

IBM z/OS Connect EE V3.0

# Developing RESTful APIs for HATS REST Services



**IBM Z**  
**Wildfire Team –**  
**Washington System Center**

# Table of Contents

<b>Overview .....</b>	<b>3</b>
<b><i>Review the Stock Trader Application.....</i></b>	<b>4</b>
<i>Logging into the application .....</i>	4
<i>The Company Selection screen.....</i>	5
<i>The Options selection screen.....</i>	6
<i>The Real-Time Quote screen .....</i>	6
<i>The Shares – Buy screen .....</i>	7
<i>The Shares – Sell screen .....</i>	7
<b><i>Generating the Services for the Stock Trader Application.....</i></b>	<b>8</b>
<i>Background.....</i>	8
<i>Generating the Service Archive (SAR) files.....</i>	10
<i>Deploying the Service Archive (SAR) files .....</i>	12
<b><i>Connect to the IBM z/OS Explorer to the z/OS Connect EE Server.....</i></b>	<b>15</b>
<b><i>Create the HATS API project.....</i></b>	<b>18</b>
<i>Import the SAR files into the API Toolkit.....</i>	20
<i>Compose an API for HATS Rest Services .....</i>	23
<b><i>Deploy the API to a z/OS Connect EE Server .....</i></b>	<b>31</b>
<b><i>Test the HATS APIs .....</i></b>	<b>33</b>
<i>Test the API using Swagger UI.....</i>	36

**Important:** On the desktop there is a file named *Developing APIs CopyPaste.txt*. This file contains commands and other text used in this workshop. Locate that file and open it. Use the copy-and-paste function (**Ctrl-C** and **Ctrl-V**) to enter commands or text. It will save time and help avoid typo errors. As a reminder text that appears in this file will be highlighted in yellow.

## Overview

**Important – You do not need any skills with HATS to perform this exercise. Even if HATS is not relevant to your current plans performing this exercise will give additional experience using the API Toolkit with developing APIs.**

The objective of these exercises is to gain experience with working with z/OS Connect EE and the API Toolkit. These two products allow the exposure of z/OS resources to JSON clients. More in-depth information about the customization of z/OS Connect EE, z/OS Connect EE security, the use of the API Toolkit and other topics is provided by the 1-day *ZCONEE – z/OS Connect Workshop*. For information about scheduling this workshop in your area contact your IBM representative.

## *General Exercise Information and Guidelines*

- ✓ This exercise requires using z/OS user identity *USER1*. The password for this user will be provided by the lab instructor.
- ✓ Any time you have any questions about the use of IBM z/OS Explorer, 3270 screens, features or tools do not hesitate to ask the instructor for assistance.
- ✓ Please note that there may be minor differences between the screen shots in this exercise versus what you see on your desktop. These differences should not impact the completion of this exercise.
- ✓ Text in **bold** and highlighted in **yellow** in this document should be available for copying and pasting in a file named *Development APIs CopyPaste* file on the desktop.
- ✓ For information regarding the use of the Personal Communication 3270 emulator, see the *Personal Communications Tips* PDF in the exercise folder.

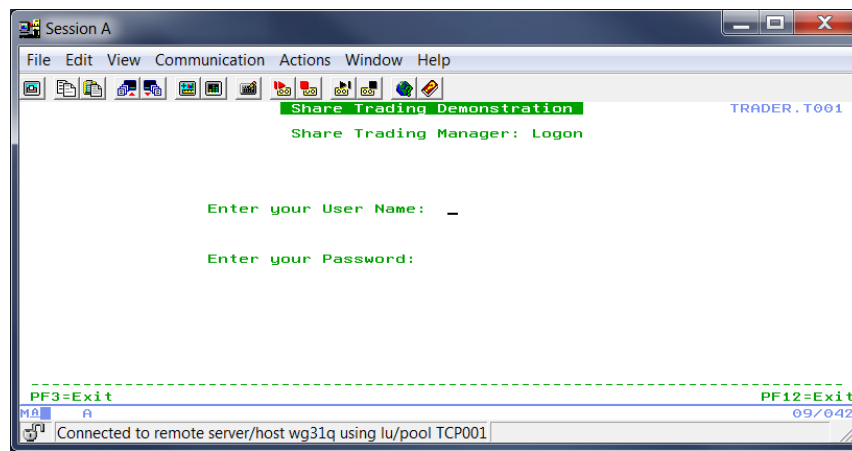
## *Review the Stock Trader Application*

The Stock Trader is a CICS application that uses BMS to display screens in a 3270-terminal session. A user of the application is required to ‘log on’ to the application and this log-on identity is used to identity the customer whose stock portfolio will be managed.

Once logged into the application, the user is presented with a list of companies supported by the application. The user selects which company to work with by entering the number of the company as displayed on the screen. Note this list is dynamic and subject to change. The next screen provides 3 options, one option is to display the details of the stock and the customer’s current stock holdings in this company, the other options provide the opportunity to buy or sell shares of this company’s stock.

### *Logging into the application*

The application is started by entering CICS transaction **TRDR** in a CICS terminal session. This will display the *Logon* screen displayed below.



If CICS security is enabled a valid RACF user identity and password must be used to log on to this application. But in this environment CICS security is not enabled so any values can be entered and will be accepted. Note that the *User Name* is used as the customer or owner when stocks are being bought or sold.

Navigation between screens is performed by pressing the **Enter** key when moving forward in the application and or by using the **F3** key to backup one screen or **F12** to exit the application entirely.

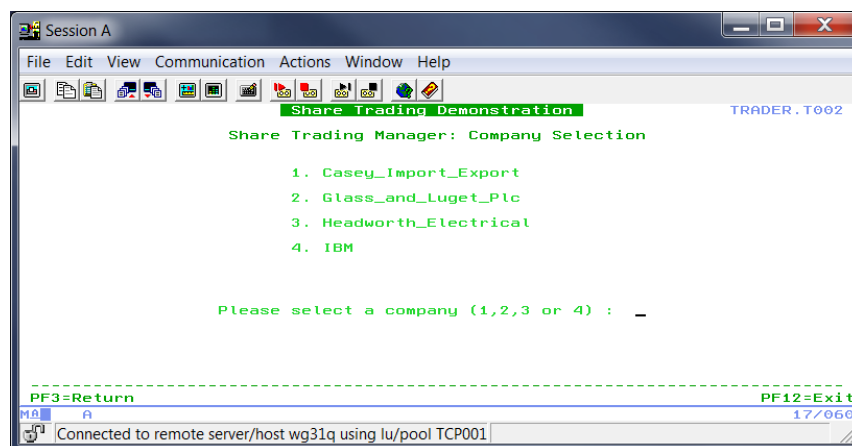
The 3270-emulator used for this workshop (IBM Personal Communication) maps the 3270 enter key to the right **Ctrl** key (see below). Any references to the *Enter* key in non-3270 windows, OMVS terminal session, etc. refers to the key labeled *Enter* on the keyboard.



N.B., Most of the 3270 screen shots in the remainder of this exercise are shown in reverse video simply for printing purposes.

### *The Company Selection screen*

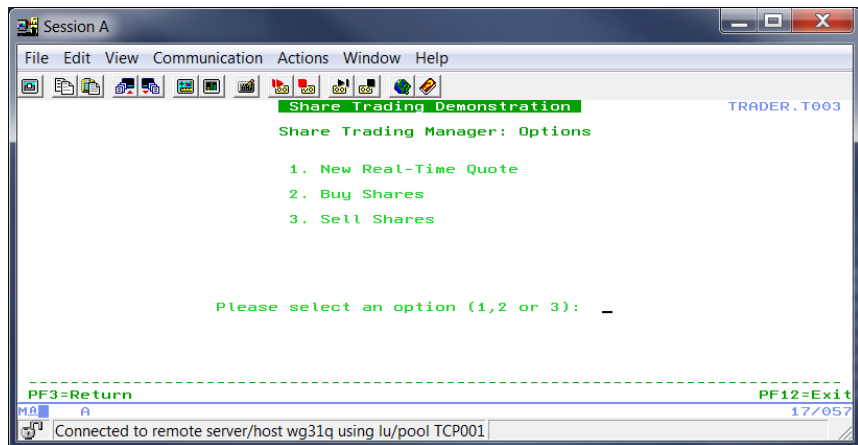
Once you have entered a *User Name* and *Password* and pressed **Enter** the screen below is displayed. The contents of the screen are built by using the contents of a VSAM data set to obtain the list of companies currently supported and this screen will be the same for all users.



On this screen select one of the companies by entering the number (1,2,3 or 4) beside the company name and pressing **Enter** to continue.

## The Options selection screen

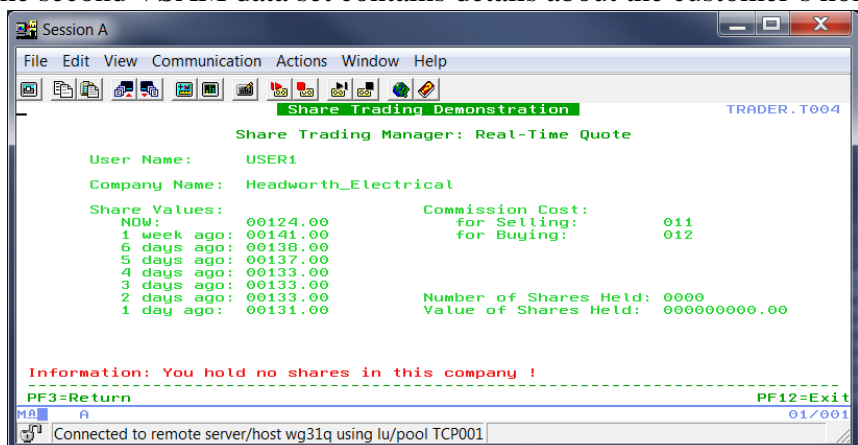
Regardless which company is selected when you press the **Enter** key the screen below is shown:



As you can see there are three options when dealing with the selected company's stock. Option 1 displays the stock's current value, history, commission details and details about current holdings of this company's stock by customer (*Real-Time Quote*). Options 2 and 3 are for buying or selling shares of this company for the customer (*Shares - Buy* or *Shares - Sell*).

## The Real-Time Quote screen

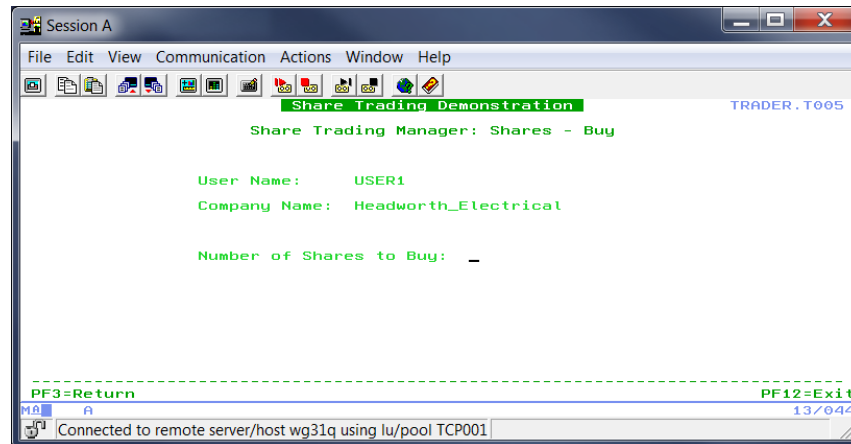
If option 1 is selected the *Real-Time Quote* screen is displayed. The information on this screen is derived from the contents of the two VSAM data sets used by this application. One VSAM data set contains details about the company's stock and the second VSAM data set contains details about the customer's holdings of this stock.



Note that a warning or informational message will be displayed in red at the bottom of the screen when necessary.

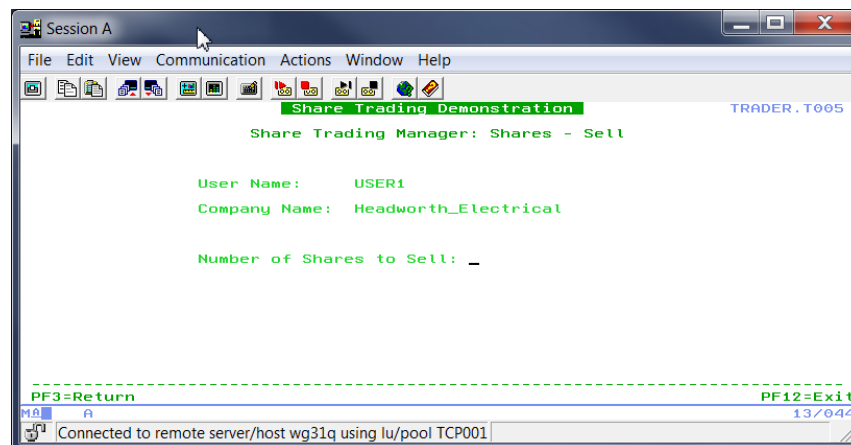
## *The Shares – Buy screen*

If option 2 is selected the *Shares – Buy* screen is displayed. On this screen the user is given an opportunity to buy additional shares of this company's stock. The number of shares to buy is entered and the customer's record in the VSAM data set is updated.



## *The Shares – Sell screen*

If option 3 is selected the *Shares – Sell* screen is displayed. On this screen the user is given an opportunity to sell shares of this company's stock. The number of shares to sell is entered and the customer's record in the VSAM data set is updated with this information. A warning message will be displayed if this customer does not own the number of shares that are being requested to sell.



# Generating the Services for the Stock Trader Application

## Background

The HATS Toolkit was used to develop 4 services or flows through these 3270 screens. The first service goes to the *Company Selection* screen to obtain the list of companies. Note as stated earlier this list is dynamic and is built on the contents of a VSAM and therefor the sequence number for a company is subject to change. The second service will return the stock values, commission costs and holding of the selected company's stock for the customer. The third and fourth services provide an opportunity to buy or sell shares. All four of these services were configured and deployed in the HATS runtime as REST services.

