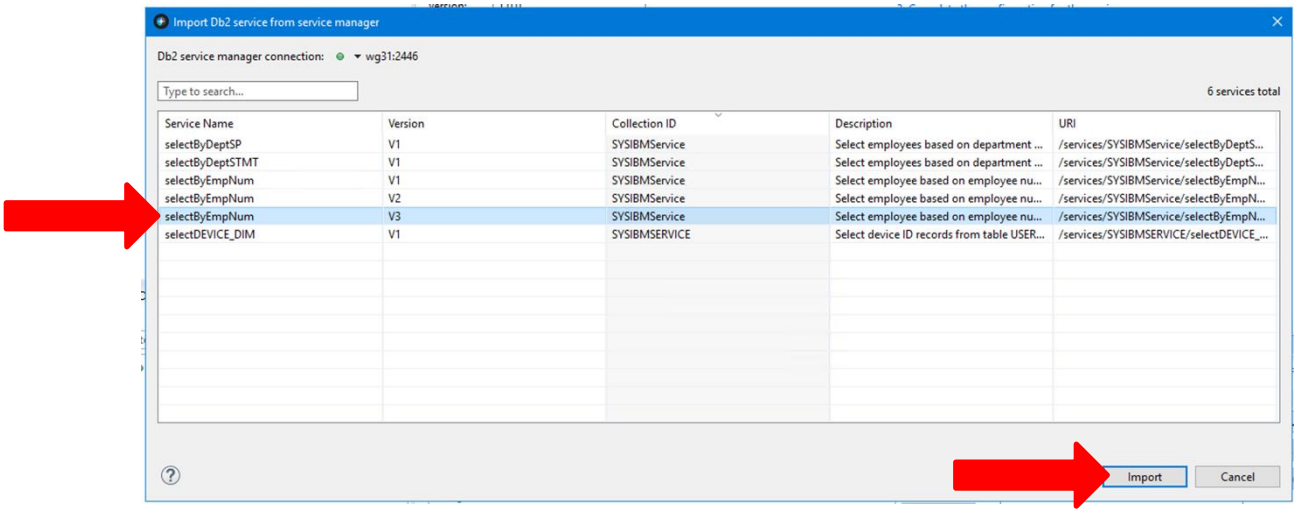
__3. Under the “Define Db2 Service”, section click **Import from Db2 service manager…**



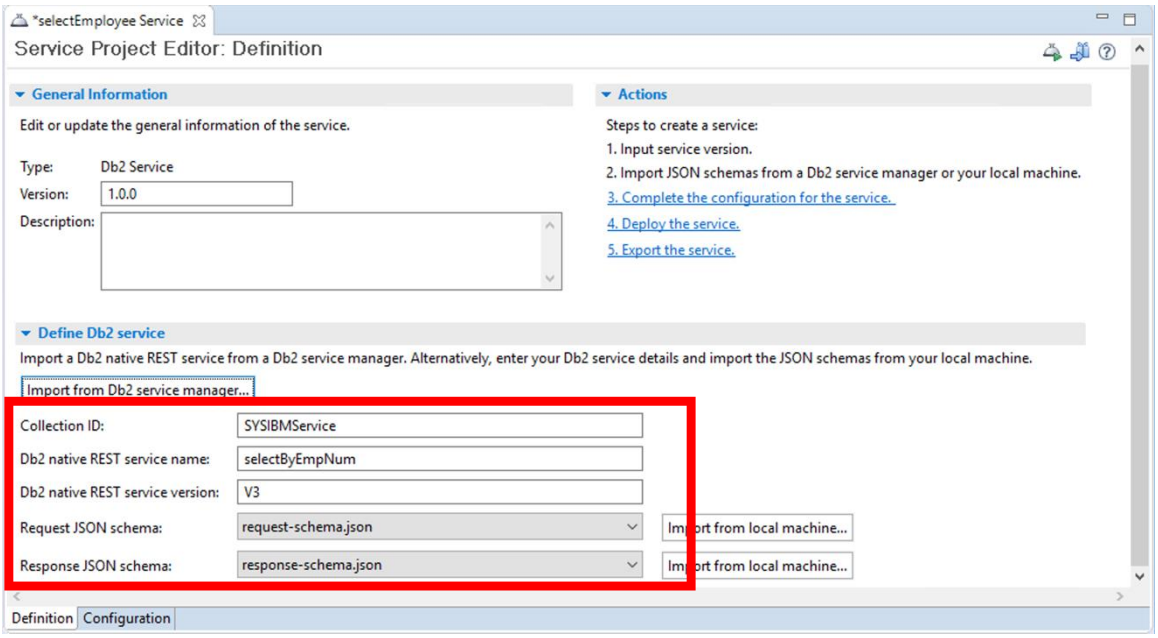
__4. In the “Import Db2 service from service manager” window, confirm the connection to the Db2 subsystem, which is indicated by the **green circle** next to the connection name “wg31:2446” – in this case the beginning of the LPAR and port number.



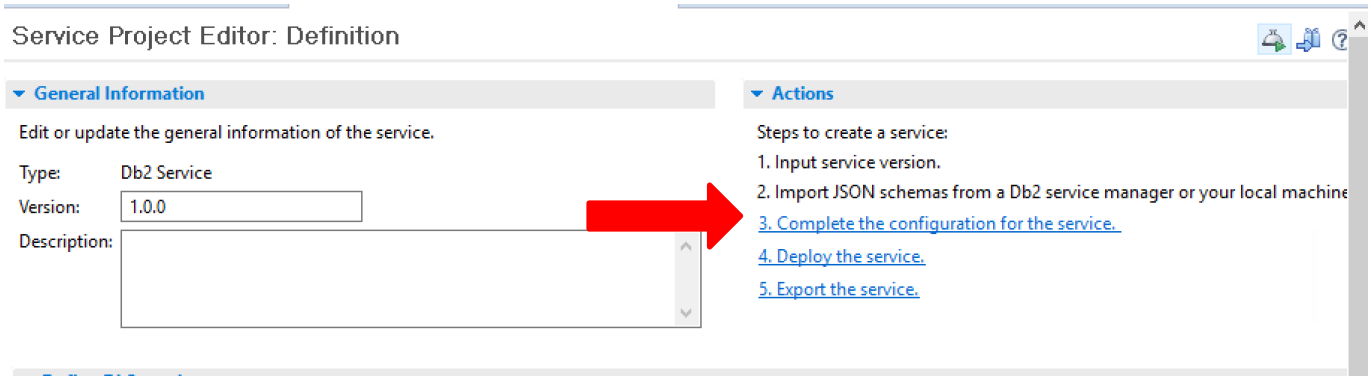
__5. Select the **selectByEmpNum V3** that you created in Lab 1 service under “Service Name” and click **Import.**



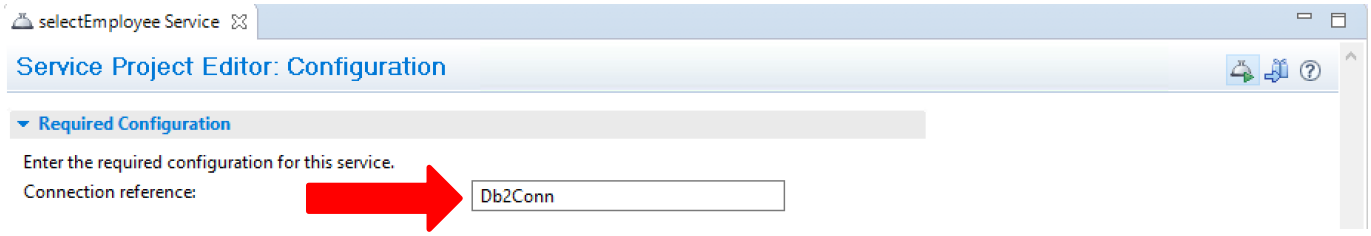
__6. This will return you to the “Service Project Editor: Definition” window. All of the service required information has been determined based on the information imported from the Db2 Service Manager.



__7. Under “Actions”, click the link for **3. Complete the configuration for the service**.

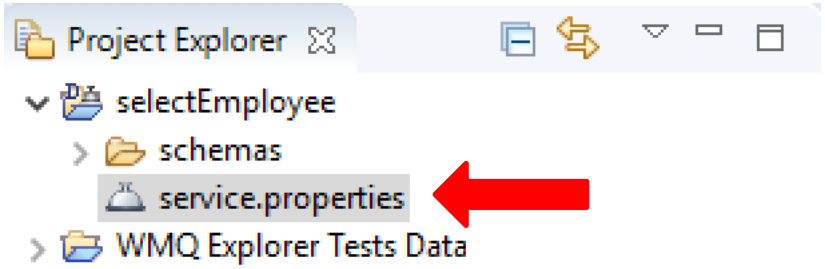


__8. Enter in “Db2Conn” in the Connection Reference box.

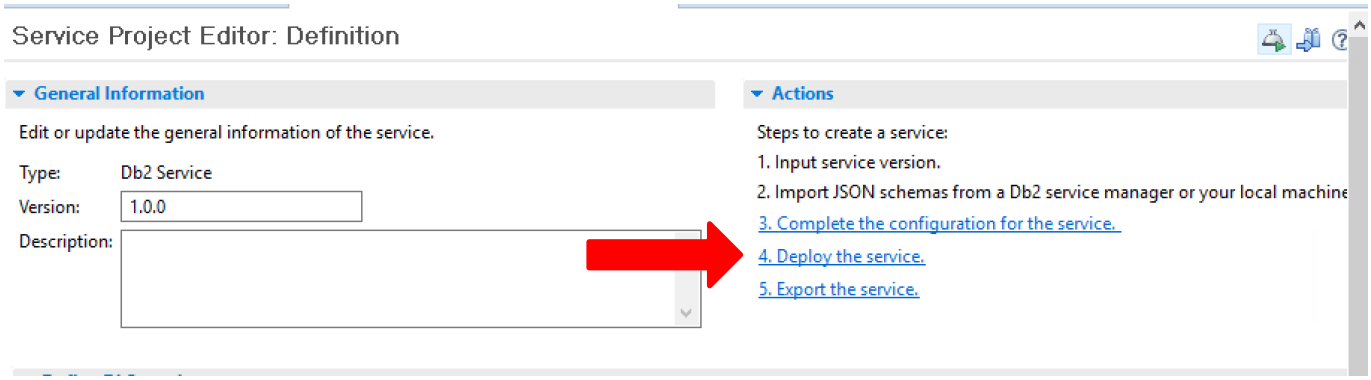


__9. Close the tab titled “selectEmployee Service Project Service” by clicking on the “X”. Make sure to **SAVE** the configuration before closing the window.

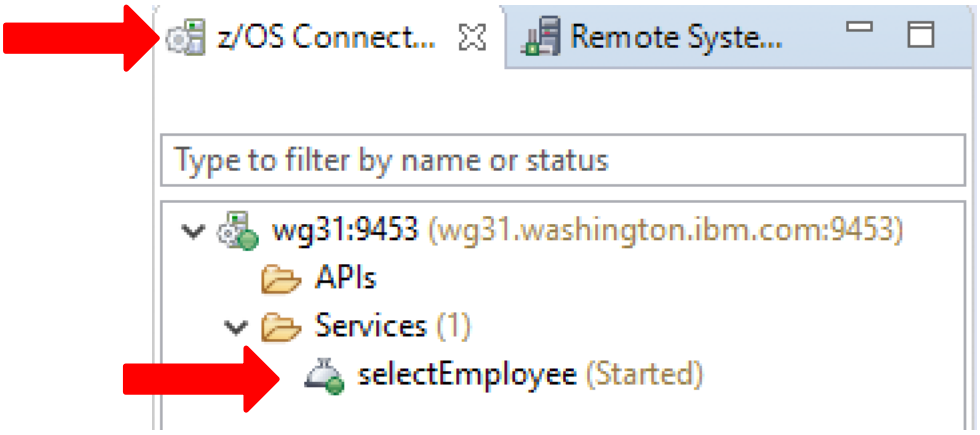
__10. Double-click on the “service.properties” file under the “selectEmployee” project folder in the **Project Explorer** shown below:



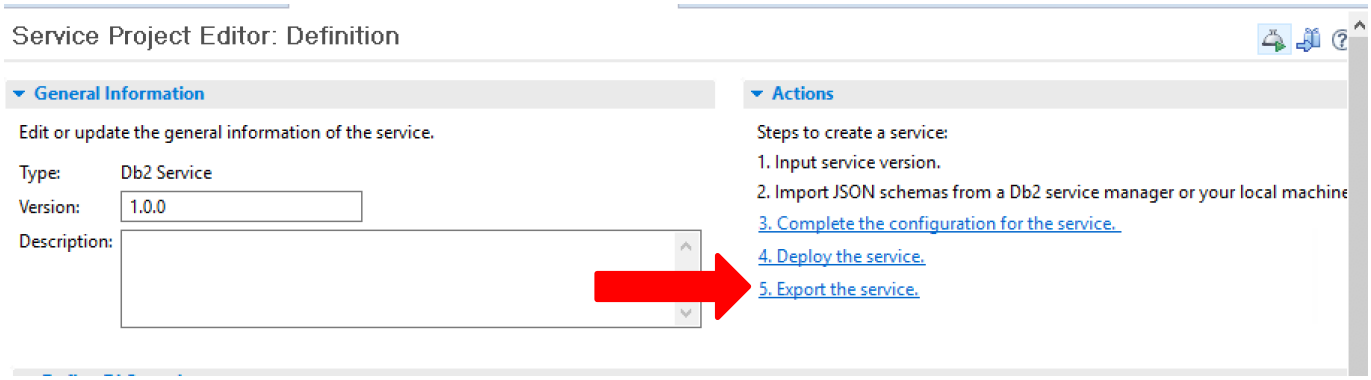
__11. Under “Actions”, click the link for **4. Deploy the service**.



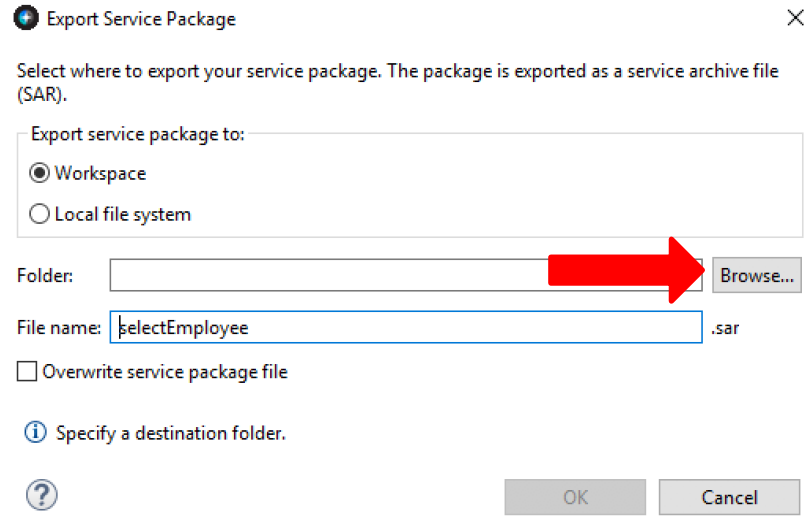
__12. Click “OK” on the pop-up window. Navigate to the “**Services**” folder under the “**wg31:9453**” z/OS Connect server, located in the bottom left-hand corner of the window in the “z/OS Connect EE Servers” tab. You should see a green dot next to your newly created service.



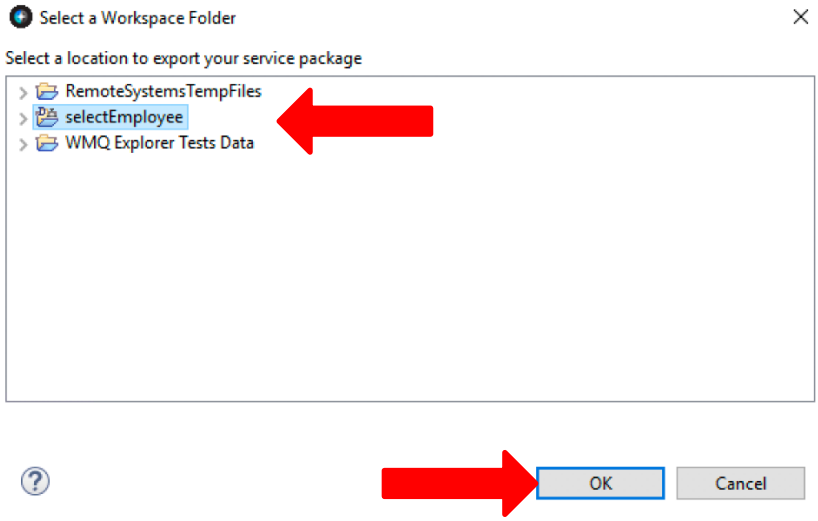
__13. Back in the “Service Project Editor”, under “Actions”, click the link for **5. Export the service.**



__14. Click the "**Browse**" button.



__15. Select the service project folder, the click OK.



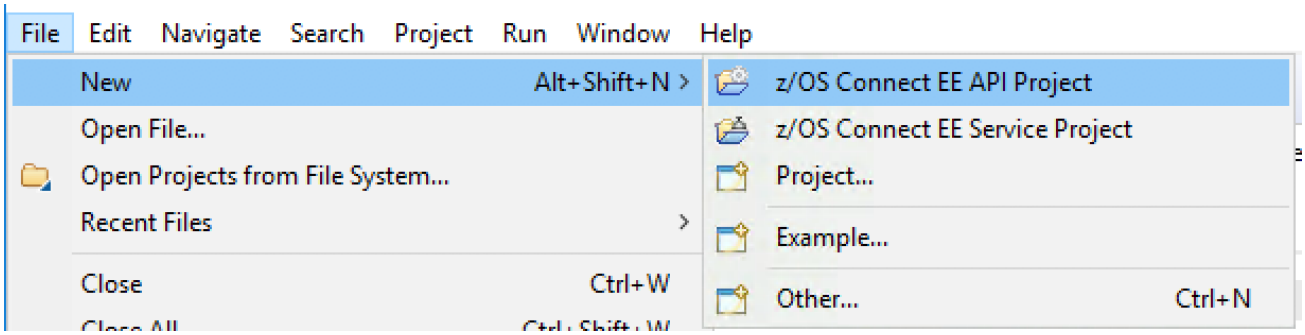
__16. Export the service package by clicking "OK" in the Export Service Package window.

Summary: In this section we created a new zCEE Connection, a new Db2 service manager connection, and created the SAR files by deploying and exporting the selectEmployee service to the server.

2.1.3 Create a Db2 REST API project

__1. Create a REST API Project

A. Using the menu bar, select “File” and navigate to “New” > “z/OS Connect EE API Project”.



B. In the “New Project” window enter the following information.

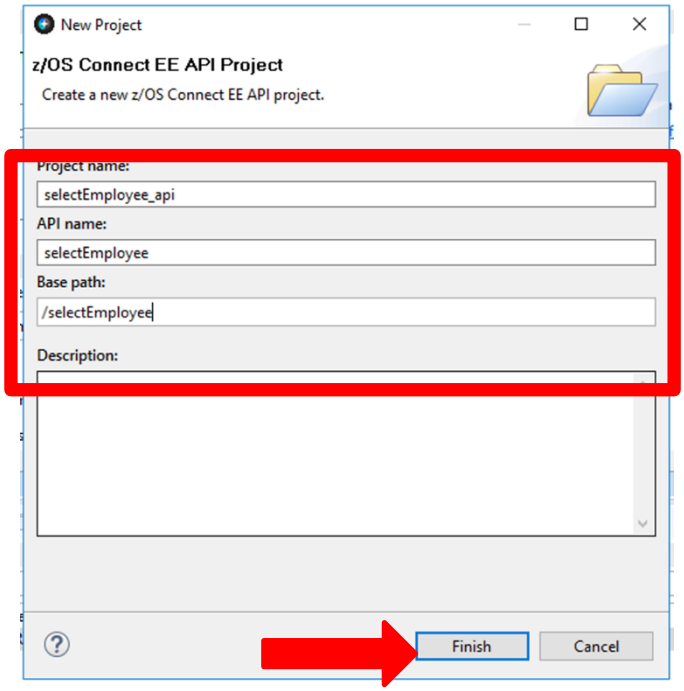
BEWARE THAT THIS INFORMATION IS CASE SENSITIVE. IF YOU TYPE THE NAME MANUALLY BE SURE TO USE camelCase BECAUSE Z/OS CONNECT AUTOFILLS THE FIELDS WITHOUT CAMELCASE.

__i. Project name: **selectEmployee_api**

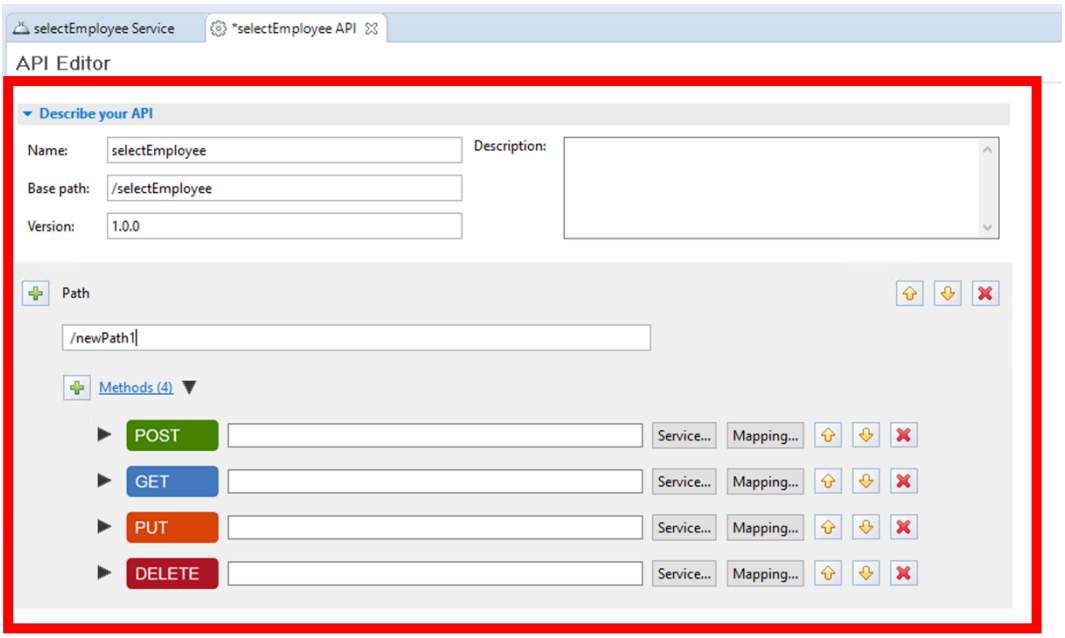
__ii. API name: **selectEmployee**

__iii. Base path: **/selectEmployee**

C. Click **Finish.**



__2. Listed below is the new selectEmployee project, and the “z/OS Connect EE API Editor” view was launched.

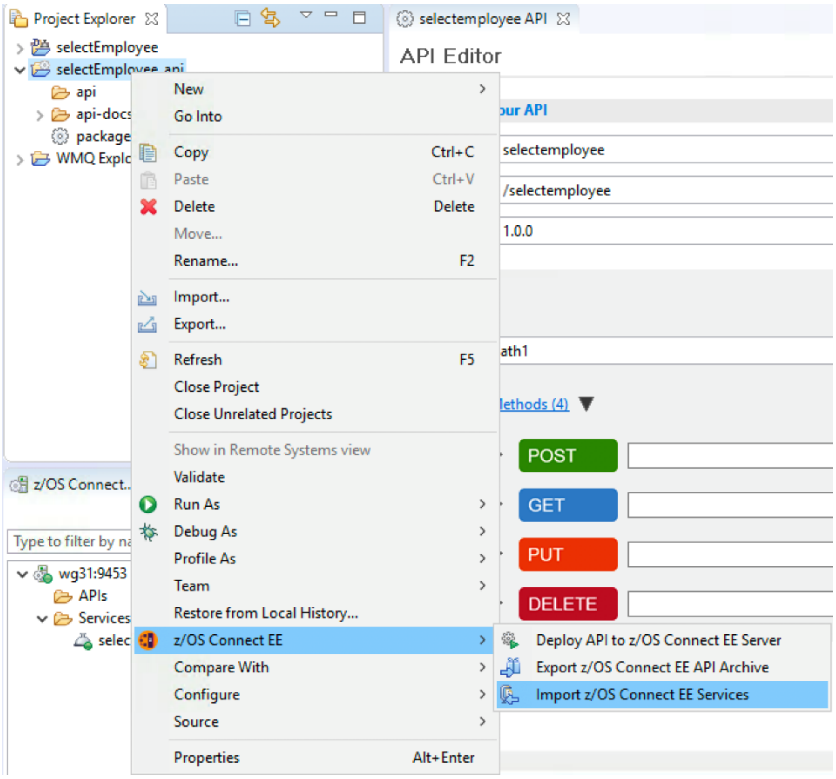


Summary: In this section we created a new REST API Project in z/OS Explorer.

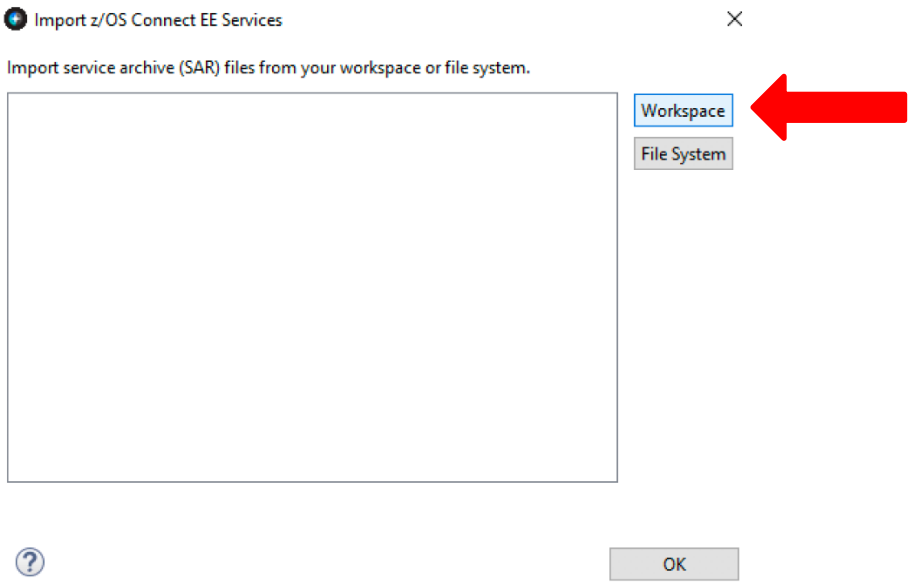
2.1.4 Import the Db2 REST service's SAR file

__1. Import the Db2 REST SAR file "selectEmployee.sar" SAR into the project.

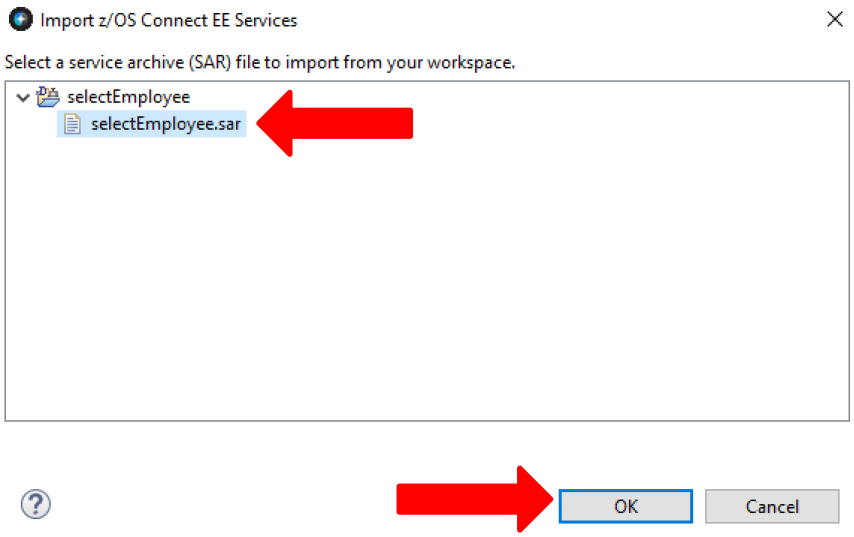
A. Right click on the newly created project name in the Project Explorer on the top left, then click on z/OS Connect EE, then click on z/OS Connect EE Services.



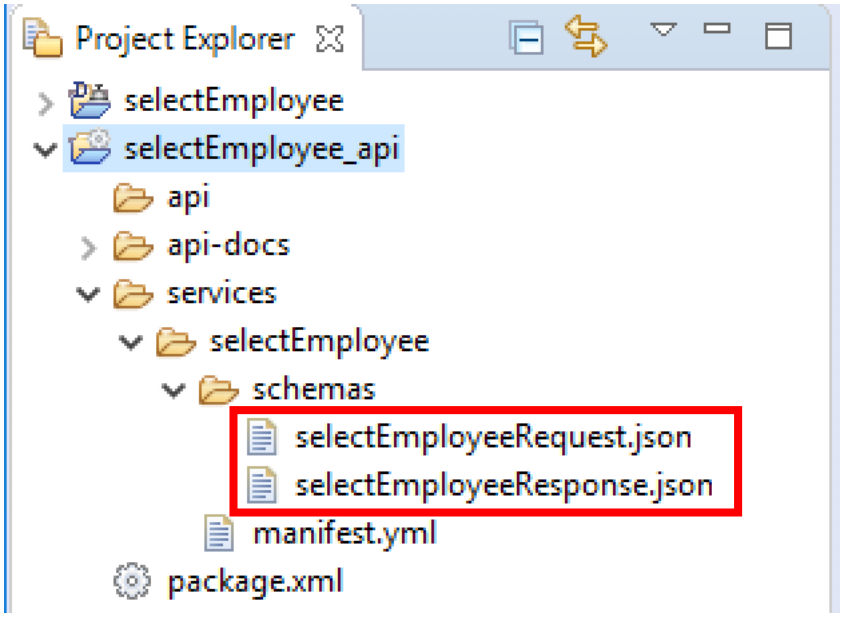
B. Click **Workspace** in the "Import z/OS Connect EE Services Window".



C. Select the "**selectEmployee**" file. Click OK.



__2. To confirm importing of the SAR completed successfully, open all of the selectEmployee_api project's toggles, in the Project Explorer. You should see the same information below and the request and response schema files. The request and response file names were changed slightly from your original names.



Summary: In this section we imported the Db2 REST service's SAR file.