The swagger document for these four HATS REST services can be found in directory *C:/HATLab* on your workstation as file *Trader.swagger*. This swagger document was obtained from HATS using URL <https://wg31.washington.ibm.com:29443/Trader/rest/swagger.json>

This swagger document was used to generate the artifacts required for the z/OS Connect EE build toolkit to build the service archive (SAR) files required to develop and deploy RESTful APIs using the z/OS Connect EE Toolkit.

- For example, the swagger document contained the contents below to describe in JSON schema format the request message for the Real-Time Quote (*getQuoteValue*) service.

```
"GetQuoteValue_invokeGetQuoteValue_Input_Properties": {  
  "type": "object",  
  "properties": {  
    "company": {  
      "type": "string"  
    },  
    "password": {  
      "type": "string"  
    },  
    "userName": {  
      "type": "string"  
    }  
  }  
},
```

The area in gray above was copied into a file that will be used as input to the z/OS Connect EE build toolkit to describe the request schema layout for this REST service, e.g. *getQuoteValueRequest.json*.



- The swagger document also contained the contents below which describes in JSON schema format the response message for the Real-Time Quote (*getQuoteValue*) service.

```
"GetQuoteValue_invokeGetQuoteValue_Output_Properties": {
  "type": "object",
  "properties": {
    "buyCommission": {
      "type": "string"
    },
    "companyName": {
      "type": "string"
    },
    "message": {
      "type": "string"
    },
    "numberOfSharesHeld": {
      "type": "string"
    },
    "sellCommission": {
      "type": "string"
    },
    "value1dayAgo": {
      "type": "string"
    },
    "value1weekAgo": {
      "type": "string"
    },
    "value2daysAgo": {
      "type": "string"
    },
    "value3daysAgo": {
      "type": "string"
    },
    "value4daysAgo": {
      "type": "string"
    },
    "value5daysAgo": {
      "type": "string"
    },
    "value6daysAgo": {
      "type": "string"
    },
    "valueNow": {
      "type": "string"
    },
    "valueOfSharesHeld": {
      "type": "string"
    }
  }
},
```

This area in gray above was copied into a file to be used as input to the z/OS Connect EE build toolkit to describe the response schema layout for this REST service, e.g. *getQuoteValueResponse.json*.

- The z/OS Connect EE build toolkit *zconbt* command requires a property file as input. This property file is used to identify the request and response schema files, the URL of the HATS REST service, the HATS HTTP verb, etc. The property file the Real-Time Quote service is shown below:

```
provider=rest
name=getQuoteValue
version=1.0
description=Get Share quote and values
requestSchemaFile=getQuoteValueRequest.json
responseSchemaFile=getQuoteValueResponse.json
verb=POST
uri=/trader/rest/GetQuoteValue
connectionRef=HatsConn
```

- Finally, the z/OS Connect EE build toolkit is invoked to build the service archive file for the REST services by creating a Windows batch file named *getCompany.bat* as shown below:

```
c:\z\zconbt\bin\zconbt.bat -p=getCompany.properties -f=getCompany.sar
```

### ***Generating the Service Archive (SAR) files.***

The first step you are required to perform is to generate the service archive (SAR) files using the z/OS Connect EE build toolkit using the artifacts as shown above.

- \_\_\_ 1. Open a DOS command prompt by opening the *Command Prompt* icon on the desktop.
- \_\_\_ 2. Use the change directory to go to directory C:\z\HATSLab, e.g. ***cd C:\z\HATSLab***.
- \_\_\_ 3. Generate the *getCompany.sar* file by invoking the build toolkit for the *getCompany* service by entering command ***getCompany***.
- \_\_\_ 4. Generate the *getquotevalue.sar* file by invoking the build toolkit for the *getquotevalue* service by entering command ***getQuoteValue***.
- \_\_\_ 5. Generate the *buyShares.sar* file by invoking the build toolkit for the *buyShares* service by entering command ***buyShares***.
- \_\_\_ 6. Generate the *sellShares.sar* file by invoking the build toolkit for the *sellShares* service by entering command ***sellShares***.

This is what you DOS session should look like.

```
c:\z\HATSLab>getcompany

c:\z\HATSLab>c:\z\zconbt\bin\zconbt.bat -p=getCompany.properties -f=getCompany.sar
BAQB0000I: z/OS Connect Enterprise Edition Build Toolkit Version 1.0
BAQB0001I: Creating service archive from configuration file getCompany.properties
BAQB0002I: Successfully created service archive file getCompany.sar

c:\z\HATSLab>getquotevalue

c:\z\HATSLab>c:\z\zconbt\bin\zconbt.bat -p=getQuoteValue.properties -f=getQuoteValue.sar
BAQB0000I: z/OS Connect Enterprise Edition Build Toolkit Version 1.0
BAQB0001I: Creating service archive from configuration file getQuoteValue.properties
BAQB0002I: Successfully created service archive file getQuoteValue.sar

c:\z\HATSLab>buyshares

c:\z\HATSLab>c:\z\zconbt\bin\zconbt.bat -p=buyShares.properties -f=buyShares.sar
BAQB0000I: z/OS Connect Enterprise Edition Build Toolkit Version 1.0
BAQB0001I: Creating service archive from configuration file buyShares.properties
BAQB0002I: Successfully created service archive file buyShares.sar

c:\z\HATSLab>sellshares

c:\z\HATSLab>c:\z\zconbt\bin\zconbt.bat -p=sellShares.properties -f=sellShares.sar
BAQB0000I: z/OS Connect Enterprise Edition Build Toolkit Version 1.0
BAQB0001I: Creating service archive from configuration file sellShares.properties
BAQB0002I: Successfully created service archive file sellShares.sar
```

## Deploying the Service Archive (SAR) files

Next the Service Archive (SAR) files need to be available to the z/OS Connect EE server. This is done by deploying them to the server's *services* directory. In this case the *services* directory is */var/ats/zosconnect/servers/zceeapir/resources/zosconnect/services*.

1. Open a DOS command session and change to directory C:\z\HATSLab e.g **cd c:\z\HATSLab**
2. Deploy the *getCompany* service by using the z/OS Connect EE RESTful administrative interface to deploy the service archive file by using the cURL command with a POST method, e.g.

```
curl -X POST --user Fred:fredpwd --data-binary @getCompany.sar --header "Content-Type: application/zip" --insecure https://wg31.washington.ibm.com:9453/zosConnect/services
```

```
c:\z\HATSLab>curl -X POST --user Fred:fredpwd --data-binary @getCompany.sar --header "Content-Type: application/zip" --insecure https://wg31.washington.ibm.com:9453/zosConnect/services
{"zosConnect":{"serviceName":"getCompany","serviceDescription":"Obtain a list of companies","serviceProvider":"IBM_ZOS_CONNECT_SERVICE_REST_CLIENT-1.0","serviceInvokeURL":"https://wg31.washington.ibm.com:9453/zosConnect/services/getCompany","serviceStatus":"Started"},"getCompany":{"receiveTimeout":60000,"port":"29080","host":"wg31.washington.ibm.com","httpMethod":"POST","connectionTimeout":30000,"uri":"/trader/rest/GetCompany"}}
```

**Tech-Tip:** If a REST client tool like cURL or Postman was not available then the SAR file could have been deployed using FTP to upload the file in binary mode to the *services* directory.

**Tech-Tip:** If a service needs to be redeployed the service must be first stopped and then deleted. The cURL command with a PUT method can be used to stop the service:

```
curl -X PUT --user Fred:fredpwd --insecure https://wg31.washington.ibm.com:9453/zosConnect/services/getCompany?status=stopped
```

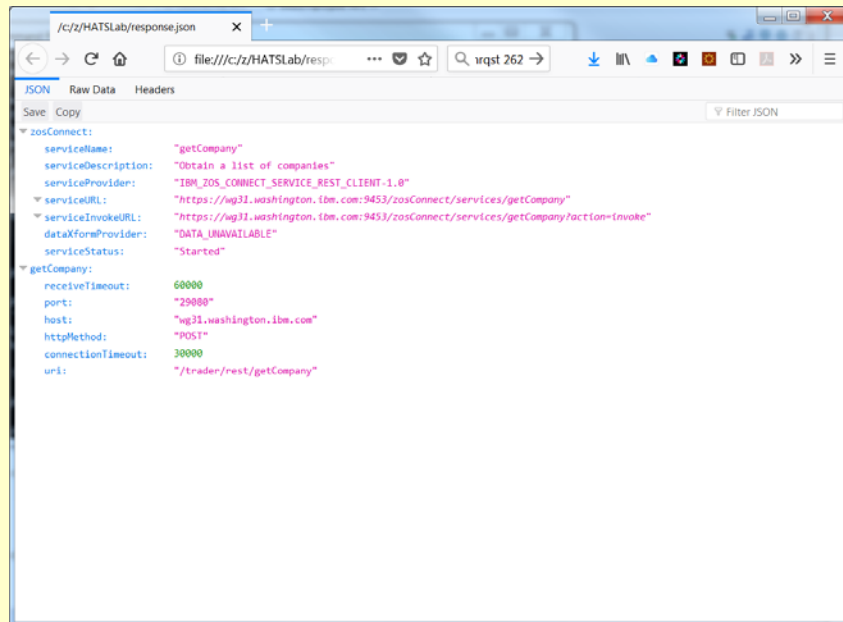
And the cURL command with a DELETE method can be used to delete the service:

```
curl -X DELETE --user Fred:fredpwd --insecure https://wg31.washington.ibm.com:9453/zosConnect/services/getCompany
```

If this is not done the service cannot be redeployed.

**Tech-Tip:** Another useful cURL directive is `-o response.json`

When this directive is used the JSON response message is written to a file named `response.json` which then can be opened with Firefox and viewed in a more readable format, e.g. command `firefox response.json`



3. Deploy the `sellShares` service by using the z/OS Connect EE RESTful administrative interface to deploy the service archive file by using the cURL command with a POST method, e.g.

```
curl -X POST --user Fred:fredpwd --data-binary @sellShares.sar --header "Content-Type: application/zip" --insecure https://wg31.washington.ibm.com:9453/zosConnect/services
```

```
c:\z\HATSLab>curl -X POST --user Fred:fredpwd --data-binary @sellShares.sar --header "Content-Type: application/zip" --insecure https://wg31.washington.ibm.com:9453/zosConnect/services
{"zosConnect":{"serviceName":"sellShares","serviceDescription":"Sell Shares","serviceProvider":"IBM_ZOS_CONNECT_SERVICE_REST_CLIENT-1.0","serviceURL":"https://wg31.washington.ibm.com:9453/zosConnect/services/sellShares","serviceInvokeURL":"https://wg31.washington.ibm.com:9453/zosConnect/services/sellShares?action=invoke","dataXformProvider":"DATA_UNAVAILABLE","serviceStatus":"Started"},"sellShares":{"receiveTimeout":60000,"port":"29080","host":"wg31.washington.ibm.com","httpMethod":"POST","connectionTimeout":30000,"uri":"/trader/rest/SellShares"}}
```

4. Deploy the *buyShares* service by using the z/OS Connect EE RESTful administrative interface to deploy the service archive file by using the cURL command with a POST method, e.g.

```
curl -X POST --user Fred:fredpwd --data-binary @buyShares.sar --header "Content-Type: application/zip" --insecure https://wg31.washington.ibm.com:9453/zosConnect/services
```

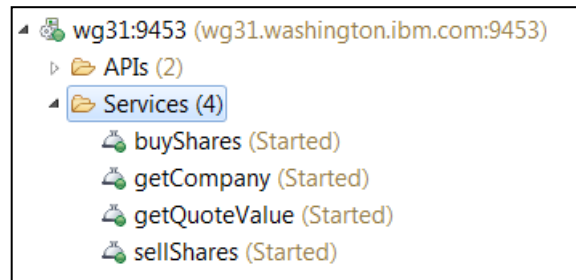
```
c:\z\HATSLab>curl -X POST --user Fred:fredpwd --data-binary @buyShares.sar --header "Content-Type: application/zip" --insecure https://wg31.washington.ibm.com:9453/zosConnect/services
{"zosConnect":{"serviceName":"buyShares","serviceDescription":"Buy Shares","serviceProvider":"IBM_ZOS_CONNECT_SERVICE_REST_CLIENT-1.0","serviceURL":"https://wg31.washington.ibm.com:9453/zosConnect/services/buyShares","serviceInvokeURL":"https://wg31.washington.ibm.com:9453/zosConnect/services/buyShares?action=invoke","dataXformProvider":"DATA_UNAVAILABLE","serviceStatus":"Started"},"buyShares":{"receiveTimeout":60000,"port":"29080","host":"wg31.washington.ibm.com","httpMethod":"POST","connectionTimeout":30000,"uri":"/trader/rest/BuyShares"}}
```

5. Deploy the *buyShares* service by using the z/OS Connect EE RESTful administrative interface to deploy the service archive file by using the cURL command with a POST method, e.g.

```
curl -X POST --user Fred:fredpwd --data-binary @getQuoteValue.sar --header "Content-Type: application/zip" --insecure https://wg31.washington.ibm.com:9453/zosConnect/services
```

```
c:\z\HATSLab>curl -X POST --user Fred:fredpwd --data-binary @getQuoteValue.sar --header "Content-Type: application/zip" --insecure https://wg31.washington.ibm.com:9453/zosConnect/services
{"zosConnect":{"serviceName":"getQuoteValue","serviceDescription":"Get Share quote and values","serviceProvider":"IBM_ZOS_CONNECT_SERVICE_REST_CLIENT-1.0","serviceURL":"https://wg31.washington.ibm.com:9453/zosConnect/services/getQuoteValue","serviceInvokeURL":"https://wg31.washington.ibm.com:9453/zosConnect/services/getQuoteValue?action=invoke","dataXformProvider":"DATA_UNAVAILABLE","serviceStatus":"Started"},"getQuoteValue":{"receiveTimeout":60000,"port":"29080","host":"wg31.washington.ibm.com","httpMethod":"POST","connectionTimeout":30000,"uri":"/trade/r/rest/GetQuoteValue"}}
```

6. Expanding the Services folder in the IBM z/OS Explorer shows the 4 HATS services have been installed and started.



## Summary

The Swagger document generated by HATS has been used to define the request and response JSON schema for each of the services. These JSON schema files along with the properties provided in each service's properties files have been used by the z/OS Connect EE build toolkit to generate service archive files for each service

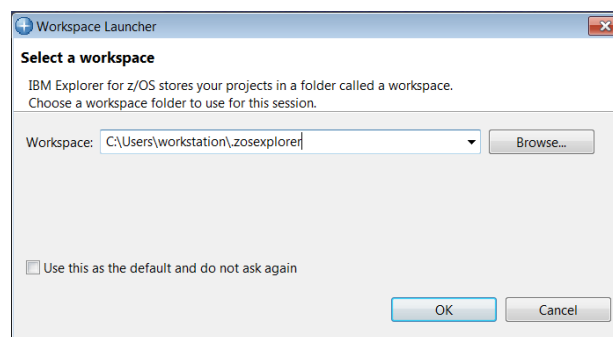
## Connect to the IBM z/OS Explorer to the z/OS Connect EE Server

Now the API Editor will be used to build and test the API using the service archives just generated. If you have previously connected to the z/OS Connect EE server in another exercise then you can proceed to *Create the HATS API project* on page 18.