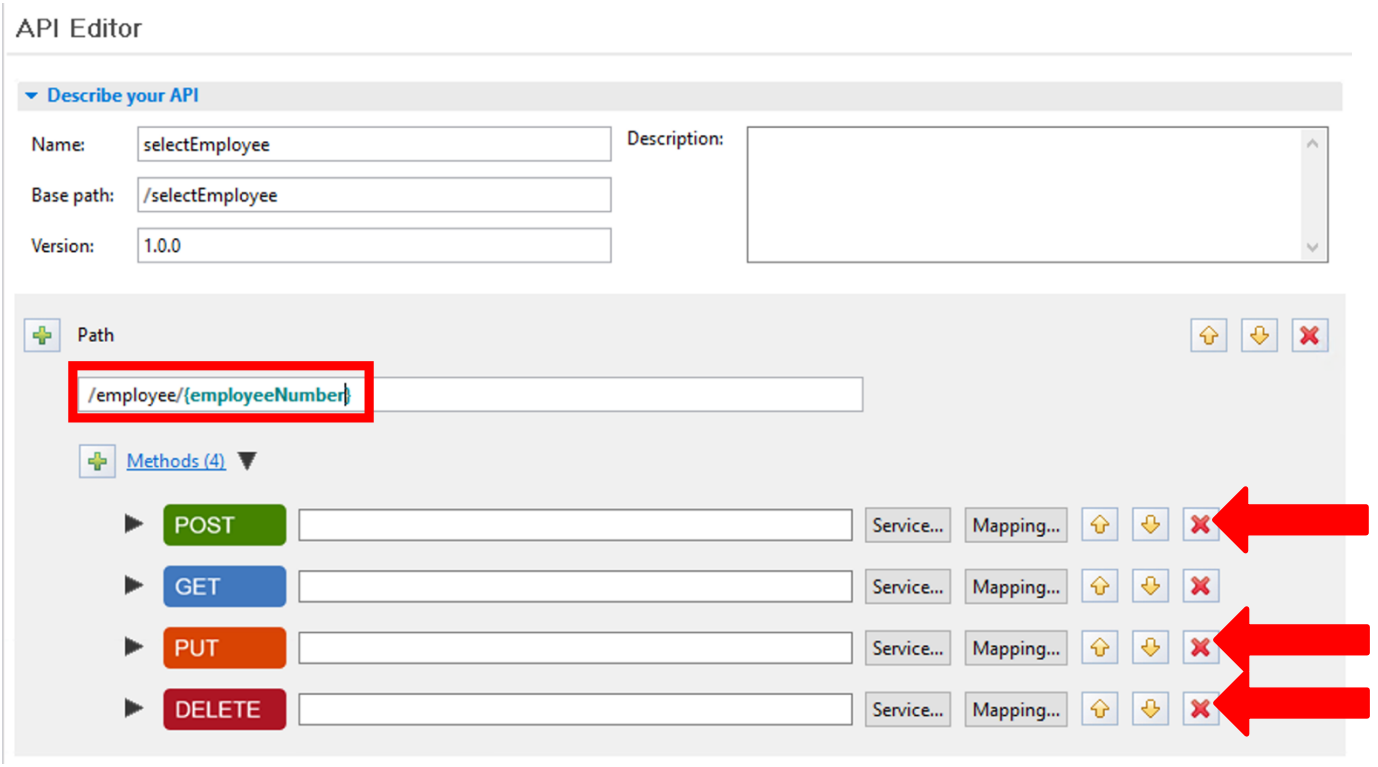
2.1.5 Map a POST method to a GET method

__1. Update the API Editor properties and path as follows:

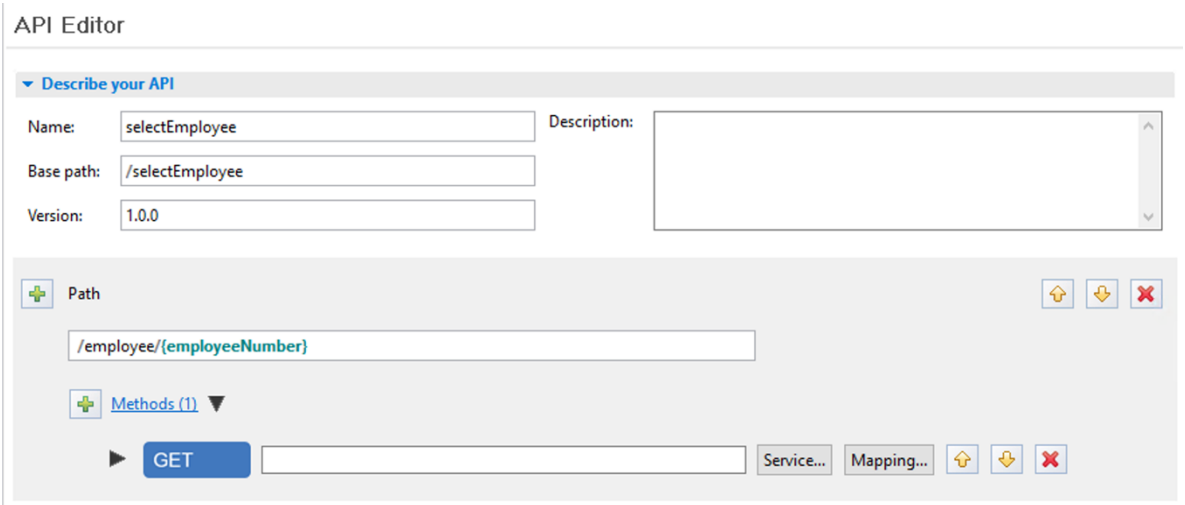
A. Path: **/employee/{employeeNumber}**

__i. {employeeNumber} is for the input variable, which is for the employee's identification number

B. Click the red X buttons to remove the POST, PUT, and DELETE methods.

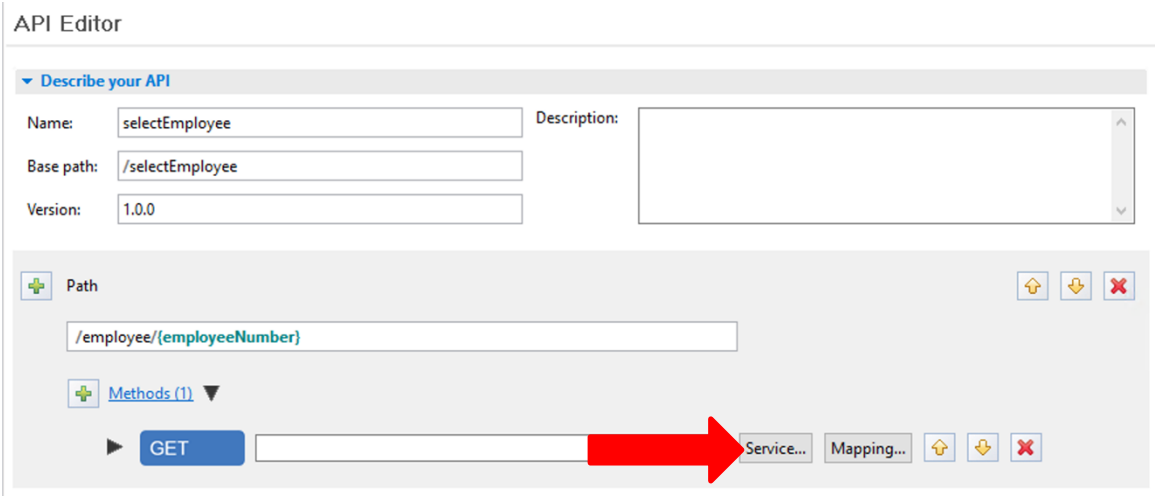


C. The updated API Editor view should look like below after the changes are made:



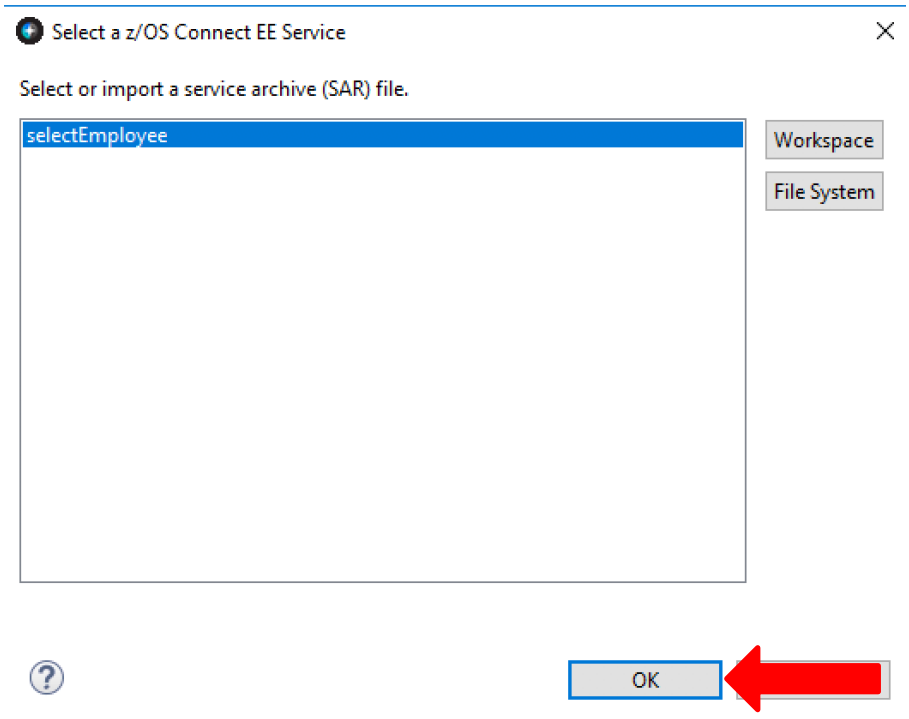
__2. Apply the Db2 service to the method by importing the updated selectEmployee SAR file. Note – the service name must be the same as the native REST service name.

A. Click the **Service…** button for the **GET** method.

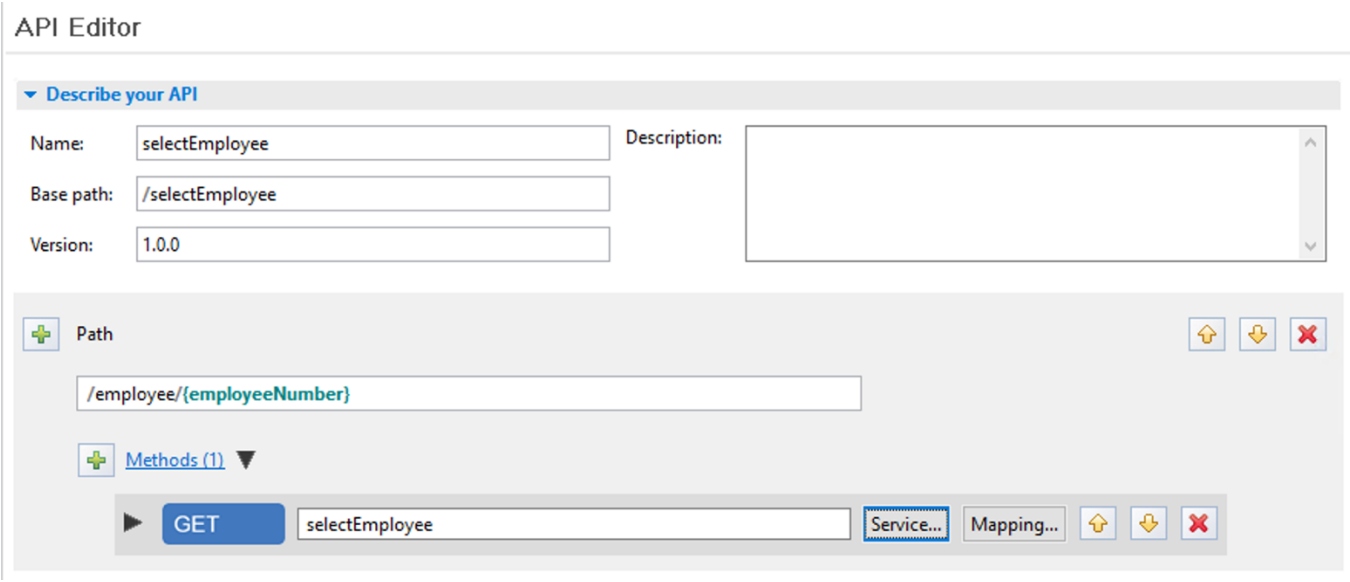


B. Select the selectEmployee SAR file that you imported in previously.

C. Click the **OK** button.



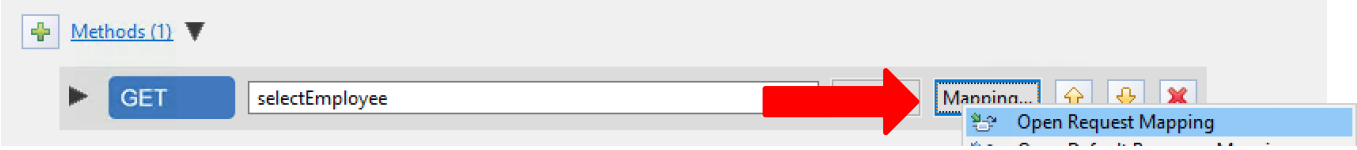
__3. Shown below is the updated GET Method in the API Editor.



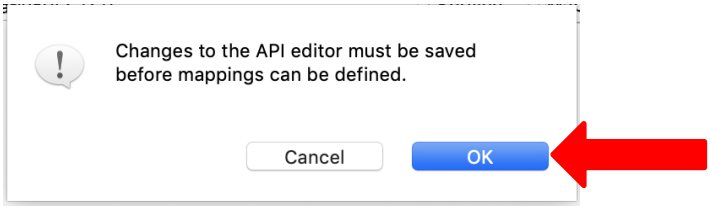
Summary: In this section, we mapped the selectEmployee POST request to the GET method by linking the SAR file to the GET Method in the API editor.

2.1.6 Use the Request Mapping Editor to define the JSON request and response schema fields

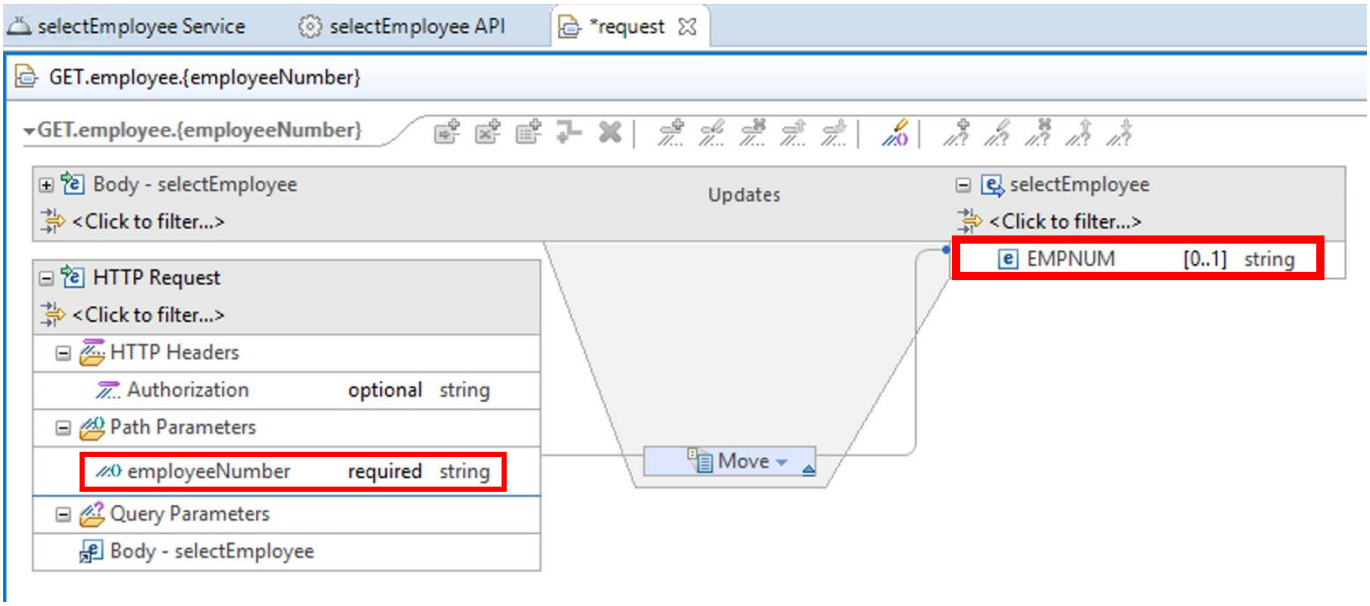
__1. Launch the Request Mapping Editor by clicking the **Mapping…** button for the GET Method and then selecting "**Open Request Mapping**".



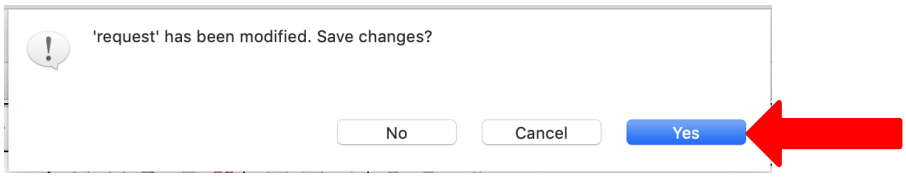
A. Make sure to save before moving forward. Click **OK** to continue.



B. Map the input parameter employeeNumber to EMPNUM. To do this, click the "employeeNumber" in the "**HTTP Request**" box on the **LEFT** **under Path Parameters** and drag it to "EMPNUM" in the "**selectEmployee**" box on the **RIGHT**.



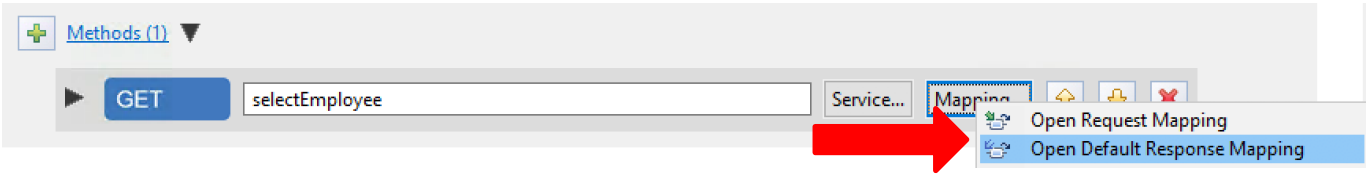
__2. Close the Mapping Editor and select **Yes** in the pop up window to save the changes made. Or issue the command **CNTRL-S**.



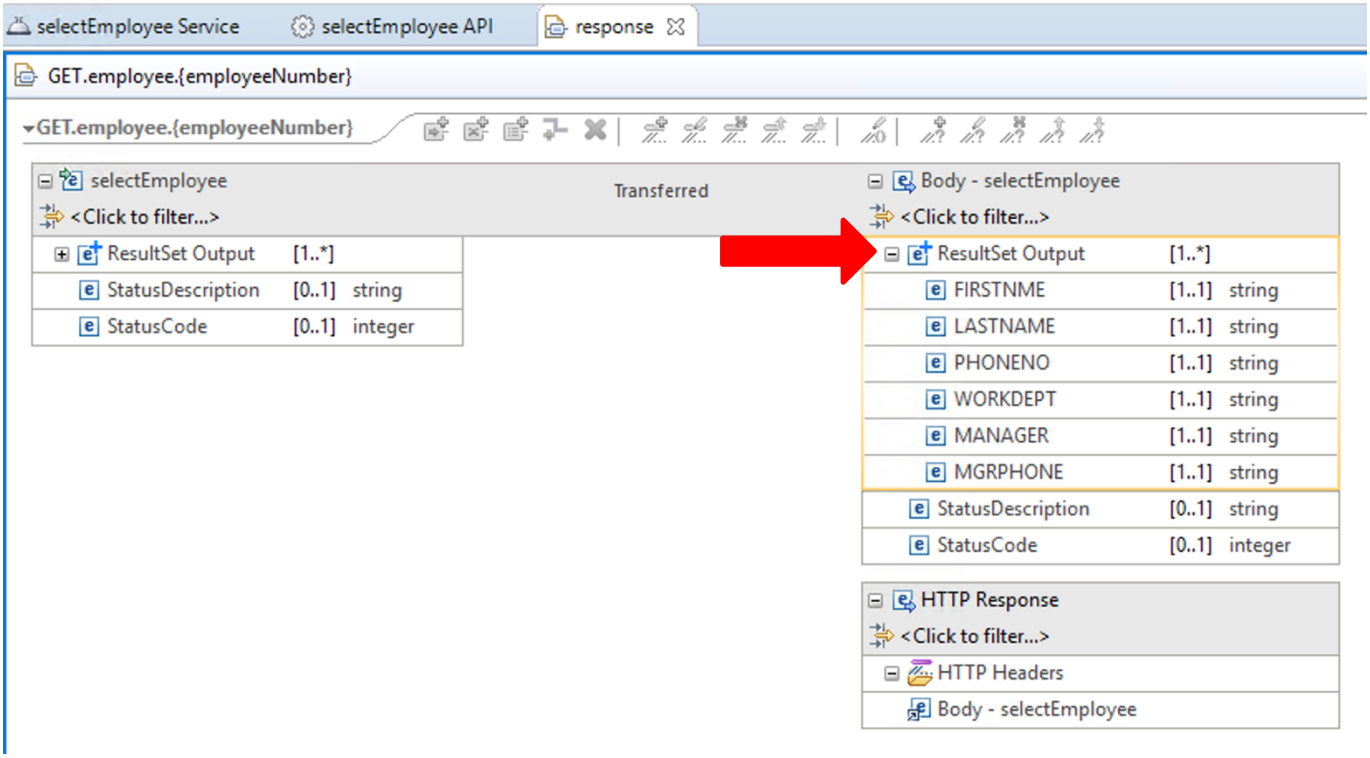
Important!

If you do not issue a save before trying to deploy the API, the deploy WILL NOT WORK

__3. Launch the Request Mapping Editor by clicking the **Mapping…** button for the GET Method and then selecting "**Open Default Response Mapping**".

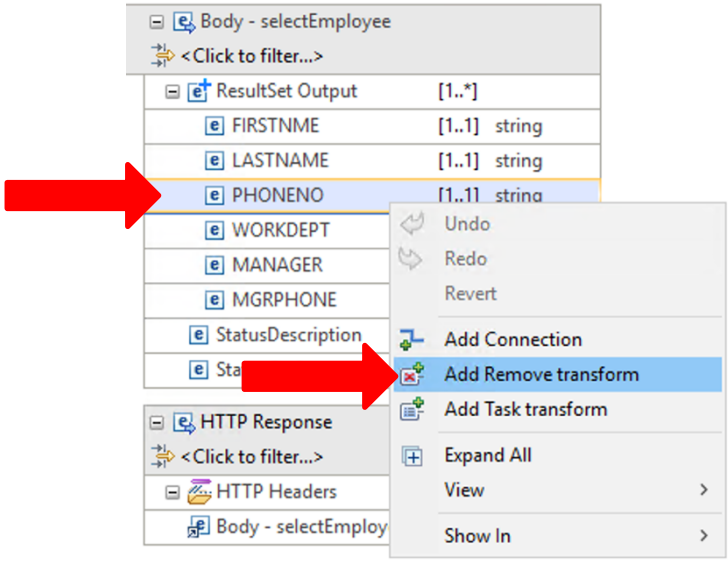


A. If necessary, click the plus icon next to "ResultSet Output" under the **"selectEmployee"** body box on the **RIGHT**.

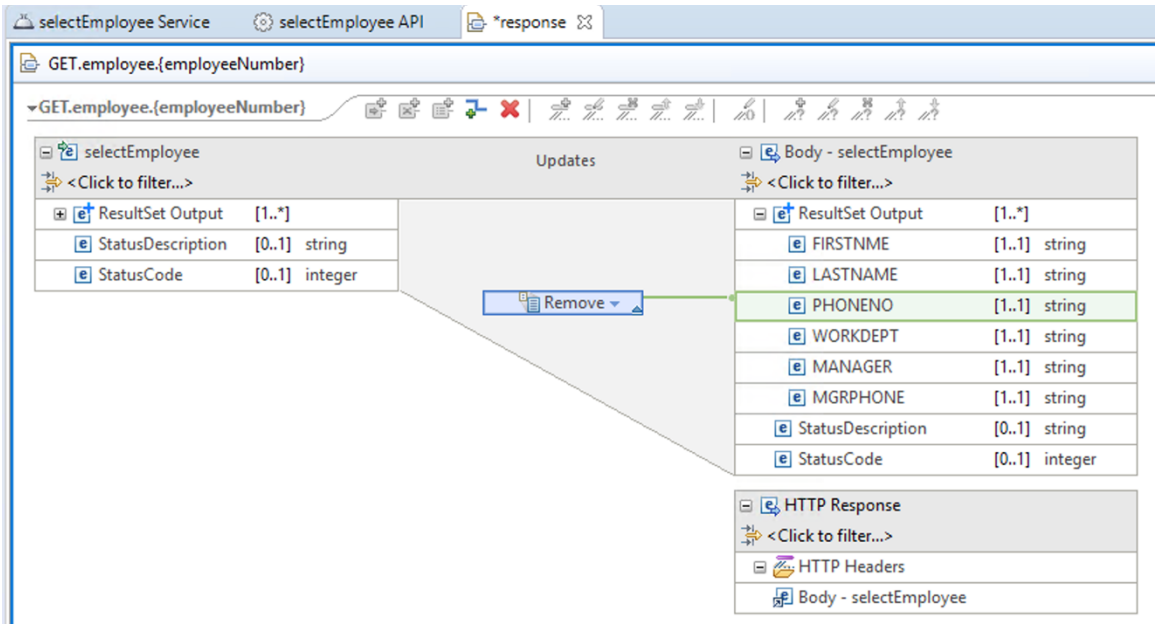


B. Under “selectEmployee” on the right-hand side, select the field with “**PHONENO**”.

C. Right click on the “phoneNumber” field and click “**Add Remove transform**”.



__4. Save the updates, close the Request Mapping Editor and selectEmployee’s API Editor.



Summary: In this section, we used the Request Mapping Editor and the Response Mapping Editor to define the JSON request and response schema fields. We removed all fields with sensitive information from the API response.

2.1.7 Deploy the RESTful API to z/OS Connect EE

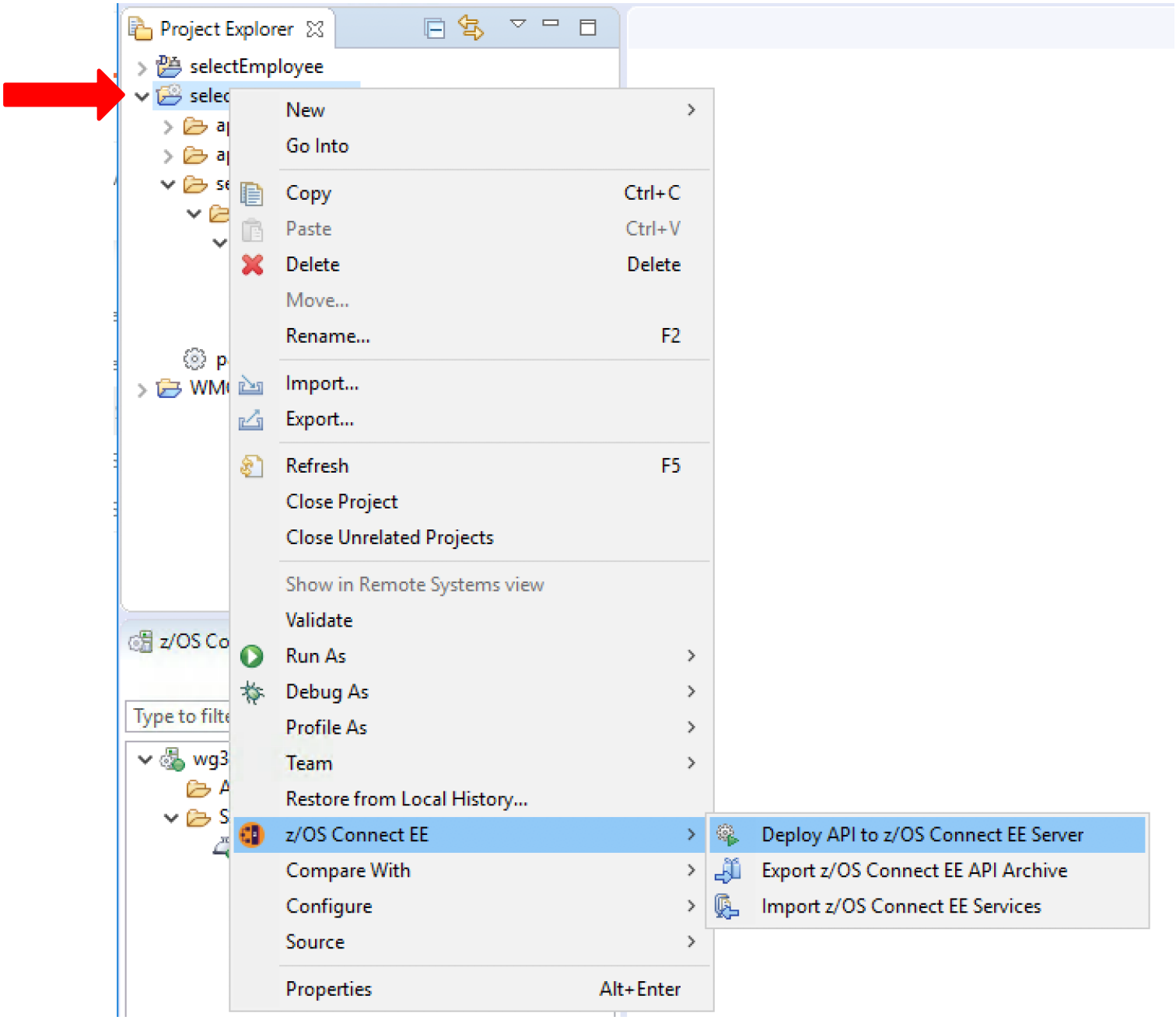
The next step will deploy the selectEmployee API to the z/OS Connect EE server ZCEESRV1 and the Swagger document will also be deployed for the API.

__1. Launch the deployment tool

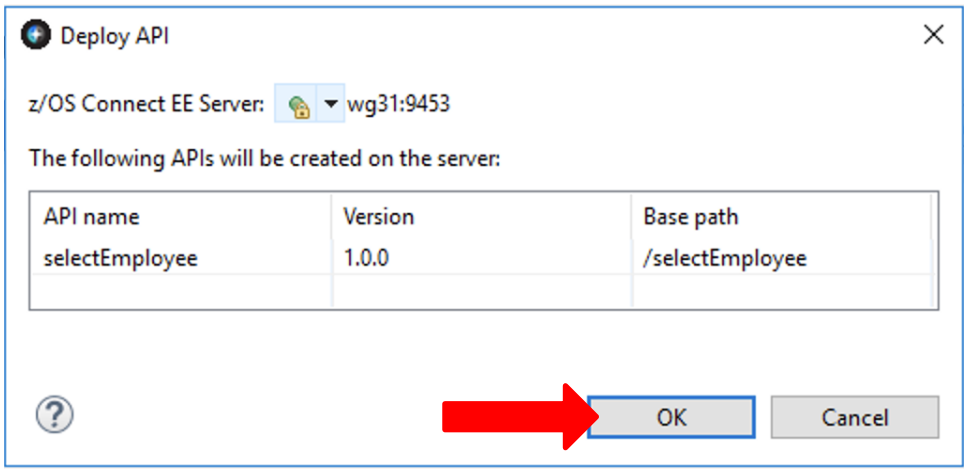
A. Right click on the selectEmployee_api project in the "**Project Explorer"**

B. Click on "z/OS Connect EE"

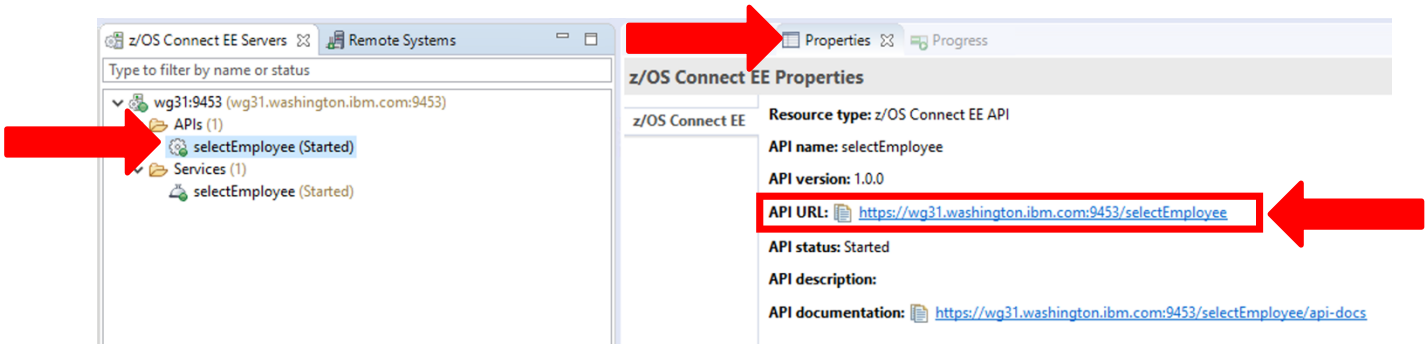
C. Click on "Deploy API to the z/OS Connect EE Server"



__2. Confirm deploying the API by clicking the **OK** button on the “Deploy API” Window.



__3. After successful deployment of the selectEmployee Db2 REST API, the properties view tab shows detailed information about the API.
NOTE the API URL camelCase. **When you invoke the API, it must match the case, otherwise you will get an error.**



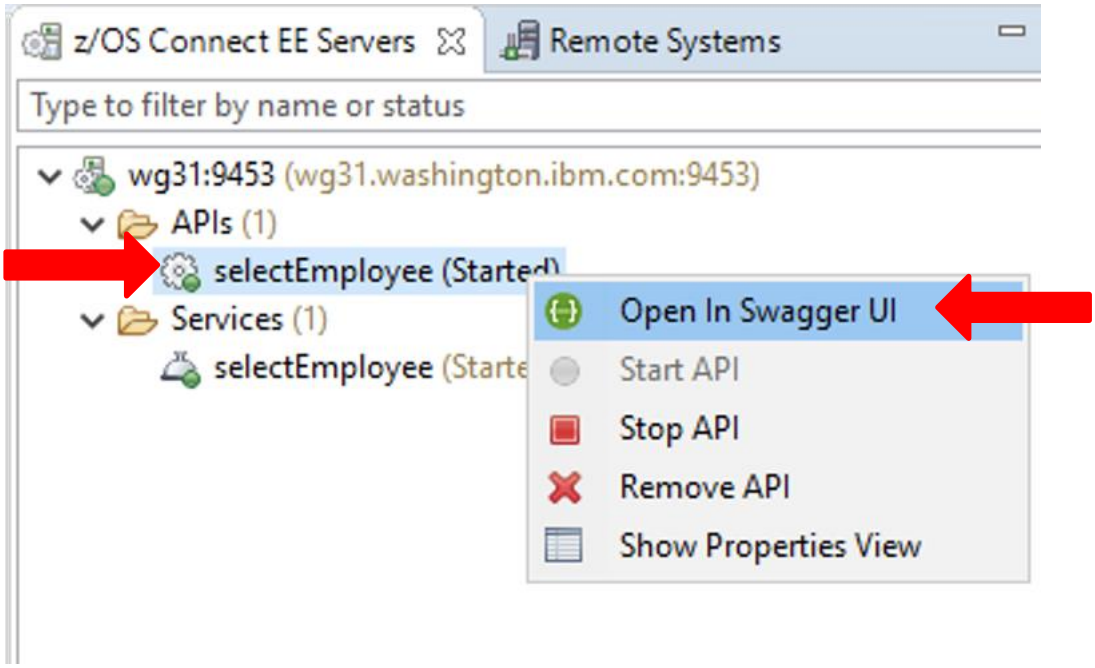
Summary: In this section, we deployed a REST API to the zCEE Server.