1. On the workstation desktop, locate the *z/OS Explorer* icon and double click on it to open the tool. If the z/OS Explorer tool is already started and you have a connection to the z/OS Connect sever *wg31:9453* server go to section *Create the HATS API project* on page 18.

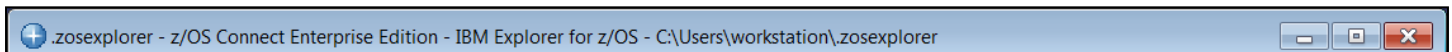
**Tech-Tip:** Windows desktop tools can be opened either by double clicking the icon or by selecting the icon and right mouse button clicking and then selecting the *Open* option.

2. You will be prompted for a workspace:



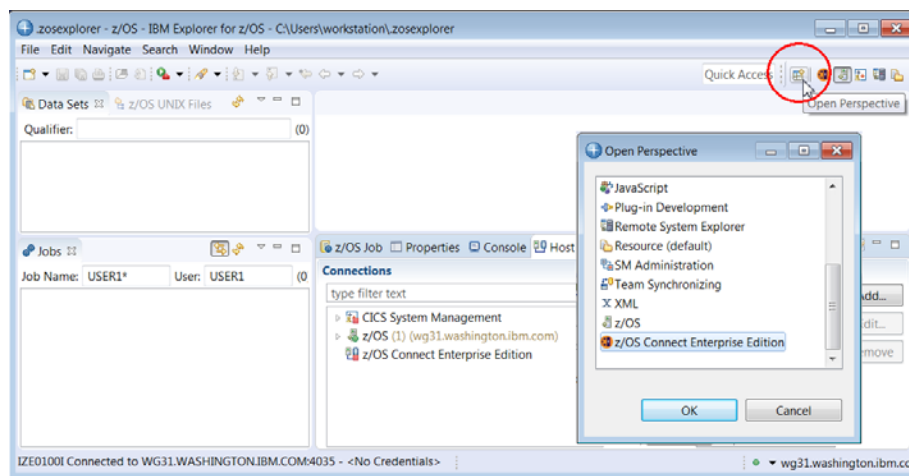
Take the default value by clicking **OK**.

3. The Explorer should open in the *z/OS Connect Enterprise Edition* perspective. Verify this by looking in the upper left corner. You should see:

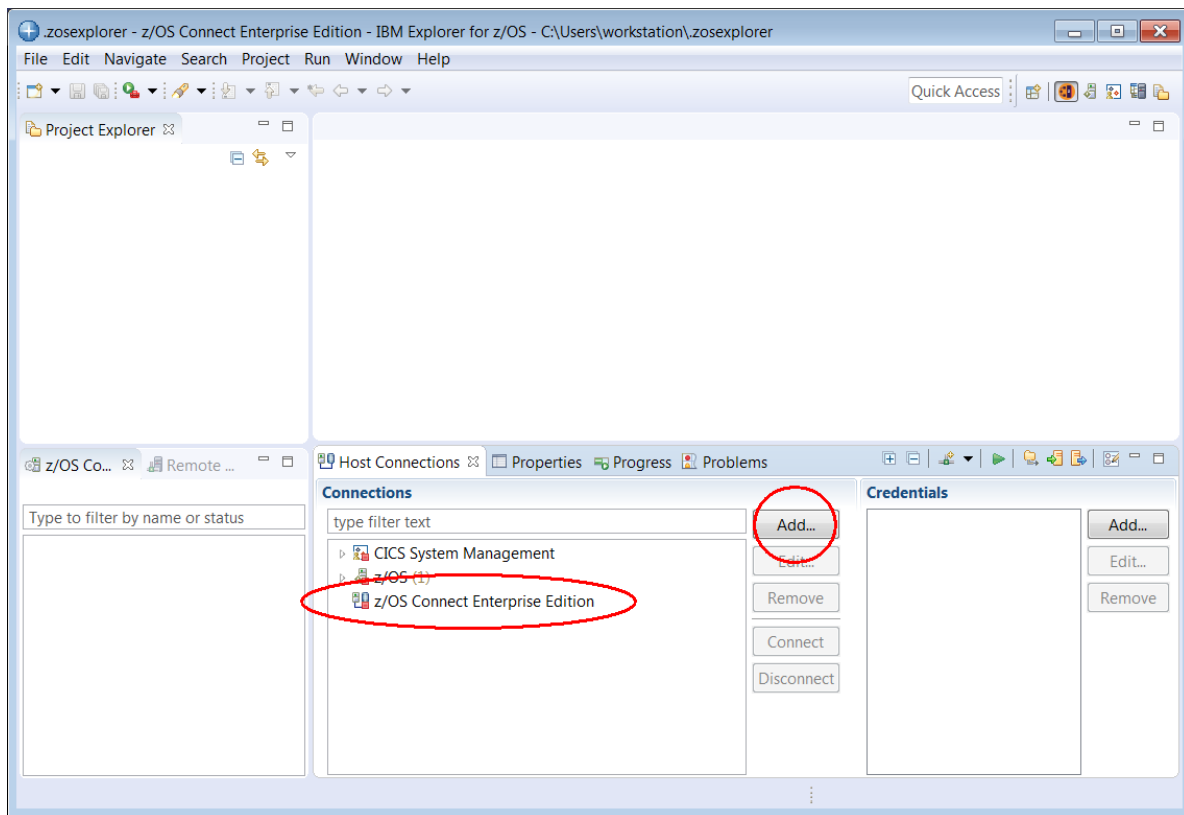


N.B. If a *Welcome* screen is displayed then click the white X beside *Welcome* to close this view.

4. If the current perspective is not *z/OS Connect Enterprise Edition*, select the *Open Perspective* icon on the top right side to display the list of available perspectives, see below. Select **z/OS Connect Enterprise Edition** and click the **OK** button to switch to this perspective.



5. To add a connection to the z/OS Connect Server select *z/OS Connect Enterprise Edition* connection in the *Host connections* tab in the lower view and then click the **Add** button.



**Tech-Tip:** Eclipse based development tools like z/OS Explorer; provide a graphical interface consisting of multiple views within a single window.

A view is an area in the window dedicated to providing a specific tool or function. For example, in the window above, *Host Connections* and *Project Explorer* are views that use different areas of the window for displaying information. At bottom on the right there is a single area for displaying the contents of four views stacked together (commonly called a *stacked views*), *z/OS Host Connections*, *Properties*, *Progress* and *Problems*. In a stacked view, the contents of each view can be displayed by clicking on the view tab (the name of the view).

At any time, a specific view can be enlarged to fill the entire window by double clicking in the view's title bar. Double clicking in the view's title bar will be restored the original arrangement. If a z/OS Explorer view is closed or otherwise disappears, the original arrangement can be restored by selecting **Windows → Reset Perspective** in the window's tool bar.

Eclipse based tools also can display multiple views based on the current role of the user. In this context, a window is known as a perspective. The contents (or views) of a perspective are based on the role the user, i.e., developer or administrator.



6. In the pop-up list displayed select *z/OS Connect Enterprise Edition* and on the *Add z/OS Connect Enterprise Edition Connection* screen enter **wg31.washington.ibm.com** for the *Host name*, **9453** for the *Port Number*, check the box for *Secure connection (TLS/SSL)* and then click the **Save and Connect** button.

7. On the *z/OS Connect Enterprise Edition – User ID* required screen create new credentials for a *User ID* of **Fred** and a *Password or Passphrase* of **fredpwd** (case matters). Remember the server is configured to use basic security. If SAF security had been enabled then a valid RACF User ID and password will have to be used instead. Click **OK** to continue.
8. Click the **Accept** button on the *Server certificate alert – Accept this certificate* screen. You may be presented with another prompt for a userid and password, enter **Fred** and **fredpwd** again.

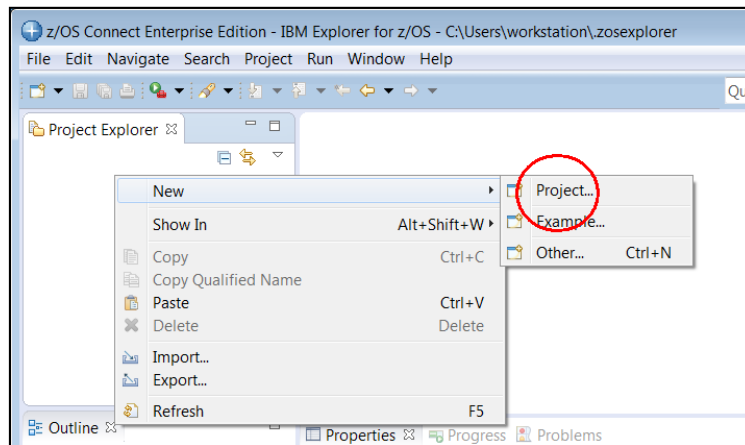
9. The status icon beside **wg31:9453** should now be a green circle with a lock. This shows that a secure connection has been established between the z/OS Explorer and the z/OS Connect server. A red box indicates that no connection exists.
10. A connection to the remote z/OS system was previously added. In the *Host Connection* view expand *z/OS Remote System* under *z/OS* and select **wg31.washington.ibm.com**. If the connection is not active the **Connect** button will be enabled. Click the **Connect** button and this will establish a session to the z/OS system. This step is required when submitting job for execution and viewing the output of these jobs later in this exercise.

## Summary

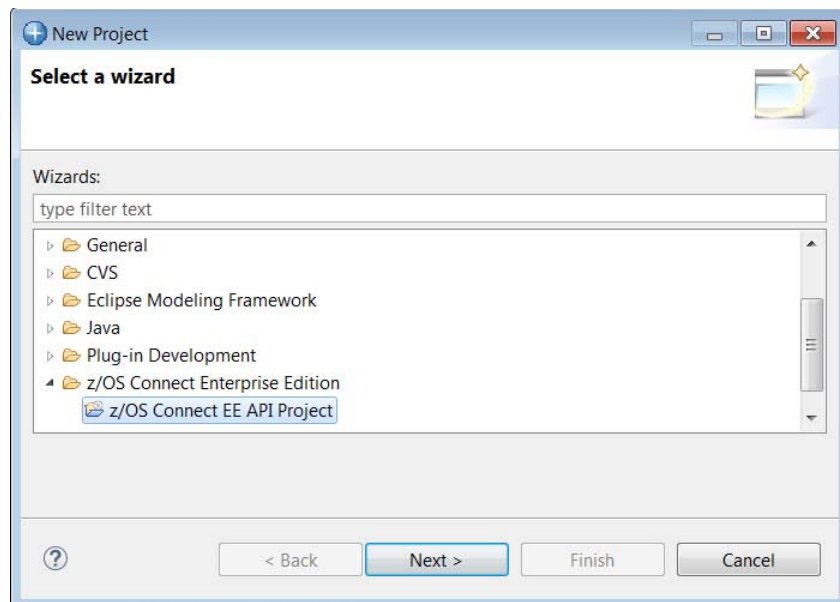
The essential service definitions of z/OS Connect EE V3.0 are in place. The next step is the composing and deployment of the API and then the testing of the API functions.

## Create the HATS API project

1. In the *z/OS Connect Enterprise Edition* perspective of the z/OS Explorer create a new API project by clicking the right mouse button and selecting **New** → **Project**:



2. In the *New Project* screen, scroll down and open the *z/OS Connect Enterprise Edition* folder and select *z/OS Connect EE API Project* and then click the **Next** button.



3. Enter **HATSAPI** for the *Project name*. Be sure the *API name* is set to **Trader** and the *Base path* is set to **/trader**. Click **Finish** to continue.

The screenshot shows the 'New Project' dialog box. The title bar says 'New Project'. The main title is 'z/OS Connect EE API Project'. Below it is the instruction 'Create a new z/OS Connect EE API project.' The form has four sections: 'Project name:' with a text box containing 'HATSAPI'; 'API name:' with a text box containing 'Trader'; 'Base path:' with a text box containing '/trader'; and 'Description:' with a large empty text area. At the bottom right are 'Finish' and 'Cancel' buttons. A help icon (?) is at the bottom left.

**Important:** The values are somewhat arbitrary, but they do relate to later tasks. If you use the values and cases as supplied, then the subsequent commands and the use of subsequent URLs will work seamlessly.

4. You should now see something like the view below. The view may need to be adjusted by dragging the view boundary lines.

The screenshot shows the 'z/OS Connect EE API Editor' window. The title bar says 'Trader API'. The main title is 'z/OS Connect EE API Editor'. The 'Describe your API' section has three text boxes: 'Name:' (Trader), 'Base path:' (/trader), and 'Version:' (1.0.0). To the right of these is a 'Description:' text area. Below this is a 'Path' section with a text box containing '/newPath1'. The 'Methods' section is expanded, showing four methods: POST, GET, PUT, and DELETE. Each method has a 'Service...' button and a 'Mapping...' button, along with up, down, and delete icons.