2.1.8 Test the selectEmployee Db2 REST API (WARNING CASE MATTERS!!!)

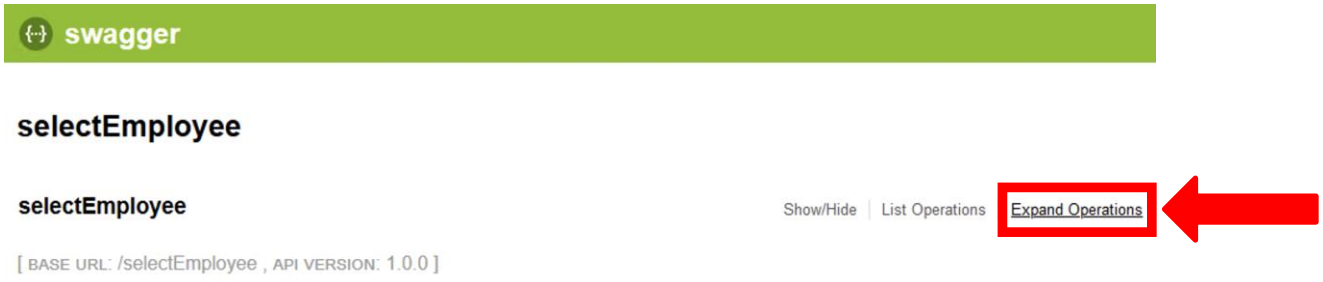
There are a few ways to accomplish this. In this lab, we will go over two ways. One way is to test the API using Swagger in a browser. The other way is to test the API in Postman.

__1. Testing Using **Swagger**

A. In the **z/OS Connect EE Servers** window in the bottom left hand corner, right click the "selectEmployee" under the "APIs" folder and select "Open In Swagger UI".



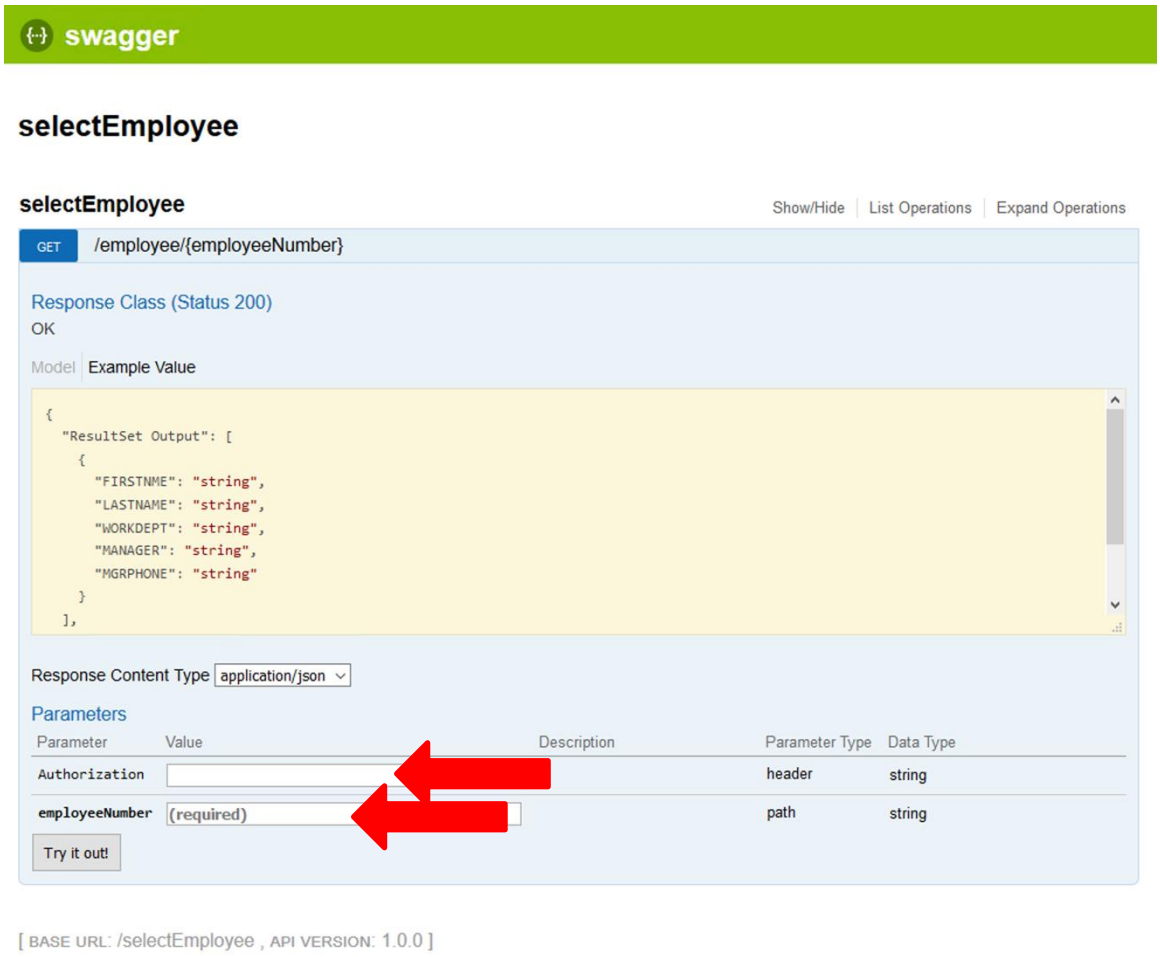
B. The Swagger UI should open up in a Firefox browser window. Click on the "Expand Operations" link on the far right.



C. In the parameters section, enter the following values as parameters:

__i. Authorization: **Basic ZnJlZDpmcmVkcHdk**

__ii. employeeNumber: **000020**



D. Click the "Try it out!" button.

Important Information

You may be challenged by Firefox because the digital certificate used by the Liberty z/OS server is self-signed. After clicking "Try it out!" if you do not get a response from the server do the following.

In a new Firefox tab, enter the url: **https://wg31.washington.ibm.com:9453/zosConnect/apis** and you should be prompted by a security warning. Click the **Advanced** button to continue. Scroll down and then click on the **Accept the Risk and Continue** button.

You may then be prompted for a userid and password. In this case, enter username **Fred** and password **fredpw** and click **OK**.

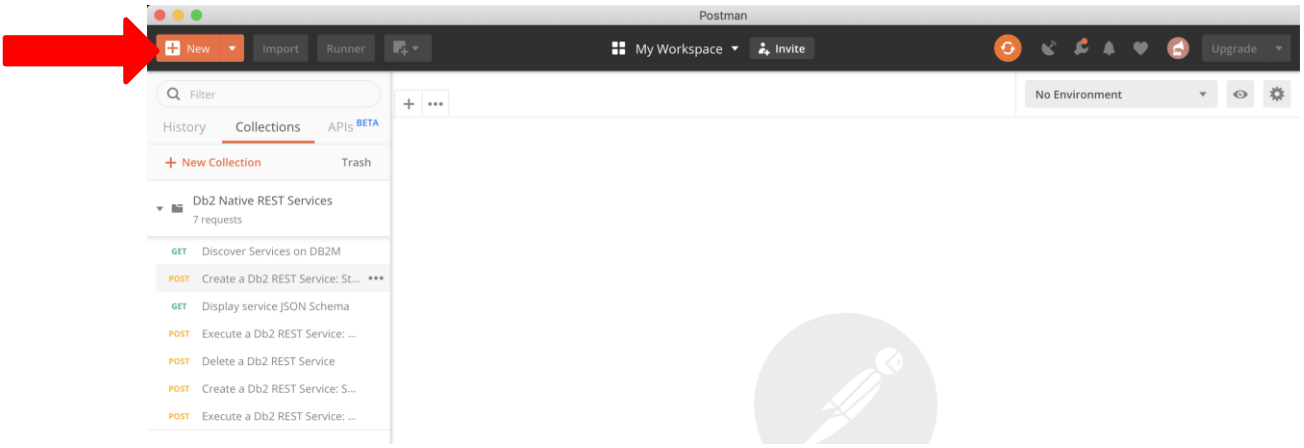
__i. A successful invocation will return a response code "200".

__ii. The response body contains the First Name, Last Name, Work Department, the Manager and the Manager's phone number code for the employee with the "**000020**" employee number. **As you can see, there is no employee Phone Number returned, as the Phone Number field was deleted in the Response mapping when creating the API.**

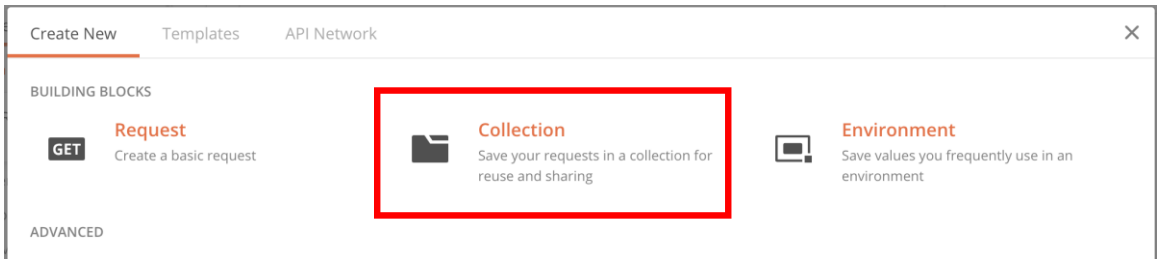
__2. Testing using **Postman**.

A. Create a new collection for testing the APIs.

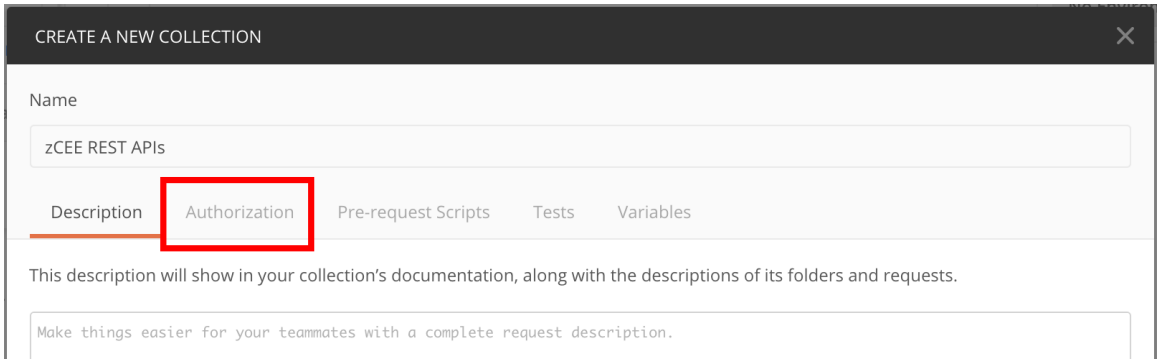
__i. Click the **New** button at the top left corner of the Postman client.



__ii. Select **Collection** to create a new folder for the new requests we are going to create.



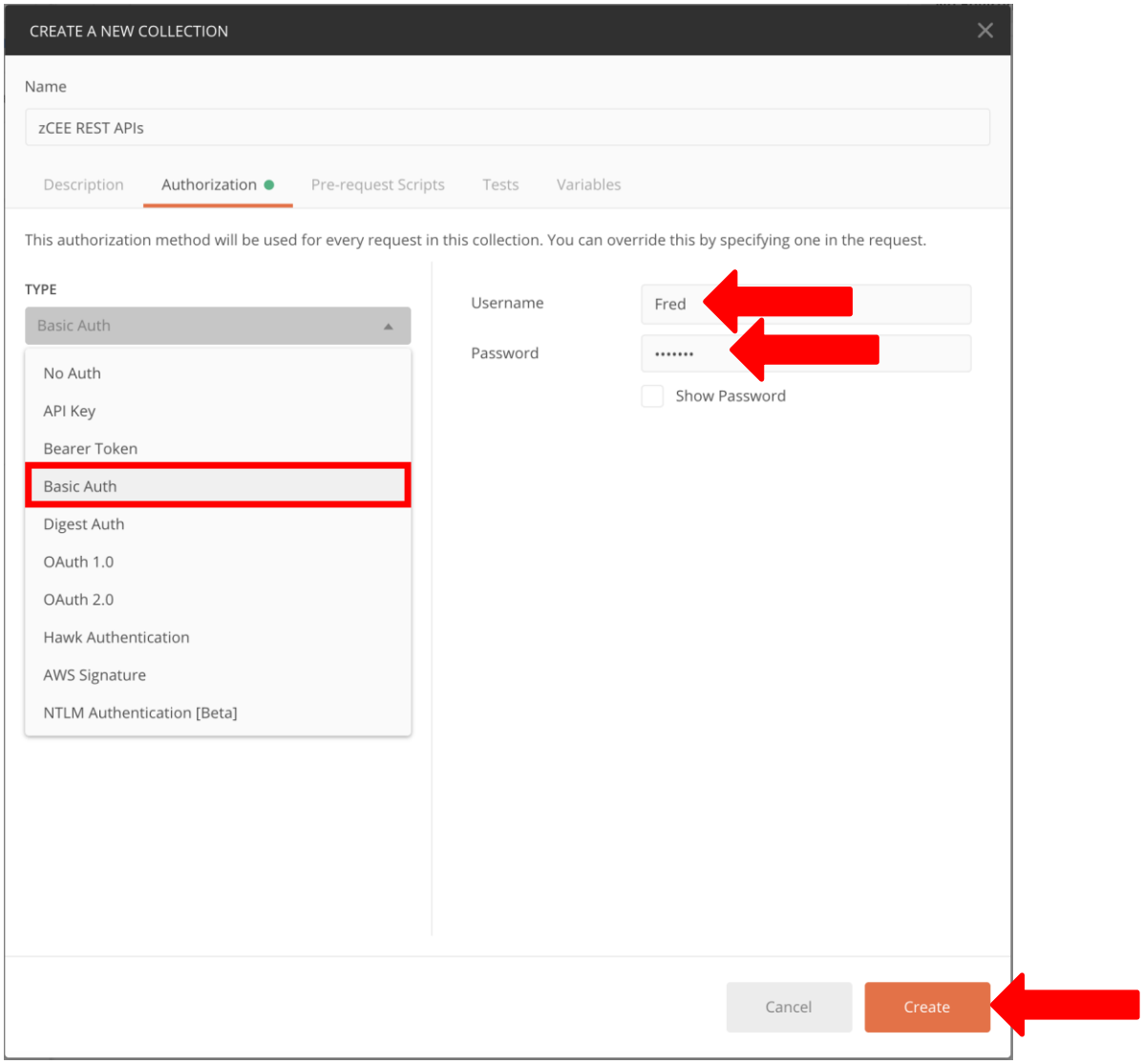
__iii. Name the collection "zCEE REST APIs". Then click the **Authorization** tab, to set the authentication for the collection.



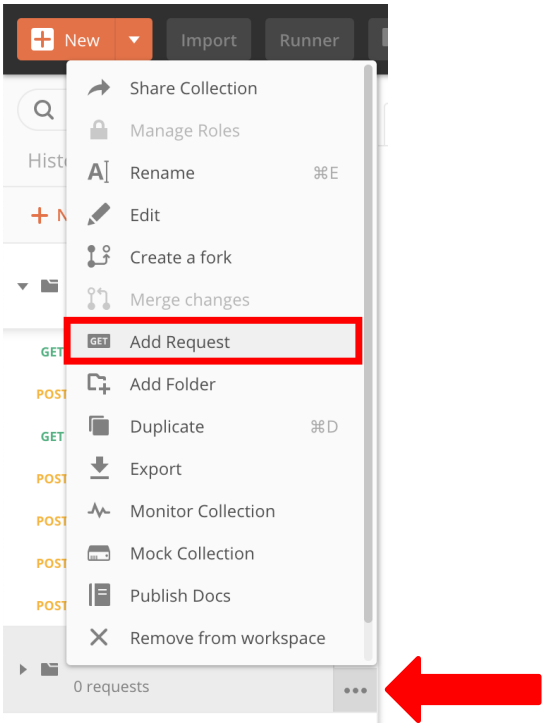
__iv. Select **Basic Auth** from the dropdown menu under **Type** and enter the User ID and Password for the ADMIN since we will be connecting through zCEE, **NOT** natively. Then click **Create**.

(1) Username: **Fred**

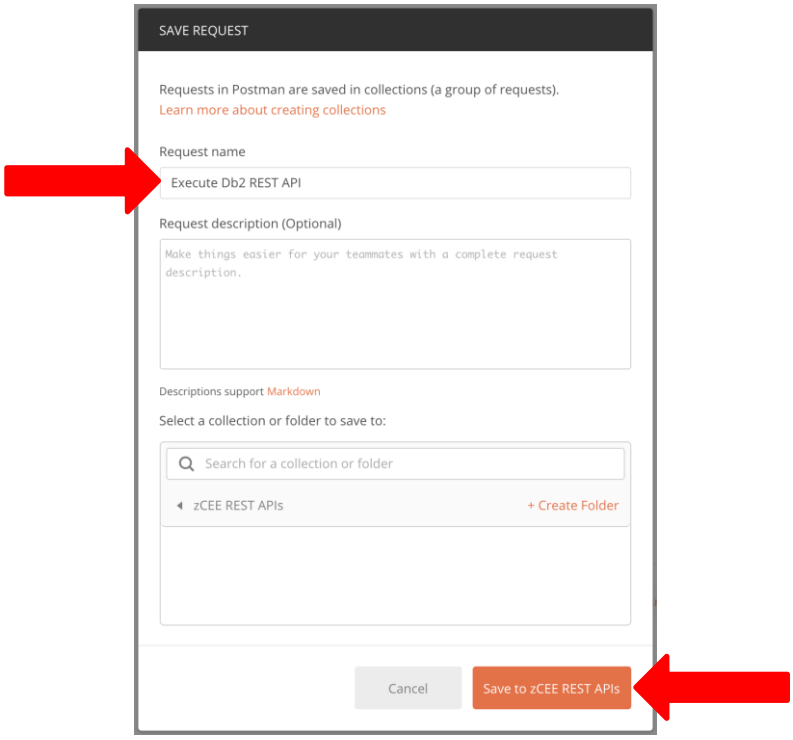
(2) Password: **fredpwd**



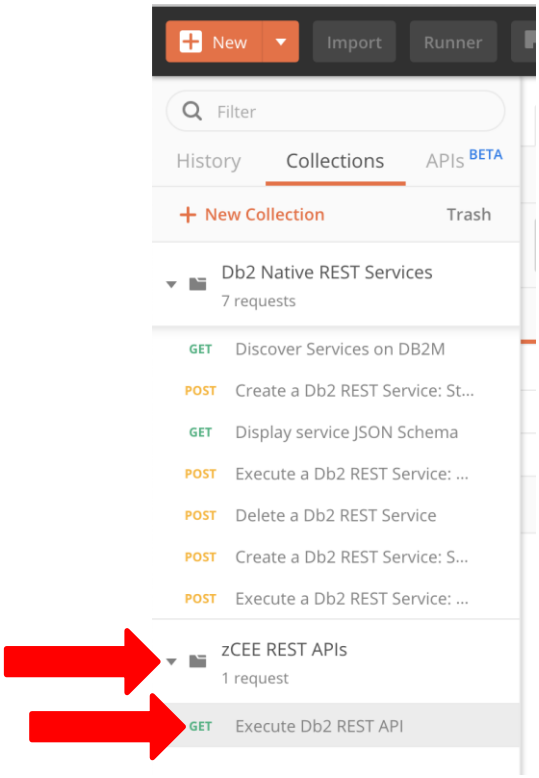
B. To create a test request, add a new request by clicking on the ellipses in the corner of the "zCEE REST APIs" collection and select **Add Request**.



C. Name the request "Execute Db2 REST API" and click **Save to zCEE REST APIs.**



D. Open the newly created request by clicking on the "zCEE REST APIs" collection in the side navigation bar and then the "Execute Db2 REST API" request.



E. Update the request with the following information:

__i. Set the REST method to: **GET**

__ii. Add the URL:

https://wg31.washington.ibm.com:9453/selectEmployee/employee/000020

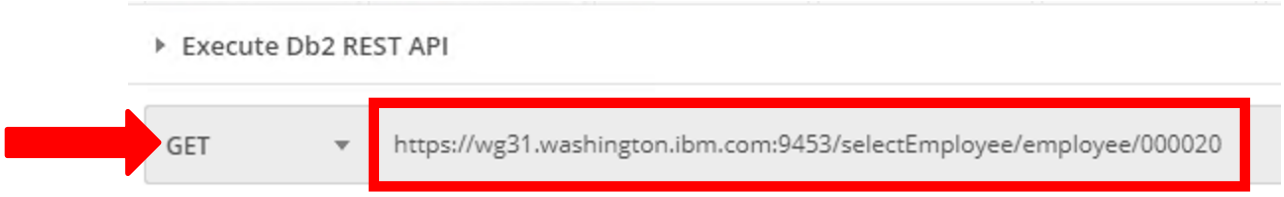
Note – the **secure connection (HTTPS)** and the **camelCase API name**

(1) TCP/IP information = **https://wg31.washington.ibm.com:9453**

(2) REST API name = **selectEmployee**

(3) GET method path = /employee/{employeeNumber}

(4) Input department number = 000020



F. Click **SEND** to invoke the command.

G. Click **SAVE** to save the request.

H. Review the Response Body.

__i. The employee information for employee number “000020” will be listed in the Response Body.

__ii. Note the Phone Number information for the employee is **NOT** listed as expected.

Summary: In this section, tested the API using Swagger and the Postman client.

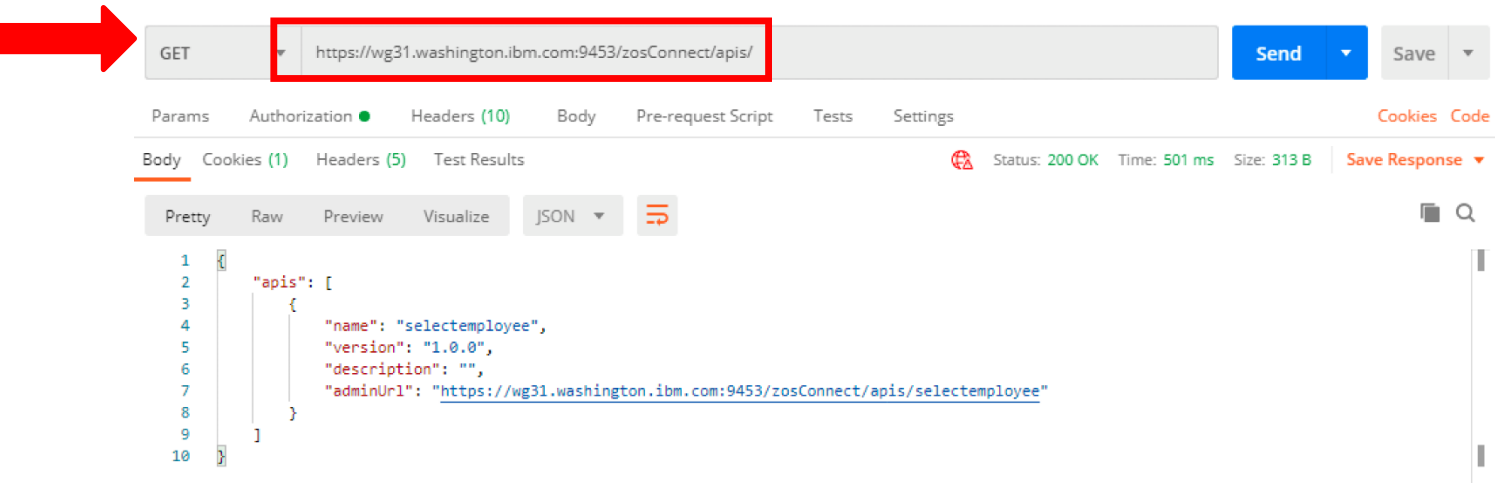
2.2 Additional information available for the REST API

__1. List installed REST APIs

A. URL = **https://wg31.washington.ibm.com:9453/zosConnect/apis**

B. Method = **GET**

C. Output API information = adminUrl



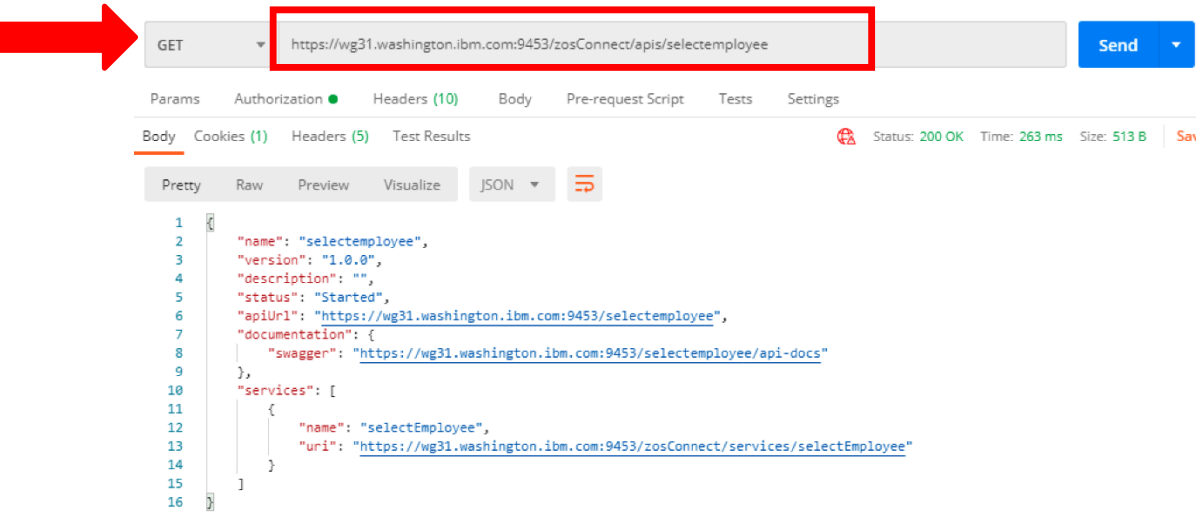
__2. REST API administration information

A. URL =

https://wg31.washington.ibm.com:9453/zosConnect/apis/selectEmployee

B. Method = **GET**

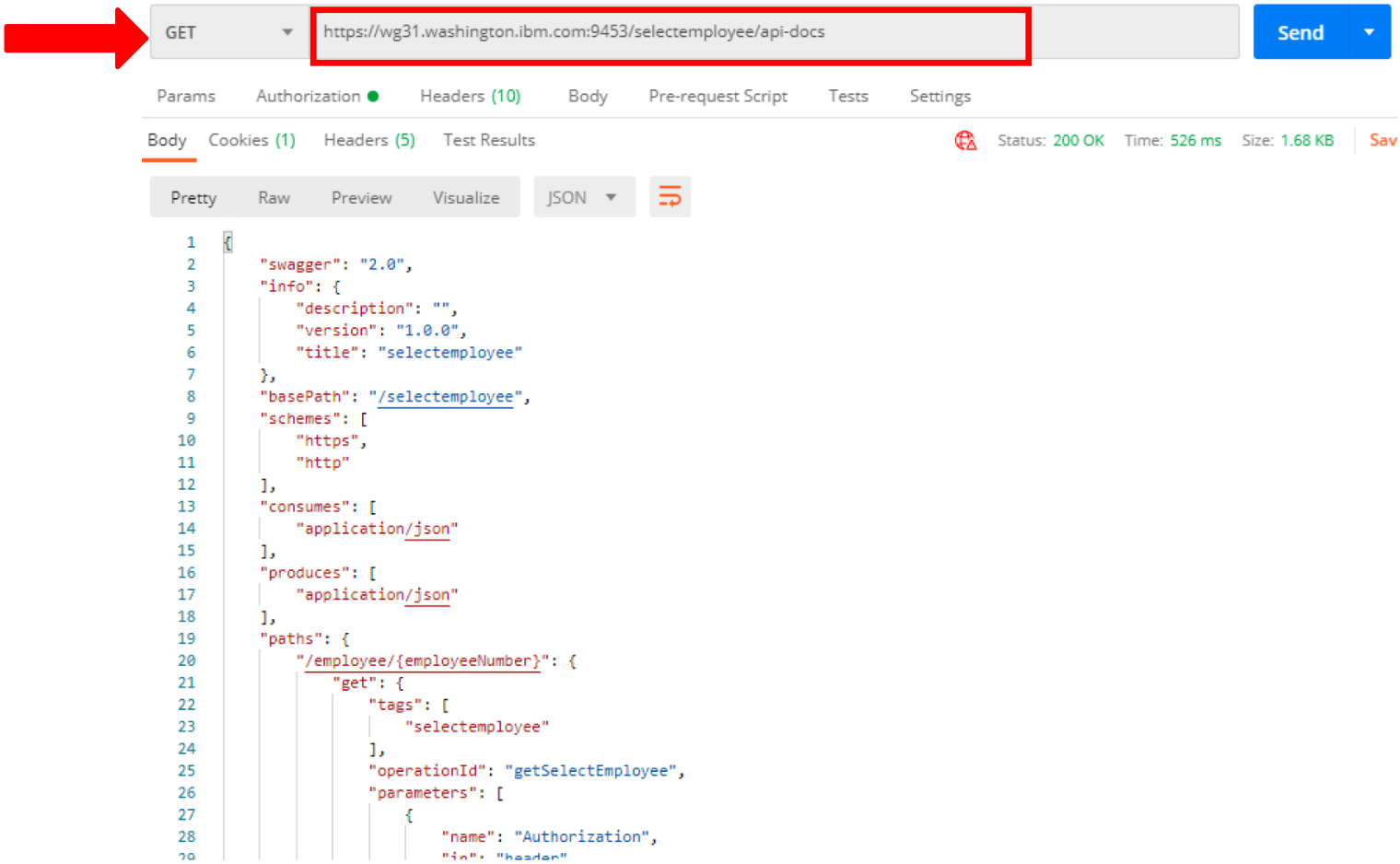
C. Output API information = Swagger



__3. REST API Swagger information

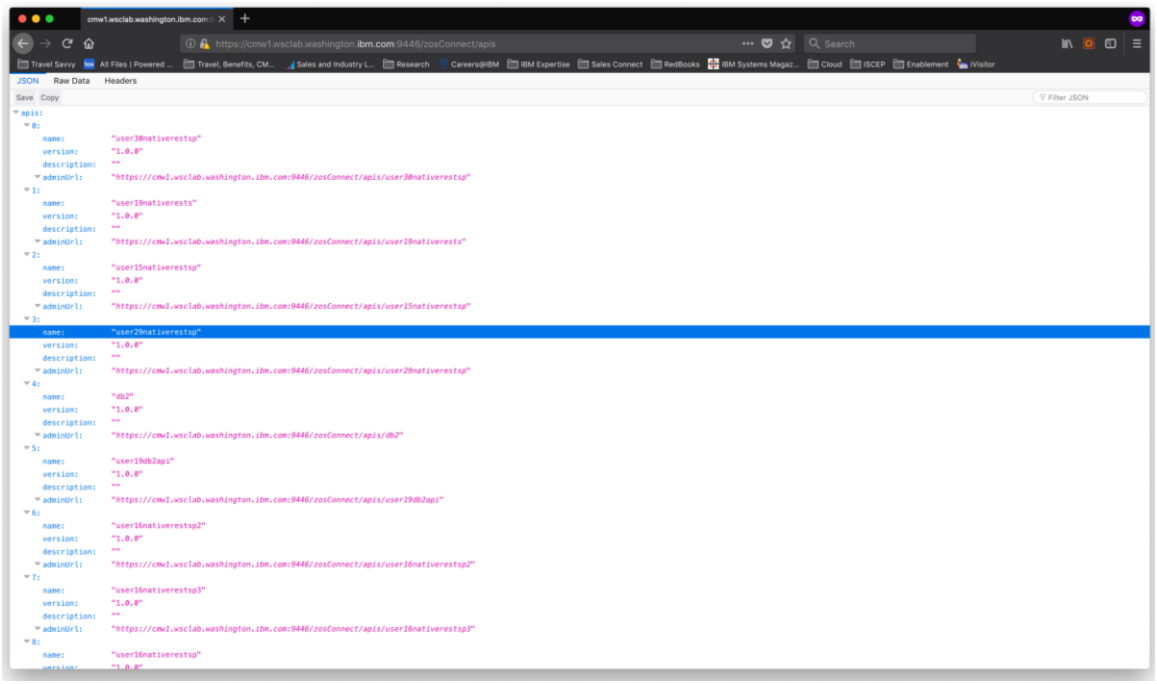
A. URL = **https://wg31.washington.ibm.com:9453/selectEmployee/api-docs**