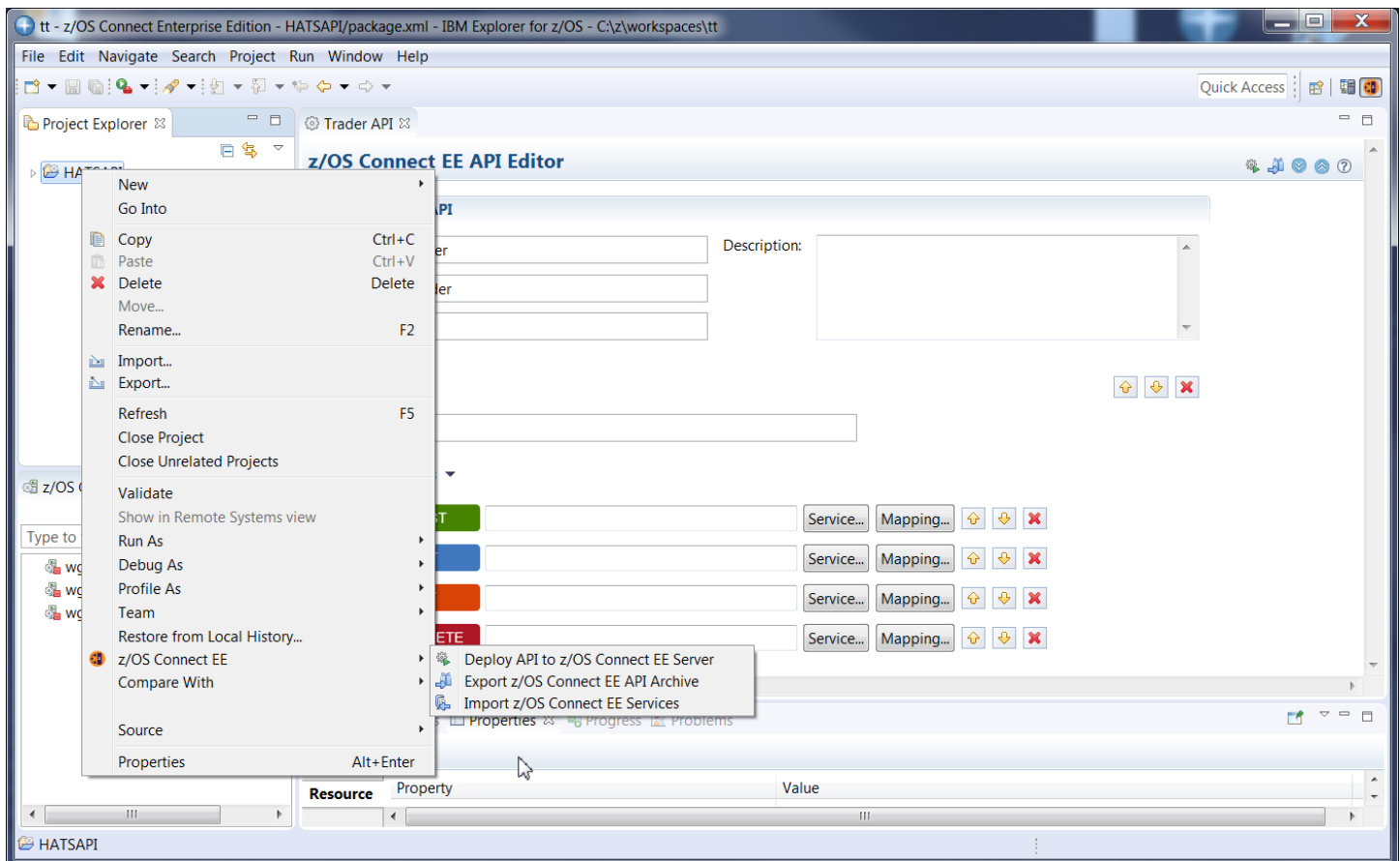
**Tech-Tip:** If the API Editor view is closed, it can be reopened by double clicking the *package.xml* file in the API project.

## Summary

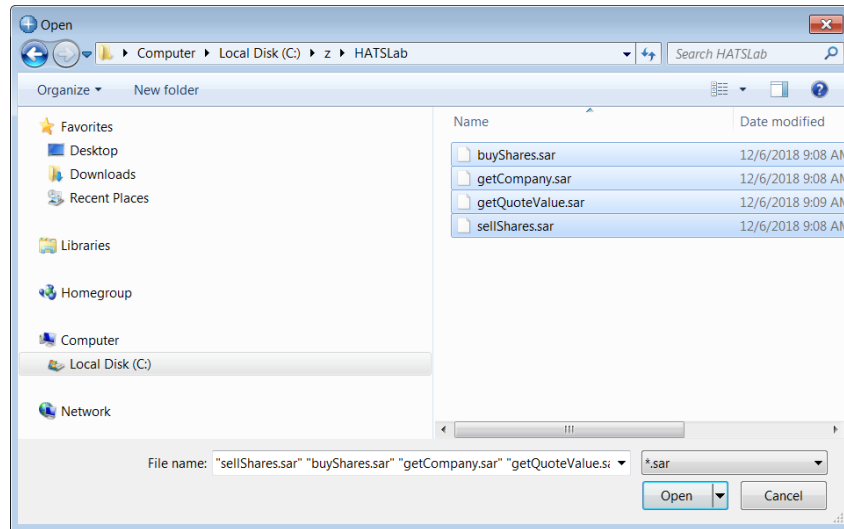
This created the basic framework for the API project in the API editor

### *Import the SAR files into the API Toolkit*

1. In the z/OS Explorer in the *z/OS Connect Enterprise Edition* perspective in the *Project Explorer* view (upper left), right-click on the *HATSPI* project, then select *z/OS Connect EE* and then *Import z/OS Connect EE Services* (see below):

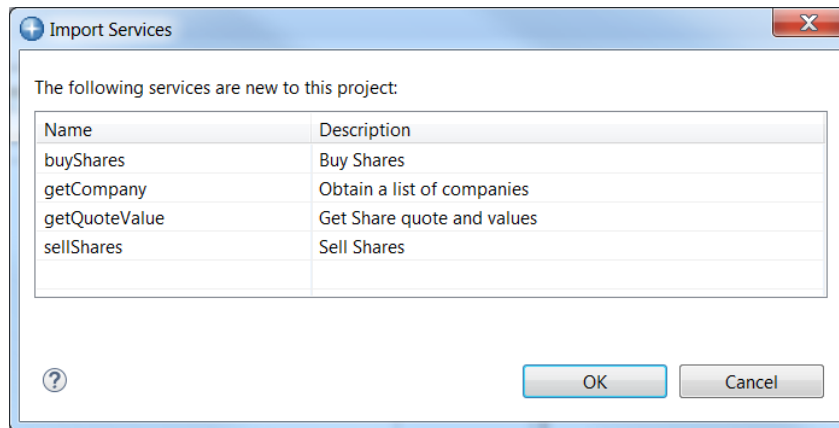


2. In the *Import z/OS Connect EE Services* window click on the **File System** button and navigate to directory *c:\z\HATSLab*. Select the four SAR files and click on the **Open** button:

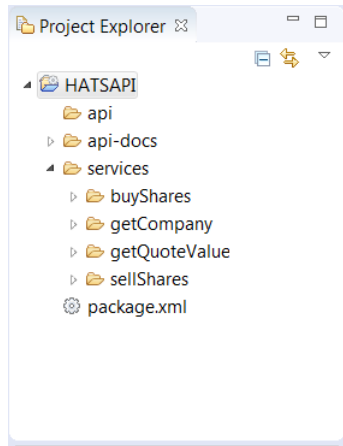


**Tech-Tip:** Multiple items can be selected by holding the **Shift** key down and then selecting the first SAR file in the list and then selecting the last SAR file in the list.

3. The four service archive files should appear in the *Import z/OS Connect EE Services* screen. Click the **OK** button twice to import them into the workspace.



4. In the *Project Explorer* view (upper left), expand the *services* folder to see the imported service:

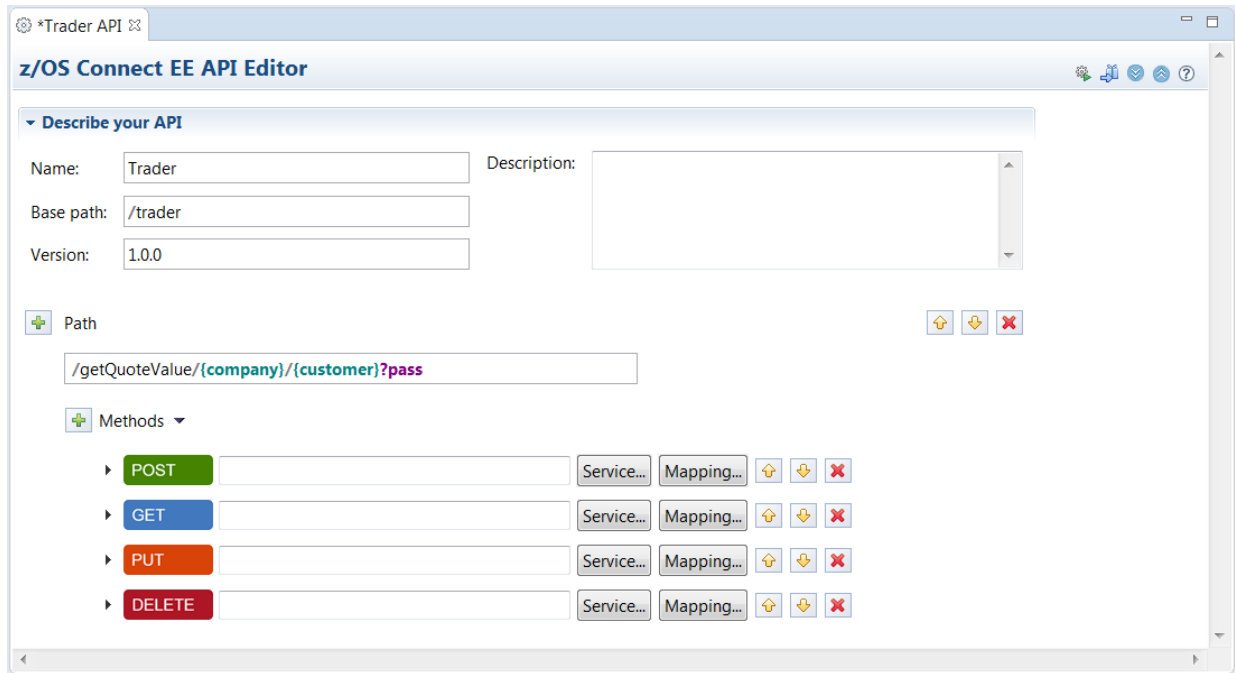


## Summary

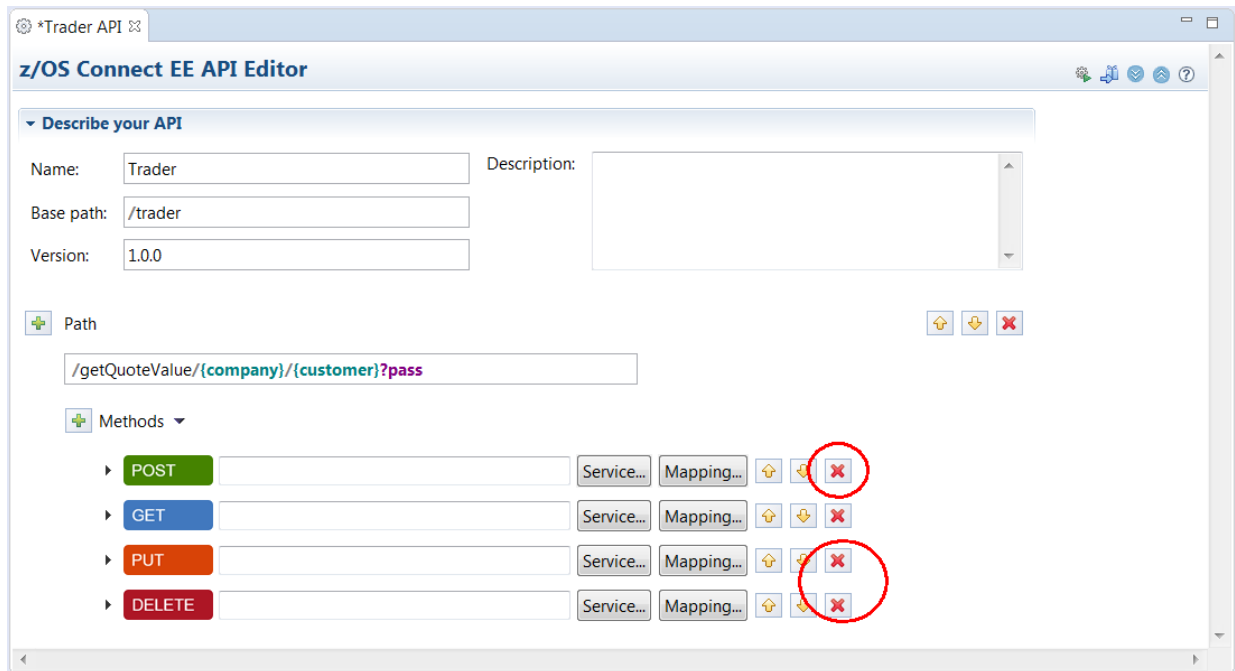
The SAR file created earlier was imported into the editor. That provides the editor with information about the underlying services and the JSON schemas.

## Compose an API for HATS Rest Services

- \_\_\_1. Start by entering a *Path* of `/getQuoteValue/{company}/{customer}?pass` in the *z/OS Connect EE API Editor* view as shown below:



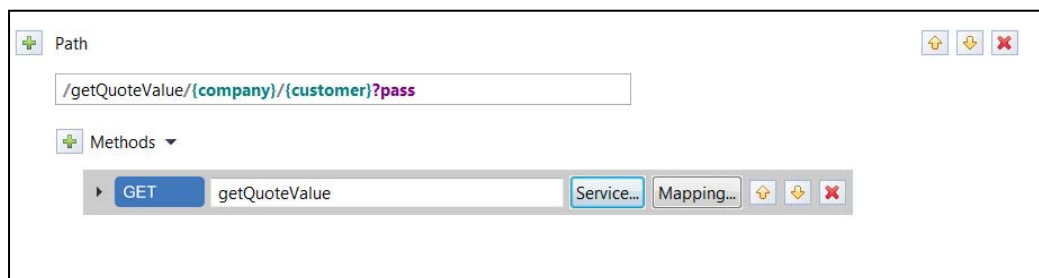
- \_\_\_2. For the *HATS API* for the *getQuoteValue* path is present the only supported HTTP methods will be **GET**. Remove the **DELETE**, **POST** and **PUT** methods by clicking the X icon to the right of each method.



That should leave you with the **GET** method.



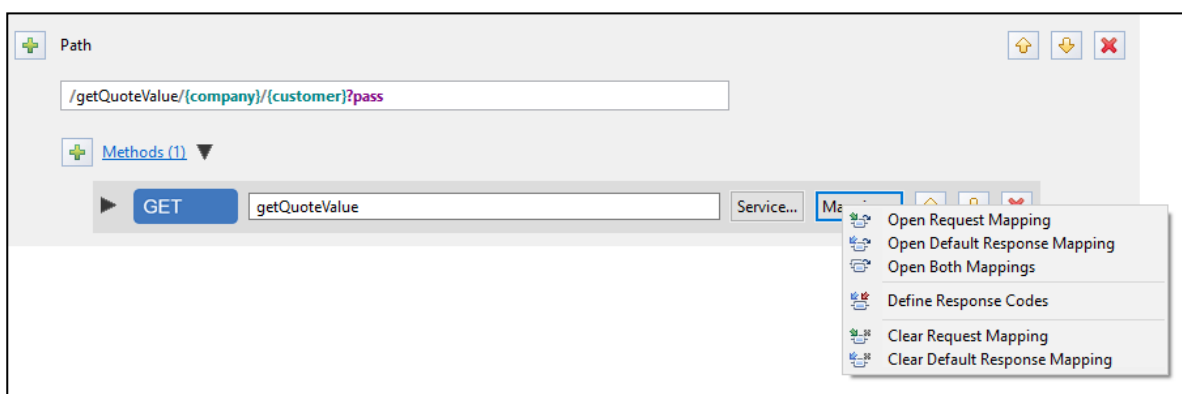
- \_\_\_3. Click on the **Service** button to the right of the **GET** method. Then select the *getQuoteValue* service from the list of service archive files and click **OK**. This will populate the field to the right of the method.



- \_\_\_4. Save the changes so far by using the key sequence **Ctrl-S**.

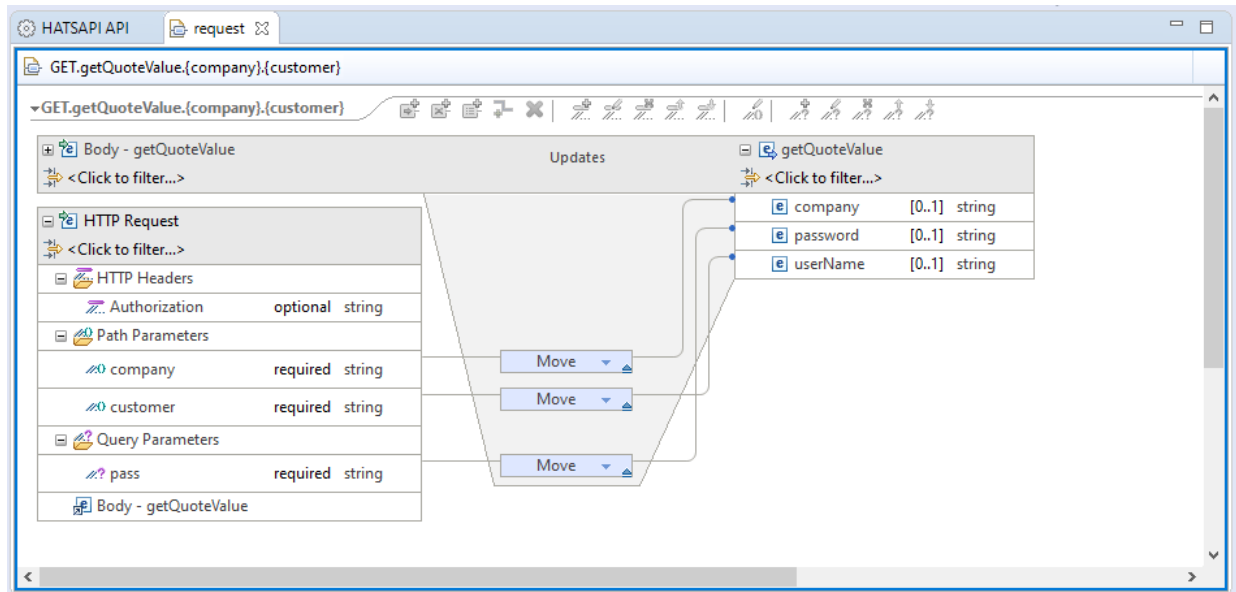
**Tech-Tip:** If any change is made in any edit view an asterisk (\*) will appear before the name of the artifact in the view tab, e.g. *\*package.xml*. Changes can be saved at any time by using the **Ctrl-S** key sequence.

- \_\_\_5. Next, click on the **Mapping** button beside the **GET** method and then select *Open Request Mapping*:



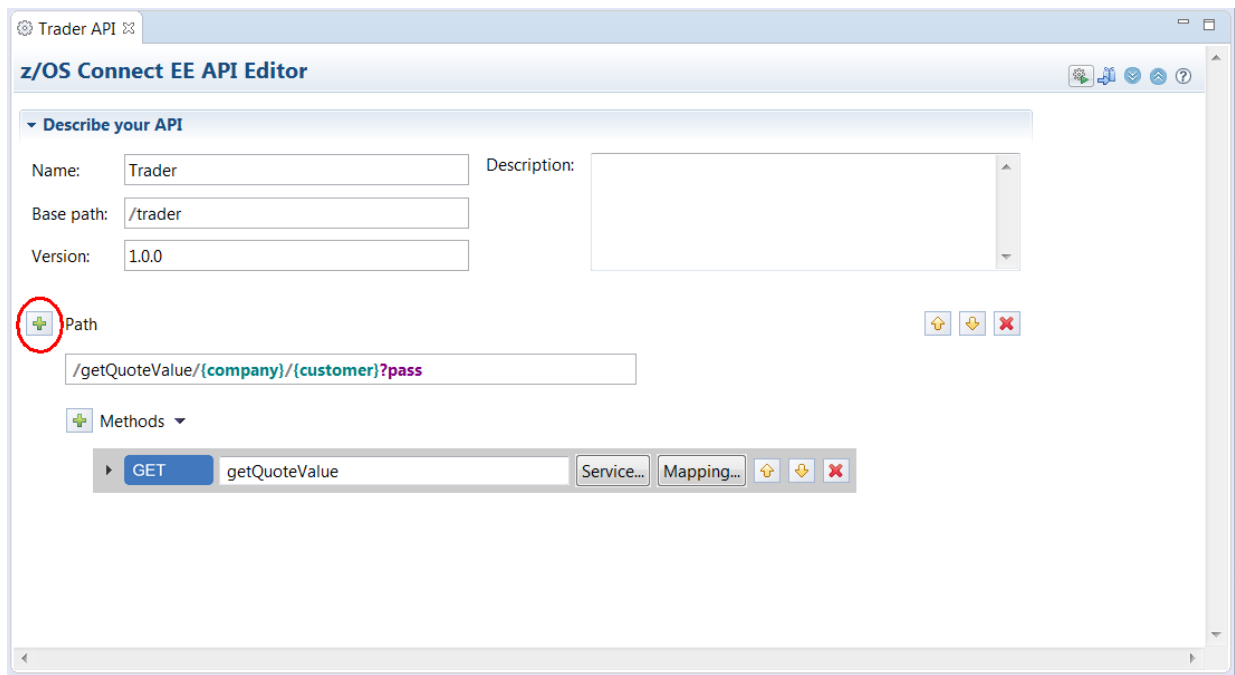


- \_\_\_6. In the mapping view that opens, use your mouse to select the *customer* field under *Path Parameters* and drag it over to the *userid* field on the right hand side. The result is a line that maps a move of the value of *customer* from the URL to the field *userid*. Repeat this mapping to move the query parameter *pass* to the *password* field on the right-hand side and the path parameter *company* to the *company* field on the right-hand side.

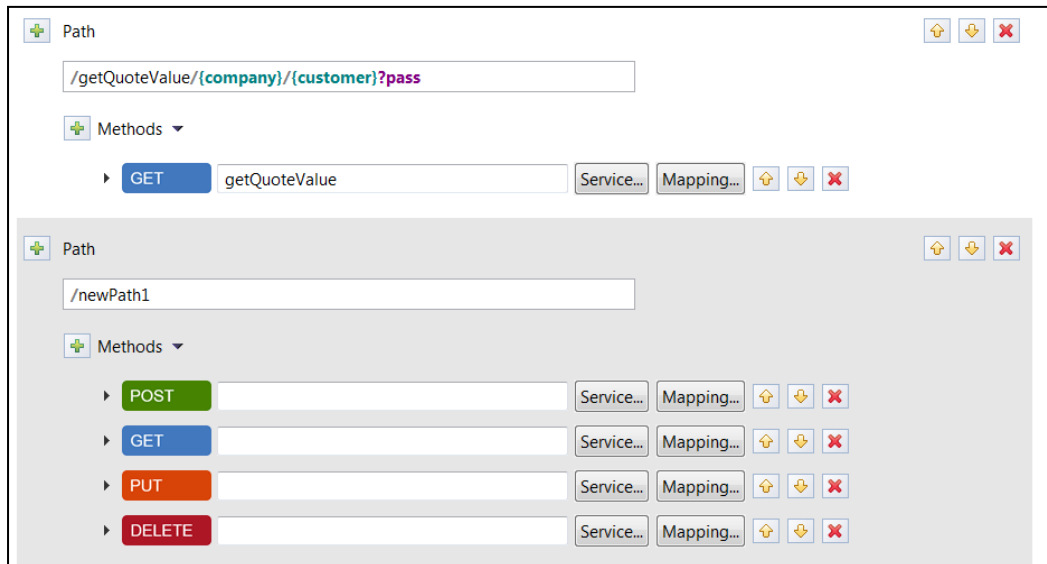


- \_\_\_7. Use the **Ctrl-S** key sequence to save all changes and close the `GET.getQuoteValue.{company}.{customer}` view.
- \_\_\_8. No Response mappings for the **GET** method are required.

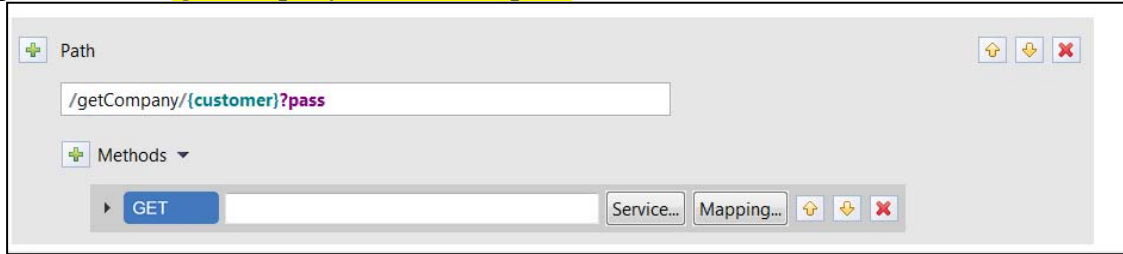
\_\_\_9. Next, we want to add a Path for a **GET** method for the *getCompany* service by clicking the plus sign beside *Path* on the screen.



The result is another full set of methods for the new *PATH*.



\_\_\_10. Enter a path value of **/getCompany/{customer}?pass** and remove the **POST**, **PUT** and **DELETE** methods.



The screenshot shows a configuration window with two main sections: 'Path' and 'Methods'. The 'Path' section has a text input field containing '/getCompany/{customer}?pass'. The 'Methods' section has a dropdown menu with 'GET' selected. To the right of the 'Methods' dropdown are buttons for 'Service...' and 'Mapping...', and three small icons (up, down, and delete).

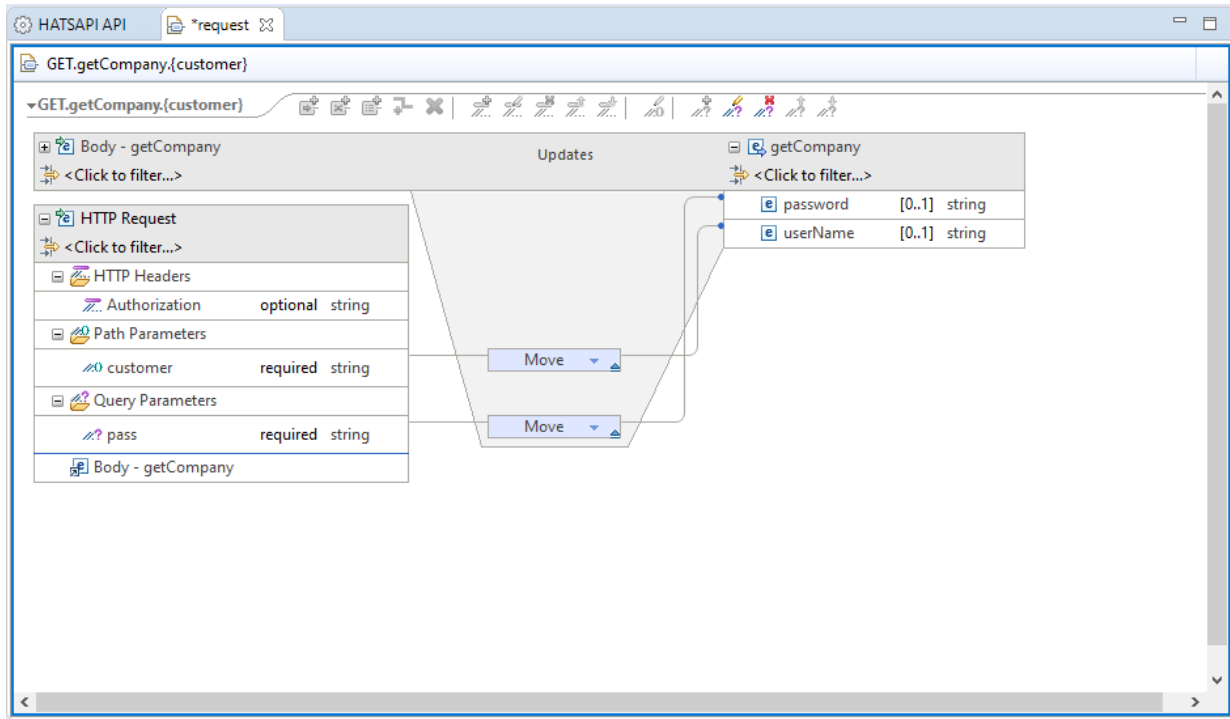
**Tech-Tip:** Additional *Paths* can be added by clicking the + icon beside *Path* and additional *Methods* can be added by clicking the + icon beside *Methods*.