B. Method = **GET**



__4. Additional information can be found by opening a web page with url =

https://wg31.washington.ibm.com:9453/zosConnect/apis



2.3 Summary

During this exercise, you performed the four steps to create a Db2 REST API. You collected the JSON schema information needed and created the SAR for the Db2 REST service, then you mapped a Db2 POST method using the SAR to a GET method. You learned how to use the API Editor's mapping feature to create an input variable and map constants to a JSON request and response schema used by Db2. Finally, you tested the API using Swagger and Postman.

Appendix A.

Deleting a Db2 REST Service using the Db2ServiceManager

You can use the Db2 REST service manager API to drop a user-defined service if you have the required authority. Dropping a service removes the service, frees its associated package, and deletes the corresponding row from the SYSIBM.DSNSERVICE catalog table. The FREE PACKAGE (DSN) command contains information about the required DROP privileges or authorities.

There is no difference in dropping a service that uses a stored procedure or a SQL statement to retrieve the data as dropping the service only requires the unique name of the service to be provided as a parameter. The following instructions walk through this process and the service named **selectByDeptSTMT** will be dropped.

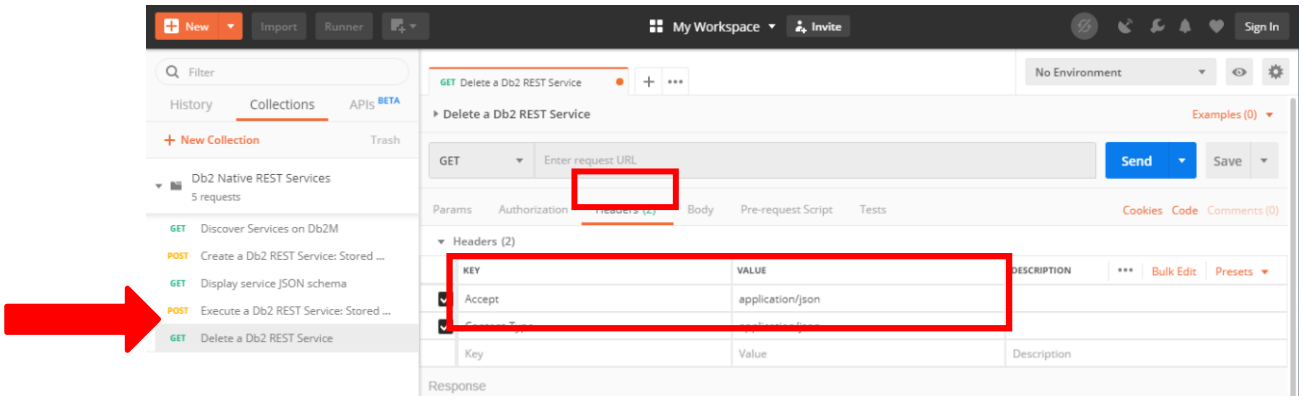
__1. Create a new request in the “Db2 Native REST Services” Collection, as done before.

__2. Name the request “Delete a Db2 REST Service” and click **Save to Db2 Native REST Services**.

__3. Open the new request from the collection in the side navigation bar and navigate to the **Headers** tab to create the appropriate REST Header fields.

A. In the **Key** column type “Accept”, and in its corresponding **Value** column type “application/json”.

B. Add another header by typing “Content-Type” in the **Key** column, and in its corresponding **Value** column type “application/json”.



__4. Update the request with the information below to delete the Db2 service we created in step 1.2. The Db2 create service information resides in the JSON body.

A. Set the REST Method to: **POST**

B. Add the Db2 REST URL:

http://wg31.washington.ibm.com:2446/services/DB2ServiceManager

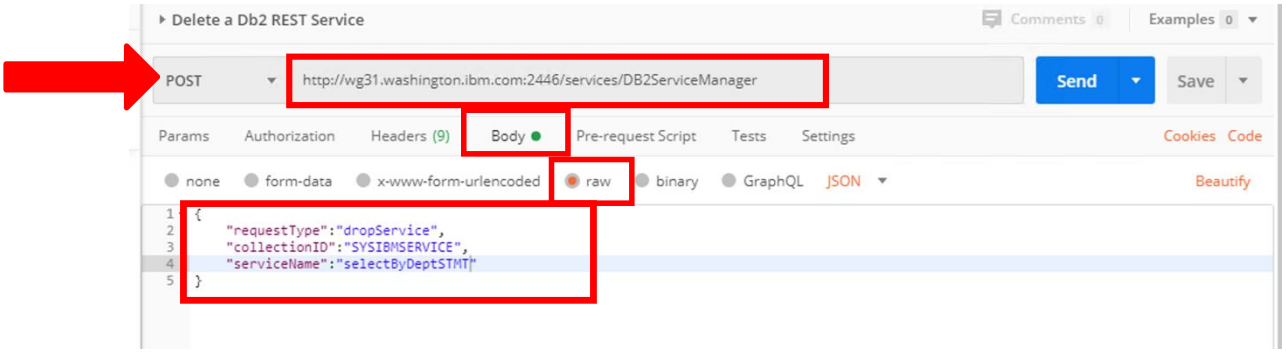
__i. Db2 DNS name = **wg31.washington.ibm.com**

__ii. Db2 port = **2446**

__iii. Db2 Service URI = **/services/DB2ServiceManager**

C. Create the dropService JSON Body by entering all the fields needed to delete a Db2 REST service, provided below. Click on the **Body** tab and then the **raw** radio button.

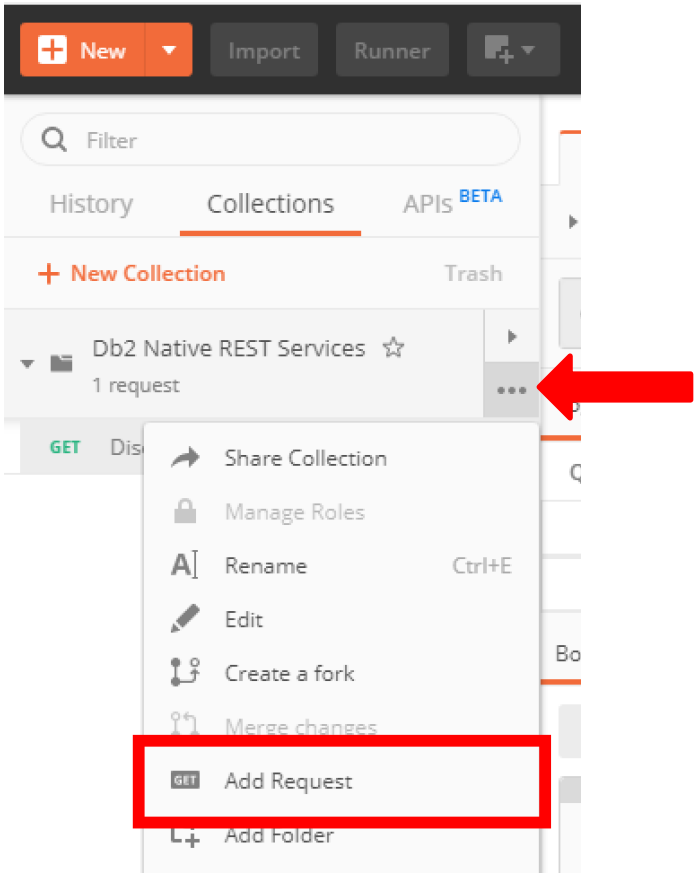
```
{
"requestType": "dropService",
"collectionID": "SYSIBMService",
"serviceName": "selectByDeptSTMT"
}
```



__5. Delete Service Response. The proper output would be the output provided below:

A. The StatusCode is **200 OK** and that the StatusDescription indicates that the service was dropped successfully.

__1. Add a new request to the “Db2 Native REST Services” Collection, as detailed before in **section 1.2.2 step 1**.

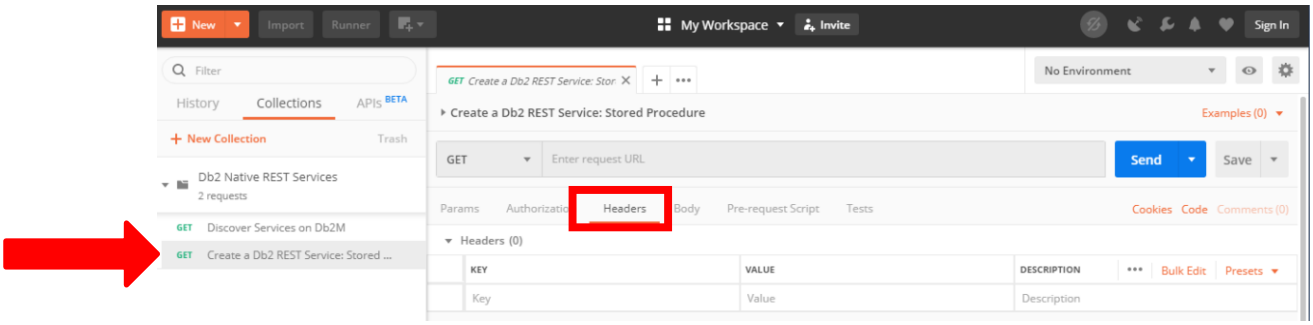


__2. At this point you should decide if you plan to use the existing stored procedure in the request or a simple SQL statement. For organization and clarity, please choose the corresponding name for your request below. Regardless of which option you choose, the following steps 3-5 need to be executed. Only at step 6 should you follow the specific instructions for your request.

If using a **stored procedure**, name the request “Create a Db2 REST Service: Stored Procedure”.
If using a **SQL statement**, name the request “Create a Db2 REST Service: SQL Statement”.

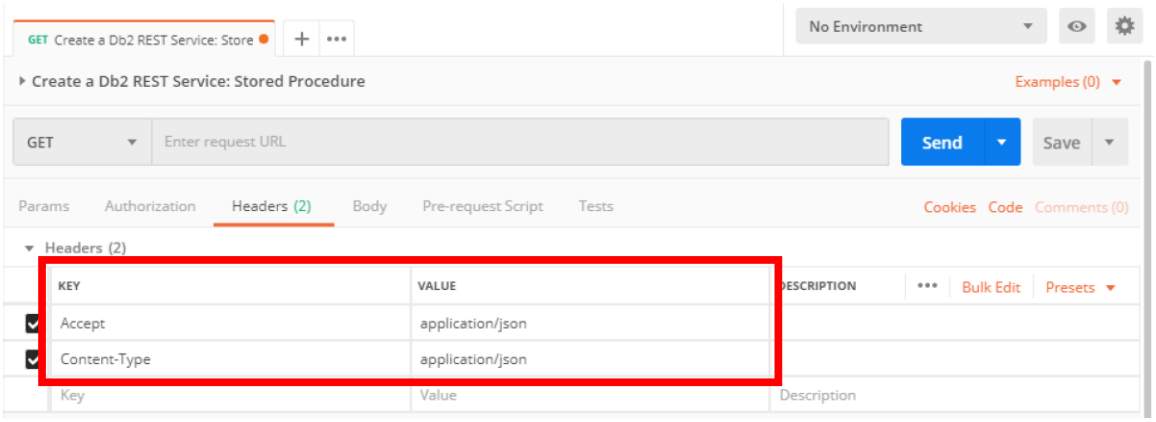
__3. Click **Save to Db2 Native REST Services**.

__4. Open the new request from the collection and navigate to the **Headers** tab. You will next create the appropriate REST header fields that we introduced at the beginning of this section.



A. In the **Key** column type “Accept”, and in its corresponding **Value** column type “application/json”.

B. Add another header by typing “Content-Type” in the **Key** column, and in its corresponding **Value** column type “application/json”.



__5. Update the request with the information below to create a Db2 service. The required Db2 create service information resides in the JSON body in step 6.

A. Set the REST Method to: **POST**

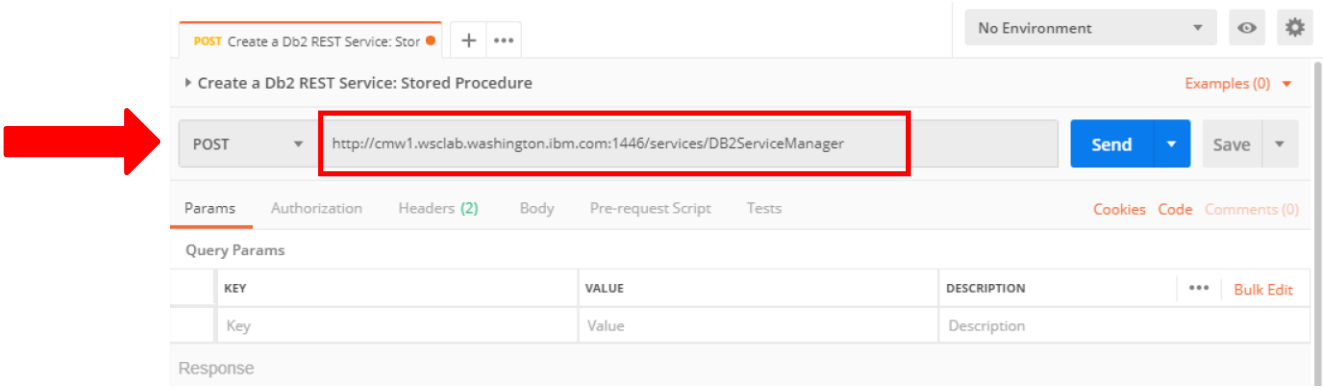
B. Add the DB2ServiceManager URL:

http://wg31.washington.ibm.com:2446/services/DB2ServiceManager

__i. Db2 DNS name = **wg31.washington.ibm.com**

__ii. Db2 port = **2446**

__iii. Db2 Service Manager URI= **/services/DB2ServiceManager**

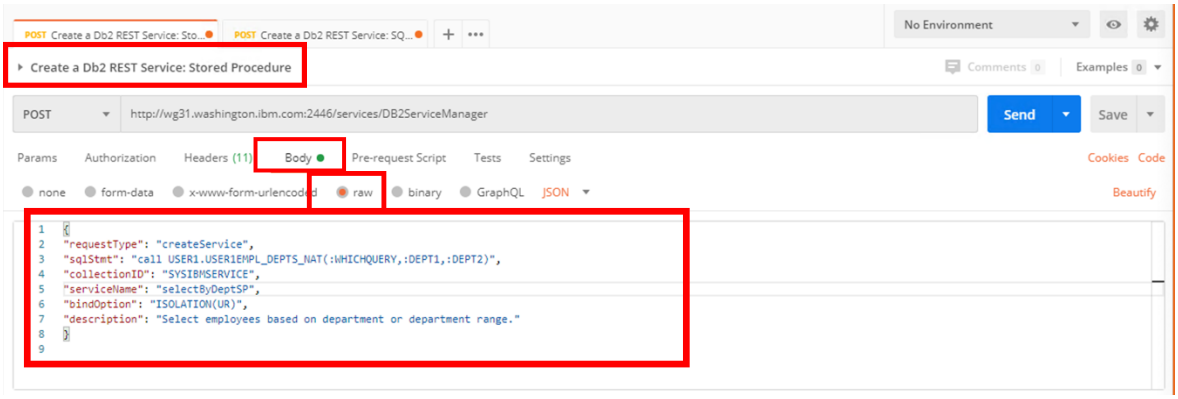


C. In the body section of the request, add the appropriate JSON information for either the stored procedure call or the SQL statement detailed in step 6 below.