\_\_\_11. Click the **Service** button beside **GET** and select the *getCompany* service:

\_\_\_12. Save the changes by using the key sequence **Ctrl-S**.

\_\_\_13. Click on *Mapping* → *Open request mapping*.

- \_\_\_14. Use the left mouse button to drag the *customer* path parameter from the left-hand side to the *userid* field on the right side. Use the left mouse button to drag the *pass* query parameter from the left-hand side to the *password* field on the right-hand side.

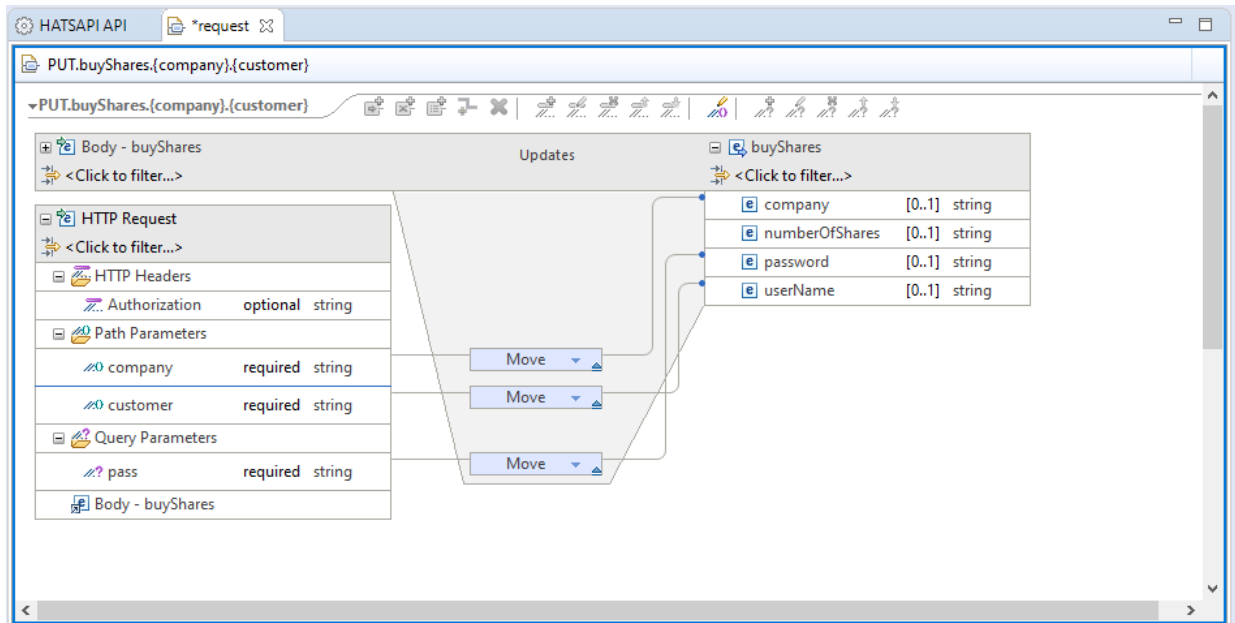


- \_\_\_15. Save the the changes using the key sequence **Ctrl-S**.
- \_\_\_16. Close any open *request* mapping tab.
- \_\_\_17. Next, we want to add a Path for a **PUT** method for the *buyShares* service by clicking the plus sign again.
- \_\_\_18. Enter a path value of **/buyShares/{company}/{customer}?pass** and remove the **POST**, **GET** and **DELETE** methods.
- \_\_\_19. Click the **Service** button beside **PUT** and select the *buyShares* service:



- \_\_\_20. Save the changes by using the key sequence **Ctrl-S**.

- \_\_\_21. Click on *Mapping* → *Open request mapping*.
- \_\_\_22. Use the left mouse button to drag the *customer* and *company* path parameters from the left-hand side to the *userid* and *company* fields on the right side. Use the left mouse button to drag the *pass* query parameter from the left-hand side to the *password* field on the right-hand side.



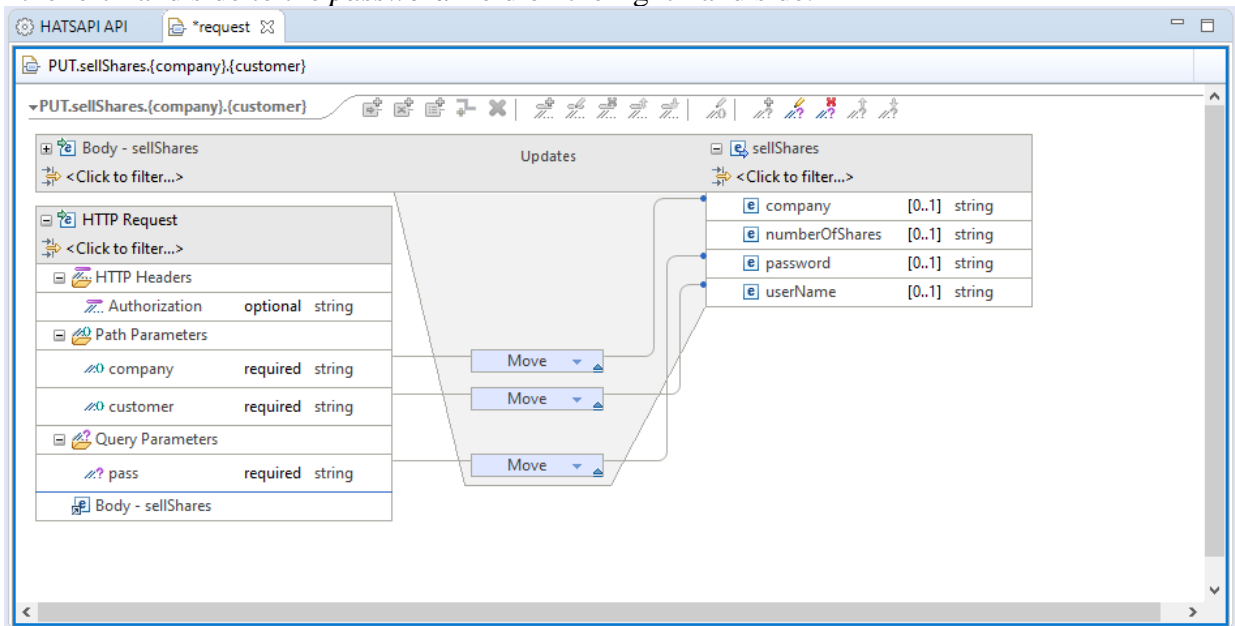
- The value for the number of shares to buy will be sent in a JSON request message.
- \_\_\_23. Save the changes using the key sequence **Ctrl-S**.
- \_\_\_24. Close any open *request* mapping tabs.
- \_\_\_25. Next, we want to add a Path for a **PUT** method for the *sellShares* service by clicking the plus sign again.
- \_\_\_26. Enter a path value of **/sellShares/{company}/{customer}?pass** and remove the **POST**, **GET** and **DELETE** methods.
- \_\_\_27. Click the **Service** button beside **PUT** and select the *sellShares* service:



- \_\_\_28. Save the changes by using the key sequence **Ctrl-S**.

\_\_\_29. Click on *Mapping* → *Open request mapping*.

\_\_\_30. Use the left mouse button to drag the *customer* and *company* path parameters from the left-hand side to the *userid* and *company* fields on the right side. Use the left mouse button to drag the *pass* query parameter from the left-hand side to the *password* field on the right-hand side.



The number of shares to sell will be sent in a JSON request message.

\_\_\_31. Save the the changes using the key sequence **Ctrl-S**.

\_\_\_32. Close any open *request* mapping tabs.

## Summary

You created the API, which consists of four paths and the request and response mapping associated with each. That API will now be deployed into z/OS Connect EE V3.0.

## Deploy the API to a z/OS Connect EE Server

Before deploying the API review the configuration required to support this API.

1. The four services were defined by the inclusion of file *hats.xml* in the *server.xml* we saw earlier.

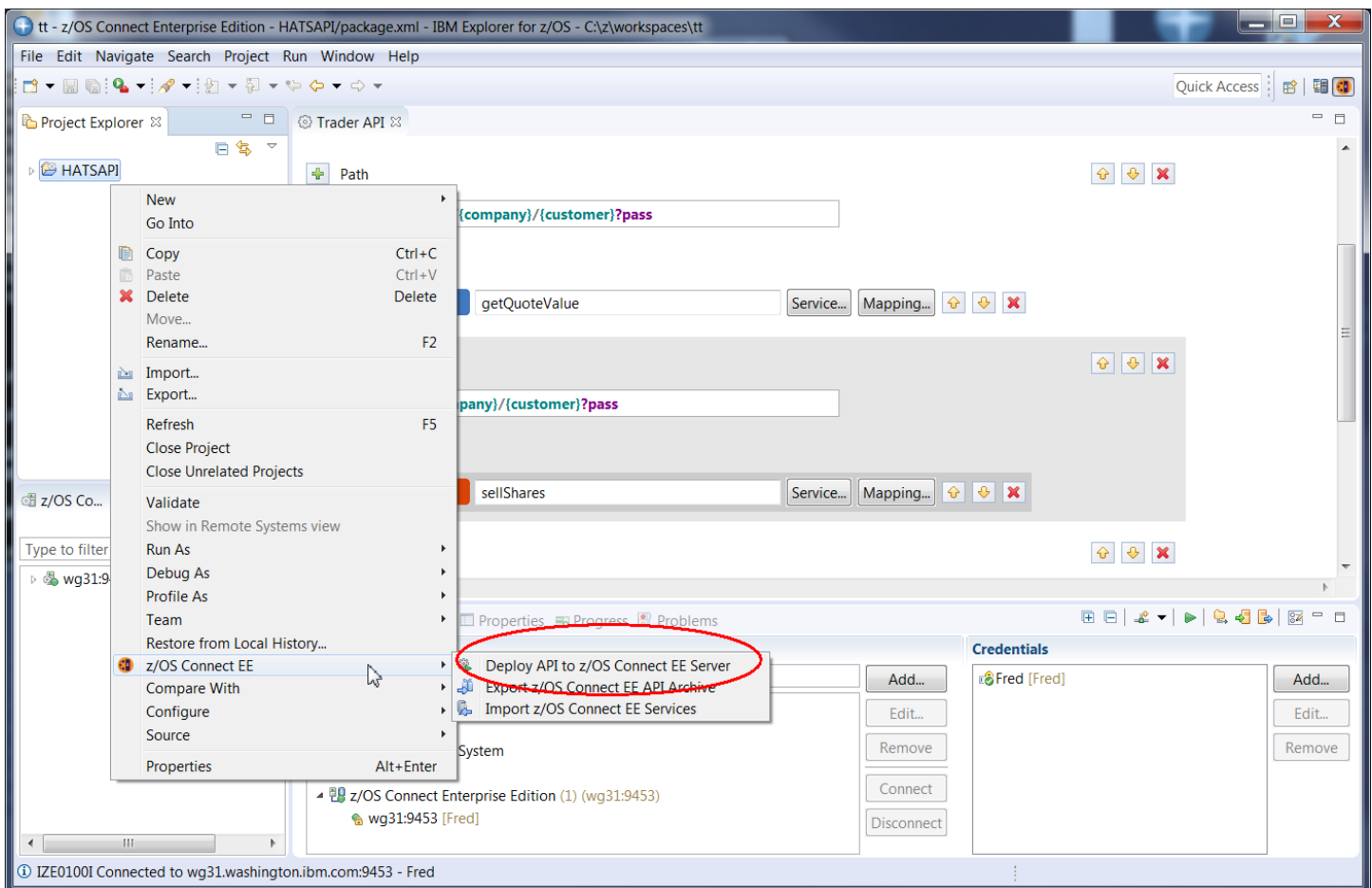
```
<server description="HATS">

  <zoscconnect_zosConnectServiceRestClientConnection id="HatsConn"
    host="wg31.washington.ibm.com"
    port="29080" />

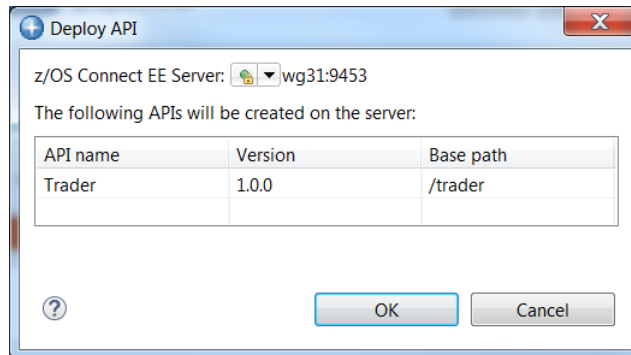
</server>
```

- The `zosConnectServiceRestClientConnection` element identifies the host on which the HATS server resides and the port used for HTTP connections.

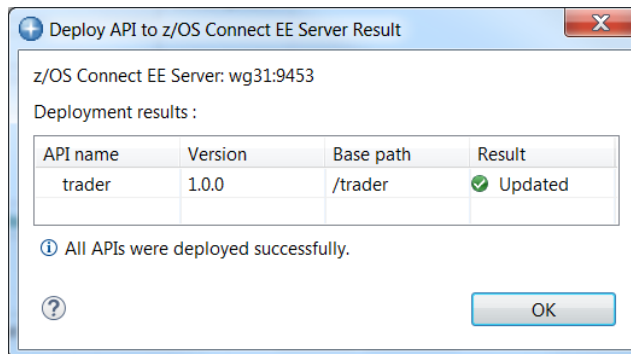
2. In the *Project Explorer* view (upper left), right-mouse click on the *HATSAPI* folder, then select *z/OS Connect EE* → *Deploy API to z/OS Connect EE Server*.



- \_\_\_3. If the z/OS Explorer is connected to only one z/OS Connect server there is only one choice (*wg31:9453*). If z/OS Explorer had multiple connections to z/OS Connect servers then the pull down arrow would allow a selection to which server to deploy, select *wg31:9453* from the list. Click **OK** on this screen to continue.



- \_\_\_4. The API artifacts will be transferred to z/OS in an API archive (AAR) file and copied into the */var/ats/zosconnect/servers/se/resources/zosconnect/apis* directory.





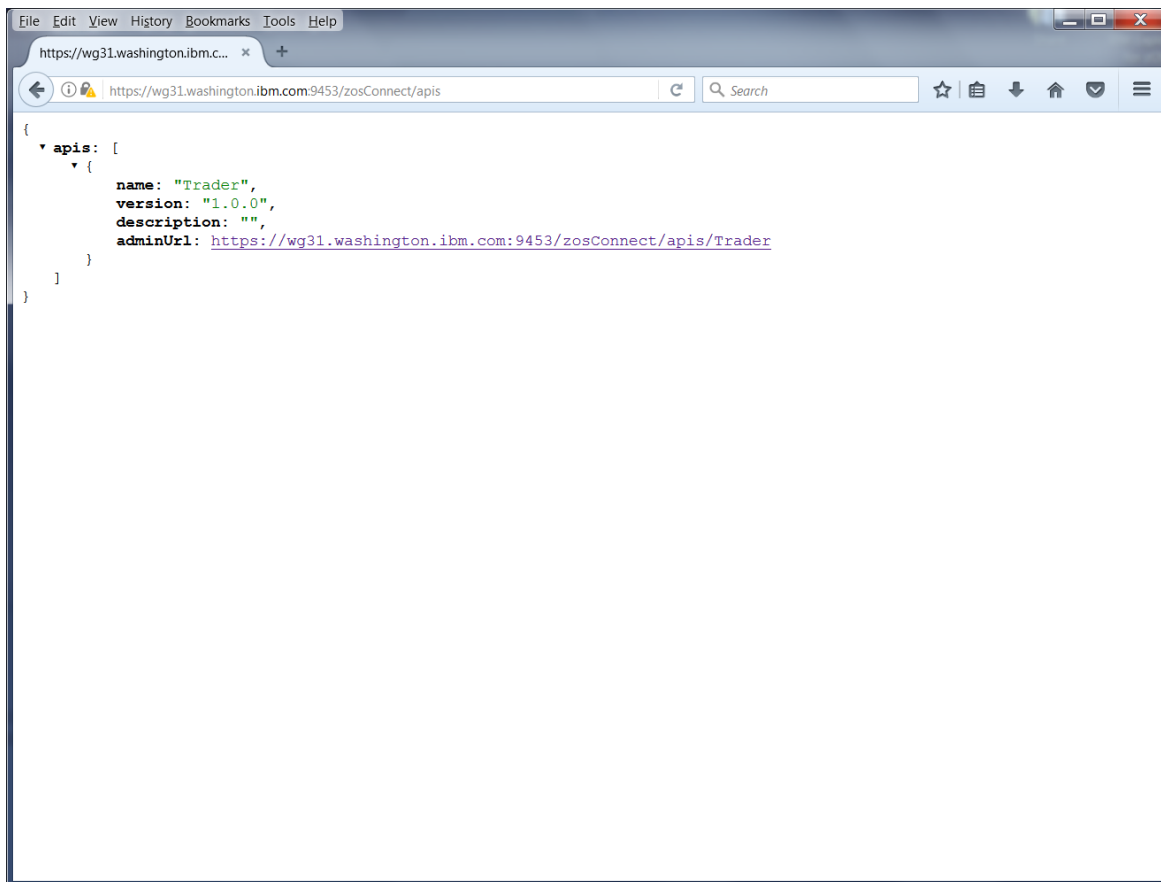
## Test the HATS APIs

1. Start a Firefox web browser session and enter the following as the URL:

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

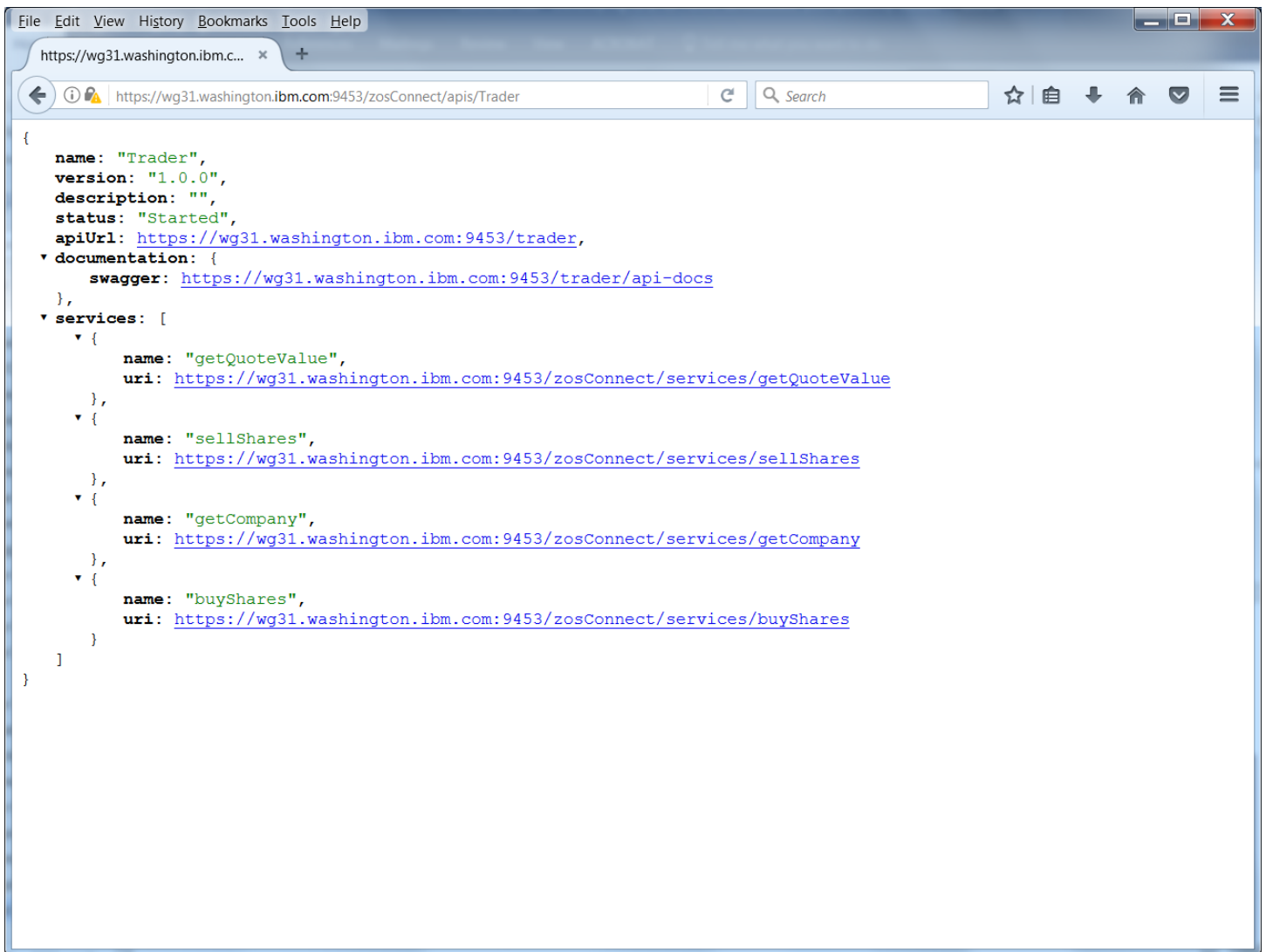
**Tech Tip:** You may be challenged by Firefox because the digital certificate used by the Liberty z/OS server is self-signed. Click the **Advanced** button to continue. Scroll down and then click on the **Accept the Risk and Continue** button. Next you may see a prompt you for a userid and password. If you do see the prompt, enter the username **Fred** and password **fredpwd** (case matters) and click **OK**. Remember we are using basic security, and this is the user identity and password defined in the server.xml file.

2. You should now see the following. The *Trader* API (as well as other) should now show as being available.



**Tech Tip:** It is very important to access the z/OS Connect server from a browser prior to any testing using the Swagger UI. Accessing a z/OS Connect URL from a browser starts an SSL handshake between the browser and the server. If this handshake has not performed prior to performing any test the test will fail with no message in the browser and no explanation. Ensuring this handshake has been performed is why you may be directed to access a z/OS Connect URL prior to using the Swagger UI during this exercise.

3. If you click on *adminUrl* URL the window below should be displayed:



Finally click on the *swagger* URL and you should see the Swagger document associated with this API.

```

{
  swagger: "2.0",
  info: {
    description: "",
    version: "1.0.0",
    title: "Trader"
  },
  basePath: "/trader",
  schemes: [
    "https",
    "http"
  ],
  consumes: [
    "application/json"
  ],
  produces: [
    "application/json"
  ],
  paths: {
    "/buyShares/{company}/{customer}": {
      put: {
        operationId: "putBuyShares",
        parameters: [
          {
            in: "body",
            name: "putBuyShares_request",
            description: "request body",
            required: true,
            schema: {
              $ref: "#/definitions/putBuyShares_request"
            }
          },
          {
            name: "company",
            in: "path",
            required: true,
            type: "string"
          },
          {
            name: "customer",
            in: "path",

```

Explore this Swagger document and you will see the results of the request and response mapping performed earlier. This Swagger document can be used by a developer or other tooling to develop REST clients for this specific API.

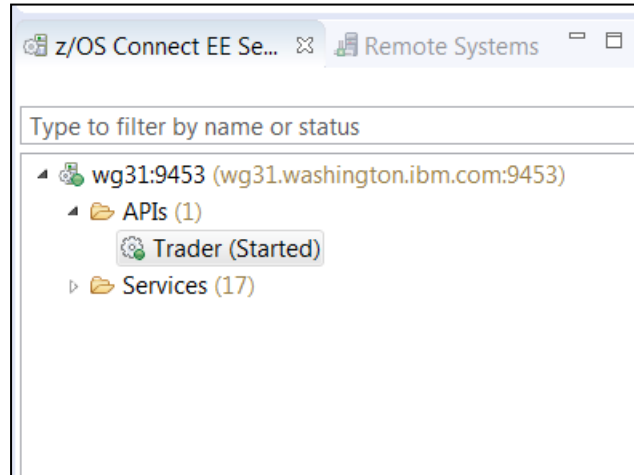
**Important:** Leave your browser open and at this screen. This will preserve the certificate challenge state and the authentication state. You'll need that later in this exercise.

## Summary

The z/OS Connect configuration was reviewed and basic operations of the server were verified with the browser.

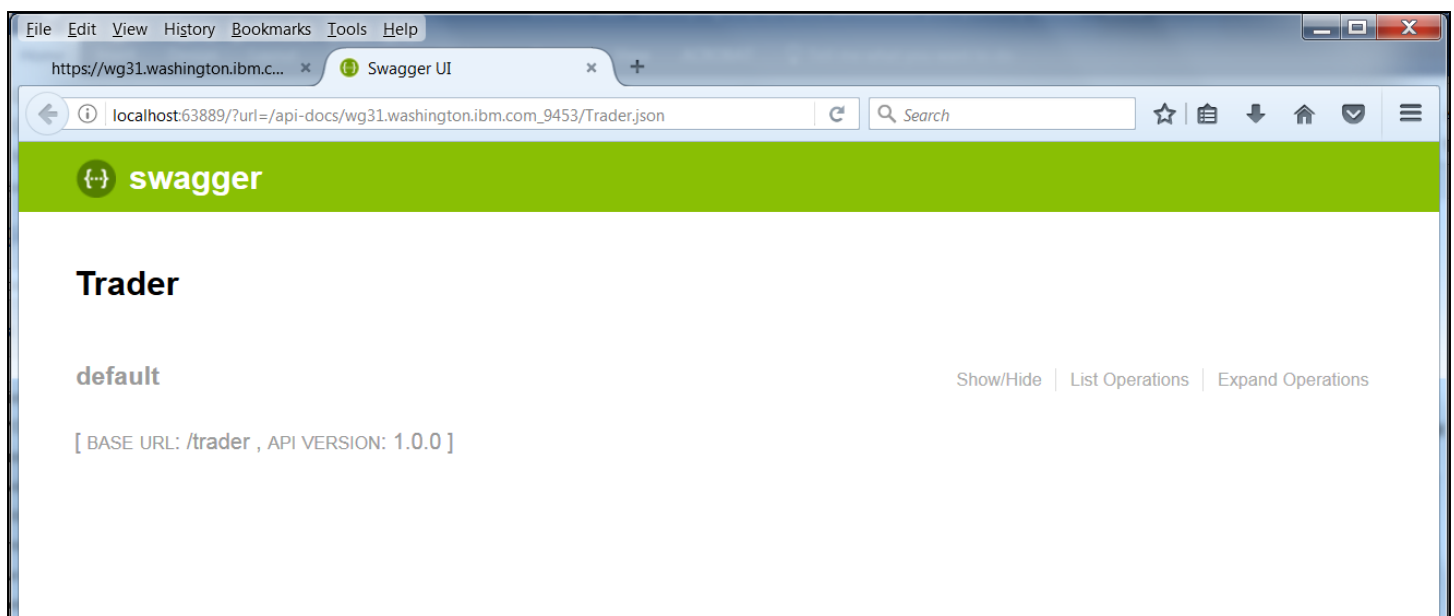
## Test the API using Swagger UI

1. In the lower left-hand side of the *z/OS Connect Explorer* perspective there is view entitled *z/OS Connect EE Servers*. Expand *wg31:9453* and then expand the *APIs* folder. You should see a list of the APIs installed in the server.