__6. To create a Db2 REST service using a **Stored Procedure** fill in the request body information with the JSON found in **step A** below.
To create a Db2 REST service using a **SQL Statement** fill in the request body information with the JSON found in **step B** below.

A. **FOR STORED PROCEDURE**:
Create the createService JSON Body by entering all the fields required to create a Db2 REST service, provided below. Click on the **Body** tab and then the **raw** radio button.

```
{
"requestType": "createService",
"sqlStmt": "call USER1.EMPL_DEPTS_NAT(:WHICHQUERY,:DEPT1,:DEPT2)",
"collectionID": "SYSIBMService",
"serviceName": "selectByDeptSP",
"bindOption": "ISOLATION(UR)",
"description": "Select employees based on department or department range."
}
```



Below are the details regarding the JSON input parameter values:

__i. requestType: createService indicates that you request to create a new service

__ii. sqlStatement: The SQL statement to include in the new service
(1) Db2 Stored Procedure Call:
call USER1.USER1EMPL_DEPTS_NAT(:WHICHQUERY,:DEPT1,:DEPT2)
- Stored Procedure Name: USER1.USER1EMPL_DEPTS_NAT
- Host variables: ":WHICHQUERY,:DEPT1,:DEPT2"
  - These variable names will be used in Db2's JSON request schema
  - The host variable names begin with a colon, ":"
  - Variable names are the creator's choice

__iii. collectionID: The Db2 package collection identifier of the package that is associated with the new service
(1) SYSIBMSERVICE is the default REST service collection name

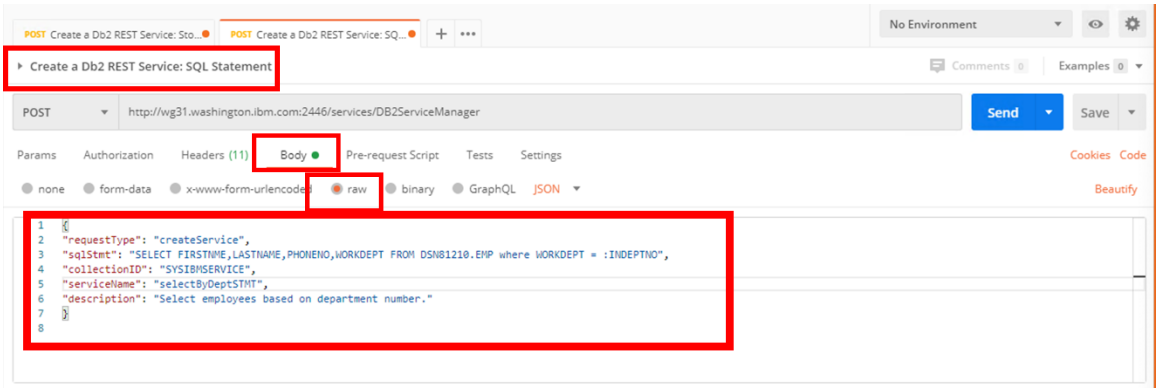
__iv. serviceName: The name of the service to create

__v. serviceDescription: A brief description of the new service

__vi. bindOption: The option that you specify for binding the package. The uncommitted read isolation level is only an example.

B. **FOR SQL STATEMENT**:
Create the createService JSON Body by entering all the fields required to create a Db2 REST service, provided below. Click on the **Body** tab and then the **raw** radio button.

```
{
"requestType": "createService",
"sqlStmt": "SELECT FIRSTNME,LASTNAME,PHONENO,WORKDEPT FROM
 DSN81210.EMP where WORKDEPT = :INDEPTNO",
"collectionID": "SYSIBMService",
"serviceName": "selectByDeptSTMT",
"description": "Select employees based on department number."
}
```



Below are the details regarding the JSON input parameter values:

__i. requestType: createService indicates that you request to create a new service

__ii. sqlStatement: The SQL statement to include in the new service
(1) Db2 SQL Statement:
SELECT FIRSTNME, LASTNAME, PHONENO, WORKDEPT FROM DSN81210.EMP where WORKDEPT = :INDEPTNO
- Host variable: ":INDEPTNO"
- The variable name will be used in Db2's JSON request schema
- The host variable names begin with a colon ":"
- The variable name is the creator's choice

__iii. collectionID: The Db2 package collection identifier of the package that is associated with the new service
(1) SYSIBMSERVICE is the default REST service collection name

__iv. serviceName: The name of the service to create

__v. serviceDescription: A brief description of the new service

Information

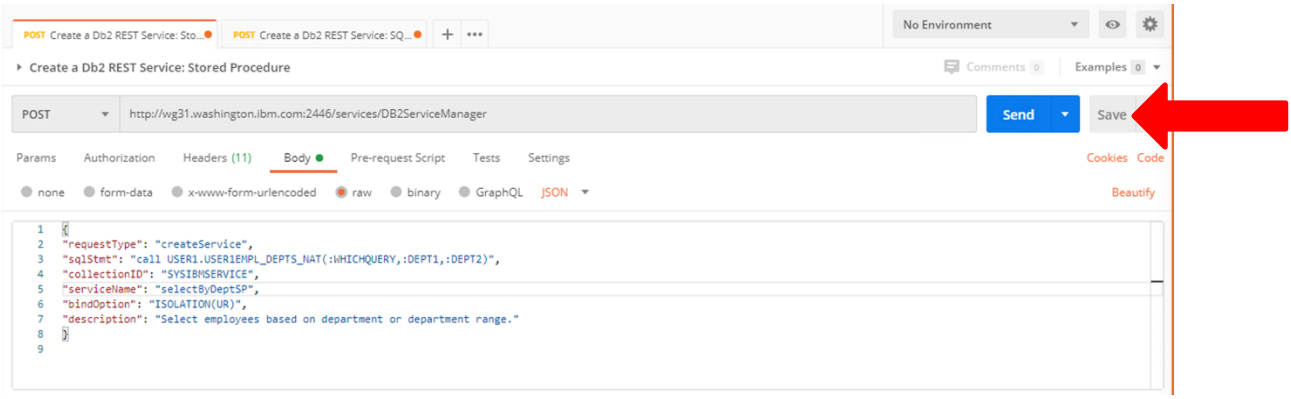
Note: For complete details about the Db2 createService parameters, please visit Creating a Db2 REST service in the IBM Knowledge Center.

__7. For the remainder of this section, whether you created the service using the provided stored procedure and/or a simple SQL statement, the following steps for each must be executed. Click **SEND** to invoke the command.

__8. Create Service Response. Confirm the proper output below:

A. The HTTP status code **201 Created** is the expected status coded for successful Db2 service creation. If you did not receive a successful return code, go back and make sure you input the correct Header in formation and copied the correct JSON body information for your request.

__9. **SAVE** the changes made to the request.



Summary: In this step we created a new Db2 Native REST service using the DB2ServiceManager API. Successful execution was accomplished by providing the parameter "createService" for the "requestType" field, and using either the EMPL_DEPTS_NAT stored procedure and/or a SQL statement for the SQL Execution.

Troubleshooting REST service requests

Common HTTP status codes for REST service error condition

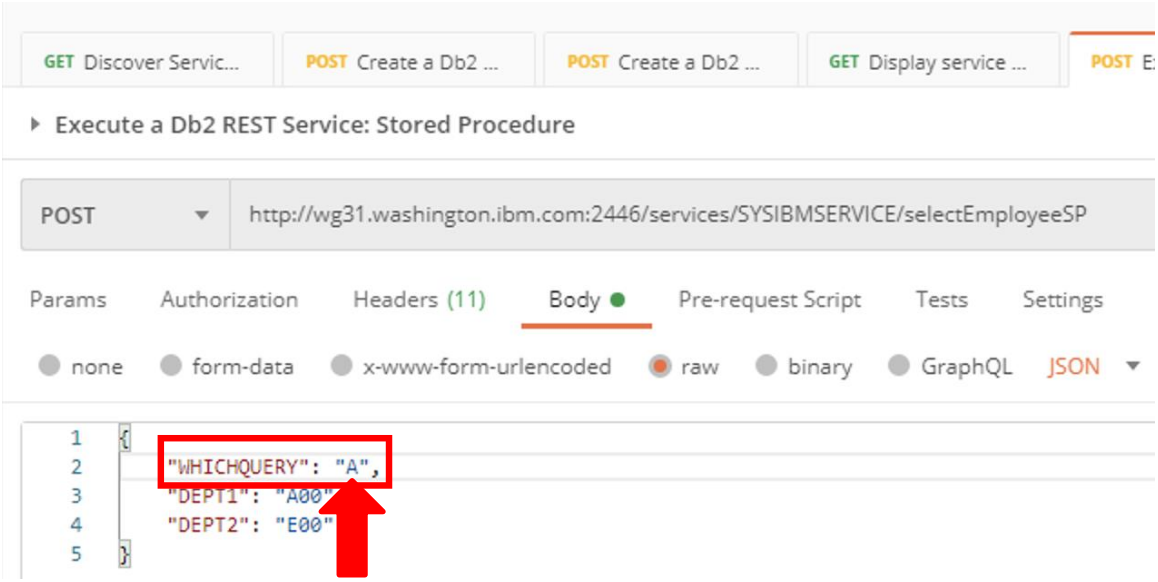
HTTP status code	Description
HTTP 500 (Internal Server Error)	Indicates that the server could not fulfill a request. In most cases, the HTTP status code is accompanied by a DB2 SQL code that provides more details about the error condition.
HTTP 400 (Bad Request)	Indicates a problem with an input parameter, such as a missing required input parameter, that is detected by the DB2 DDF native REST code prior to executing the DB2 SQL statement. This code is also used for many DB2ServiceManager failures (for example, Create/Drop service) and DB2DiscoverService failures (discover service/discover service details), which are typically caused by incorrect or missing inputs.
HTTP 401 (Unauthorized)	Indicates that the user could not be successfully authenticated.
HTTP 403 (Forbidden)	Indicates that the user might not have the required permissions to access a resource.

The following examples provide insight into some common errors that occur when executing Db2 Native REST Services.

Incorrect input parameter type

Request: Passing a parameter that does not match the expected type will result in a Server Error.

Ex: The input parameter for “WHICHQUERY” is a character when it should be an integer, which violates the schema for the service.

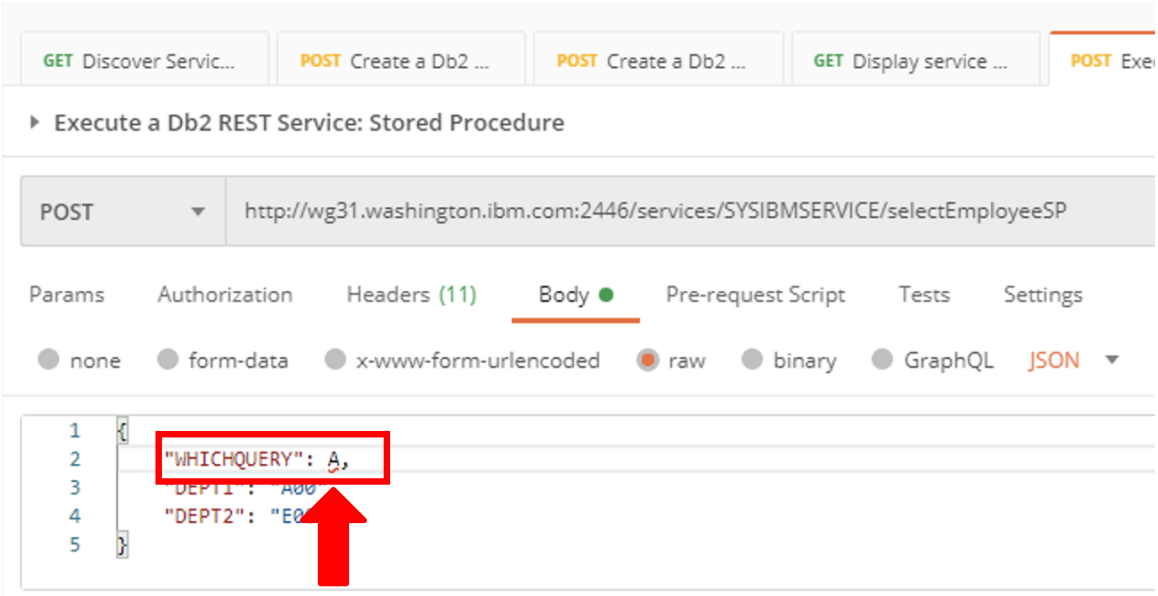


Response: A **500 Internal Server Error** is reported because Db2 was expecting an Integer Type but it received a Character Type instead.

Incorrect JSON formatting

Request: Passing a parameter with incorrect JSON formatting will result in a Bad Request error.

Ex: The “WHICHQUERY” parameter is passed as a character without quotes, which violates JSON formatting.



Information

Notice that Postman has underlined the character “A” in the JSON body with a red squiggle, indicating that the JSON entered in not formatted properly. Hovering over the underlined character provides more information on what the error is.

Response: A **400 Bad Request** is reported because the transaction is expecting properly formatted JSON Body as the request body.

Appendix B.

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106-0032, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction and intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental. All references to fictitious companies or individuals are used for illustration purposes only.

Trademarks and copyrights

The following terms are trademarks of International Business Machines Corporation in the United States, other countries, or both:

IBM	AIX	CICS	ClearCase	ClearQuest	Cloudscape
Cube Views	Db2	developerWorks	DRDA	IMS	IMS/ESA
Informix	Lotus	Lotus Workflow	MQSeries	OmniFind	
Rational	Redbooks	Red Brick	RequisitePro	System i	
System z	*Tivoli*	*WebSphere*	*Workplace*	*System p*	

Adobe, the Adobe logo, PostScript, and the PostScript logo are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, and/or other countries.

IT Infrastructure Library is a registered trademark of the Central Computer and Telecommunications Agency which is now part of the Office of Government Commerce.

Intel, Intel logo, Intel Inside, Intel Inside logo, Intel Centrino, Intel Centrino logo, Celeron, Intel Xeon, Intel SpeedStep, Itanium, and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

ITIL is a registered trademark, and a registered community trademark of The Minister for the Cabinet Office, and is registered in the U.S. Patent and Trademark Office.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

Cell Broadband Engine is a trademark of Sony Computer Entertainment, Inc. in the United States, other countries, or both and is used under license therefrom.

Linear Tape-Open, LTO, the LTO Logo, Ultrium, and the Ultrium logo are trademarks of HP, IBM Corp. and Quantum in the U.S. and other countries.

The information contained in these materials is provided for informational purposes only, and is provided AS IS without warranty of any kind, express or implied. IBM shall not be responsible for any damages arising out of the use of, or otherwise related to, these materials. Nothing contained in these materials is intended to, nor shall have the effect of, creating any warranties or representations from IBM or its suppliers or licensors, or altering the terms and conditions of the applicable license agreement governing the use of IBM software. References in these materials to IBM products, programs, or services do not imply that they will be available in all countries in which IBM operates. This information is based on current IBM product plans and strategy, which are subject to change by IBM without notice. Product release dates and/or capabilities referenced in these materials may change at any time at IBM's sole discretion based on market opportunities or other factors, and are not intended to be a commitment to future product or feature availability in any way.


Please Recycle

IBM Software