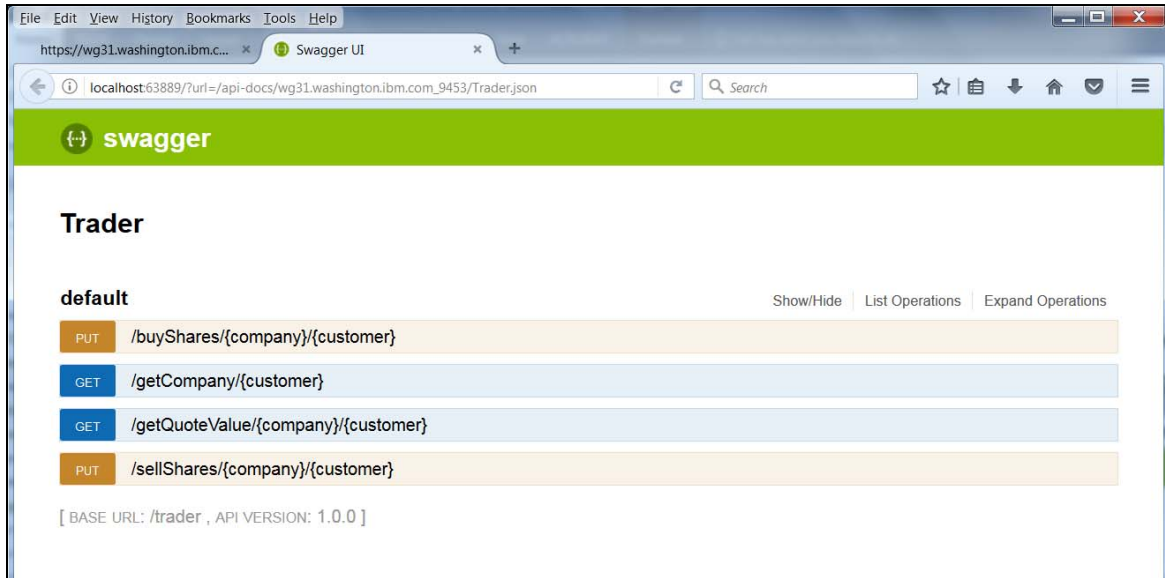


You may see a Security Alert pop-up warning about the self-signed certificate being used by the z/OS Connect EE server. Click **Yes** on this pop-up.

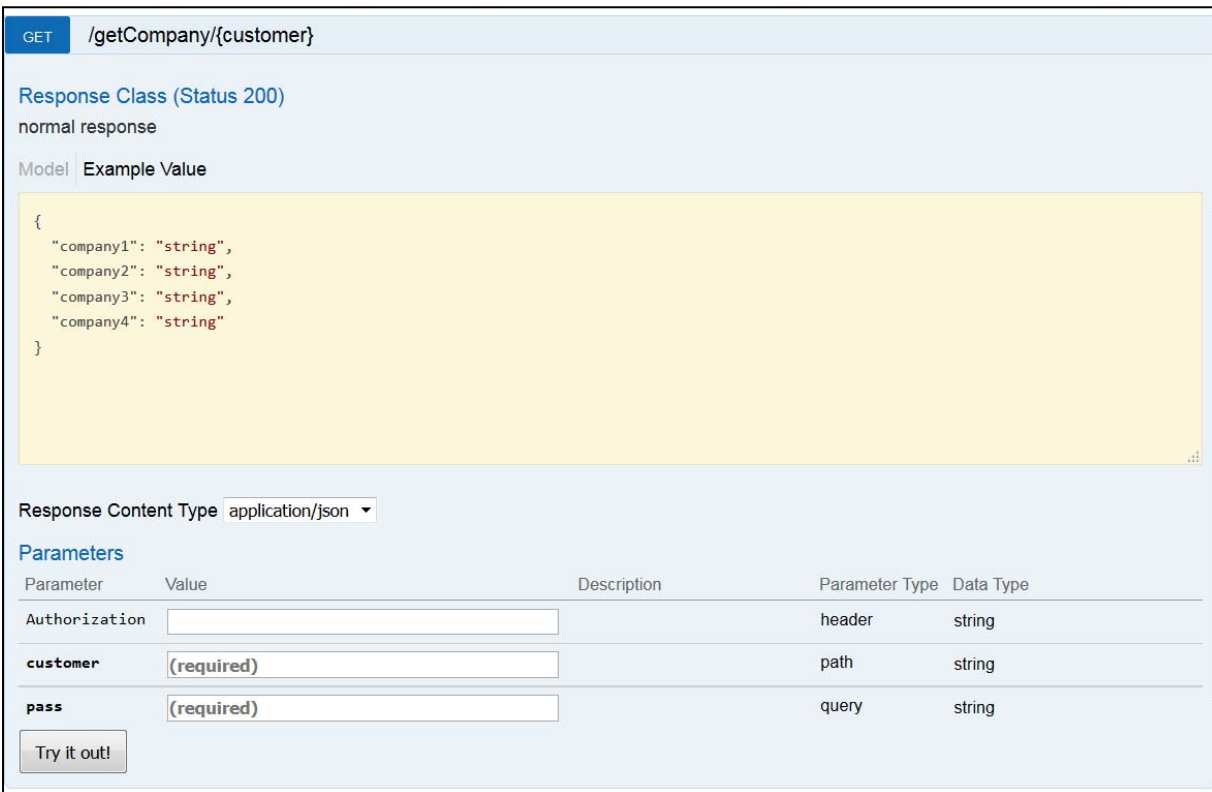
2. Right mouse button click on *Trader* and select *Open in Swagger UI*. Click **OK** if an informational prompt appears. This will open a new view showing a Swagger test client (see below).



3. Click on *List Operations* option in this view and this will display a list of available HTTP methods in this API.



4. Select the *GET* method for displaying the list of companies by clicking on the */getCompany/{customer}* URI string. Remember this was the *Path* specified for the *GET* method for the *getCompany* service when the API was defined. This action will expand this method in this view and provides a Swagger UI test client (you may have to use the slider bar and adjust the perspective to see the entire client).



- \_\_\_5. Enter **USER1** in the boxes beside customer and pass. Enter **Basic RnJlZDpmcmVkcHdk** in the box beside Authorization. Press the **Try it out!** button.
- \_\_\_6. Scroll down the view and you should see the *Response Body* which contains the results of the GET method (see below).

**Response Body**

```
{
  "company1": "Casey_Import_Export",
  "company2": "Glass_and_Luget_Plc",
  "company3": "Headworth_Electrical",
  "company4": "IBM"
}
```

**Response Code**

200

**Response Headers**

```
{
  "content-type": "application/json",
  "content-language": "en-US",
  "expires": "Thu, 01 Dec 1994 16:00:00 GMT",
  "cache-control": "no-cache=\set-cookie, set-cookie2\""
}
```

7. Select the *PUT* method for buying shares of stock by clicking on the */buyShares/{company}/{number}/{customer}* URI string. Remember this was the *Path* specified for the *PUT* method for the *buyShares* service when the API was defined. This action will expand this method in this view and provides a Swagger UI test client (you may have to use the slider bar and adjust the perspective to see the entire client).

Response Class (Status 200)

OK

Model | Example Value

```
{
  "message": "string"
}
```

Response Content Type application/json

Parameters

Parameter	Value	Description	Parameter Type	Data Type
Authorization			header	string
company	(required)		path	string
customer	(required)		path	string
pass	(required)		query	string
putBuyShares_request	-	request body	body	Model   Example Value

numberOfShares

Parameter content type: application/json

```
{
  "numberOfShares": "string"
}
```

Try it out!

8. Enter **USER1** in the boxes beside customer and pass. Enter **Basic RnJlZDpmcmVkeHdk** in the box beside Authorization and **1** in the boxes beside company and number. Press the **Try it out!** button.

9. Scroll down the view and you should see the *Response Body* which contains the results of the PUT method (see below). The message *Request Completed OK* should be the contents of the response body message. This is the message displayed on the 3270 screen when a buy is successful.

**Request URL**

```
https://wg31.washington.ibm.com:9453/trader/buyShares/1/1/user1?pass=user1
```

**Request Headers**

```
{
  "Accept": "application/json",
  "Authorization": "Basic RnJlZDpmcmVkcHdk"
}
```

**Response Body**

```
{
  "message": "Request Completed OK"
}
```

**Response Code**

```
200
```

**Response Headers**

```
{
  "content-type": "application/json",
  "content-language": "en-US",
  "expires": "Thu, 01 Dec 1994 16:00:00 GMT",
  "cache-control": "no-cache=set-cookie, set-cookie2"
}
```



10. Select the *GET* method for display the real-time quote and values for a customer's holding by clicking on the */getQuoteValue/{company}/{customer}* URI string. Remember this was the *Path* specified for the *GET* method for the *getQuoteValue* service when the API was defined. This action will expand this method in this view and provides a Swagger UI test client (you may have to use the slider bar and adjust the perspective to see the entire client).

GET
/getQuoteValue/{company}/{customer}

Response Class (Status 200)  
normal response

Model
Example Value

```
{
  "buyCommission": "string",
  "companyName": "string",
  "message": "string",
  "numberOfSharesHeld": "string",
  "sellCommission": "string",
  "value1dayAgo": "string",
  "value1weekAgo": "string",
  "value2daysAgo": "string",
  "value3daysAgo": "string",
}
```

Response Content Type
application/json

Parameters

Parameter	Value	Description	Parameter Type	Data Type
<b>customer</b>	(required)		path	string
Authorization			header	string
<b>company</b>	(required)		path	string
<b>pass</b>	(required)		query	string

Try it out!

11. Enter **USER1** in the boxes beside customer and pass. Enter **Basic RnJlZDpmcmVkcHdk** in the box beside Authorization and **1** in the boxes beside company and number. Press the **Try it out!** button.

GET /getQuoteValue/{company}/{customer}

Response Class (Status 200)  
normal response

Model Example Value

```
{
  "buyCommission": "string",
  "companyName": "string",
  "message": "string",
  "numberOfSharesHeld": "string",
  "sellCommission": "string",
  "value1dayAgo": "string",
  "value1weekAgo": "string",
  "value2daysAgo": "string",
  "value3daysAgo": "string",
}
```

Response Content Type application/json ▼

Parameters

Parameter	Value	Description	Parameter Type	Data Type
customer	user1		path	string
Authorization	Basic RnJlZDpmcmVkcHdk		header	string
company	1		path	string
pass	user1		query	string

Try it out! [Hide Response](#)

12. Scroll down the view and you should see the *Response Body* which contains the results of the GET method (see below).

#### Response Body

```
{
  "value3daysAgo": "00072.00",
  "buyCommission": "010",
  "companyName": "Casey_Import_Export",
  "value1dayAgo": "00077.00",
  "message": "Request Completed OK",
  "value1weekAgo": "00059.00",
  "numberOfSharesHeld": "0001",
  "valueOfSharesHeld": "000000079.00",
  "valueNow": "00079.00",
  "value5daysAgo": "00065.00",
  "sellCommission": "007",
  "value4daysAgo": "00070.00",
  "value6daysAgo": "00063.00",
  "value2daysAgo": "00078.00"
}
```

#### Response Code

200

13. Select the *PUT* method for selling shares of stock by clicking on the */sellShares/{company}/{number}/{customer}* URI string. Remember this was the *Path* specified for the *PUT* method for the *sellShares* service when the API was defined. This action will expand this method in this view and provides a Swagger UI test client (you may have to use the slider bar and adjust the perspective to see the entire client).

PUT

/sellShares/{company}/{customer}

Response Class (Status 200)

OK

Model | Example Value

```
{
  "message": "string"
}
```

Response Content Type

application/json

Parameters

Parameter	Value	Description	Parameter Type	Data Type
Authorization			header	string
company	(required)		path	string
customer	(required)		path	string
pass	(required)		query	string
putSellShares_request	<div>-</div> <div>numberOfShares</div> <div></div>	request body	body	<div>Model   Example Value</div> <div> <pre>{   "numberOfShares": "string" }</pre> </div>

Parameter content type: application/json

Try it out!

14. Enter *USER1* in the boxes beside customer and pass. Enter **Basic RnJlZDpmcmVkcHdk** in the box beside Authorization and *1* in the boxes beside company and number of shares. Press the **Try it out!** button.

15. Scroll down the view and you should see the *Response Body* which contains the results of the PUT method (see below). The message *Request Completed OK* should be the contents of the response body message. This is the message displayed on the 3270 screen when a buy is successful.

**Request URL**  
`https://wg31.washington.ibm.com:9453/trader/sellShares/1/1/user1?pass=user1`

**Request Headers**  

```
{
  "Accept": "application/json",
  "Authorization": "Basic RnJlZDpmcmVkcHdk"
}
```

**Response Body**  

```
{
  "message": "Request Completed OK"
}
```

**Response Code**  
`200`

Try other combinations of customers and stocks to buy and sale stocks. Also, logon to CICS and use the TRDR transaction to do the and check the result the Swagger UI test facility. Also consider starting CEDX traces on transactions TRDR and TRAD to trace the execution of the CICS programs when driven by the Swagger UI test facility.

## Summary

You have verified the API. The API layer operates above the service layer you defined and tested earlier. The API layer provides a further level of abstraction and allows a more flexible use of HTTP verbs, and better mapping of data via the API editor function.