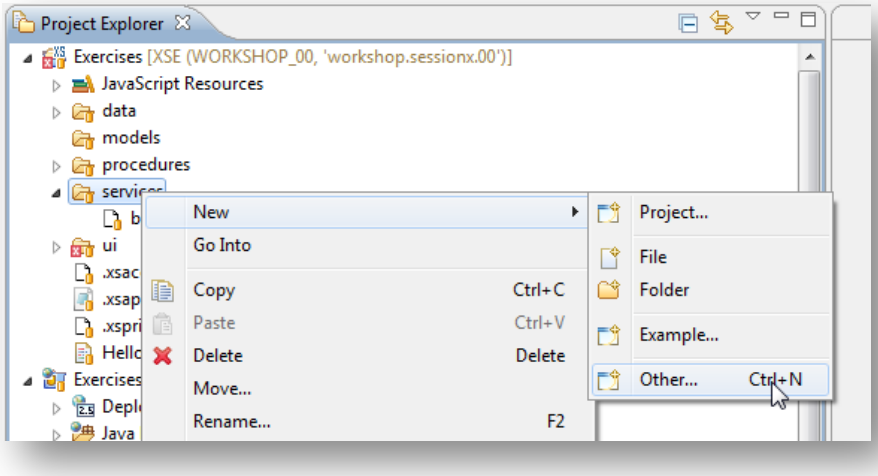
Exercise 5: XSJS – Server Side JavaScript Service

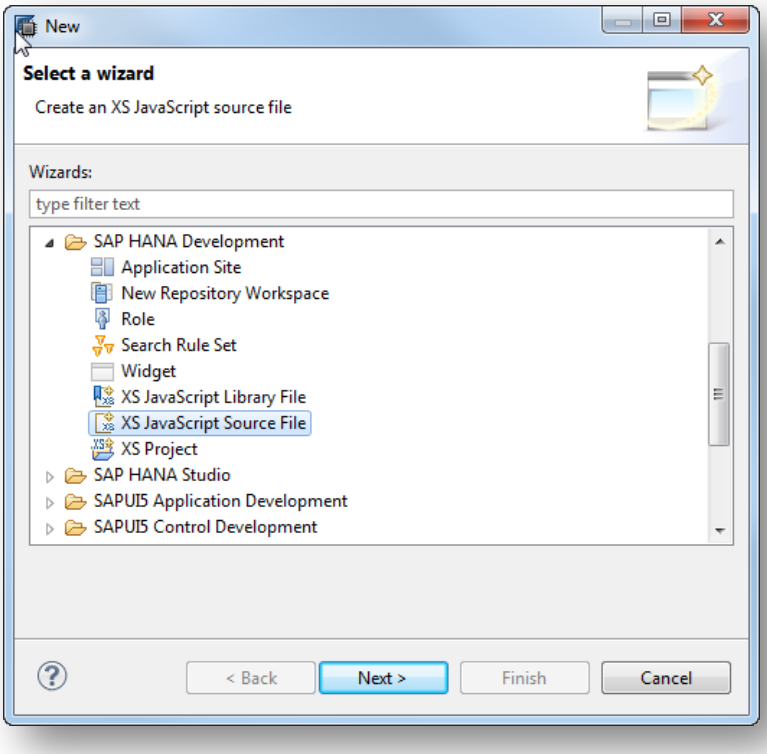
The XSODATA services are a very fast way to build OData services for existing database artifacts. However, sometimes you need more control over the processing logic of the service handler or a custom output format. For this reason, we can also write custom service handlers using server side JavaScript.

Creating a simple XSJS Service

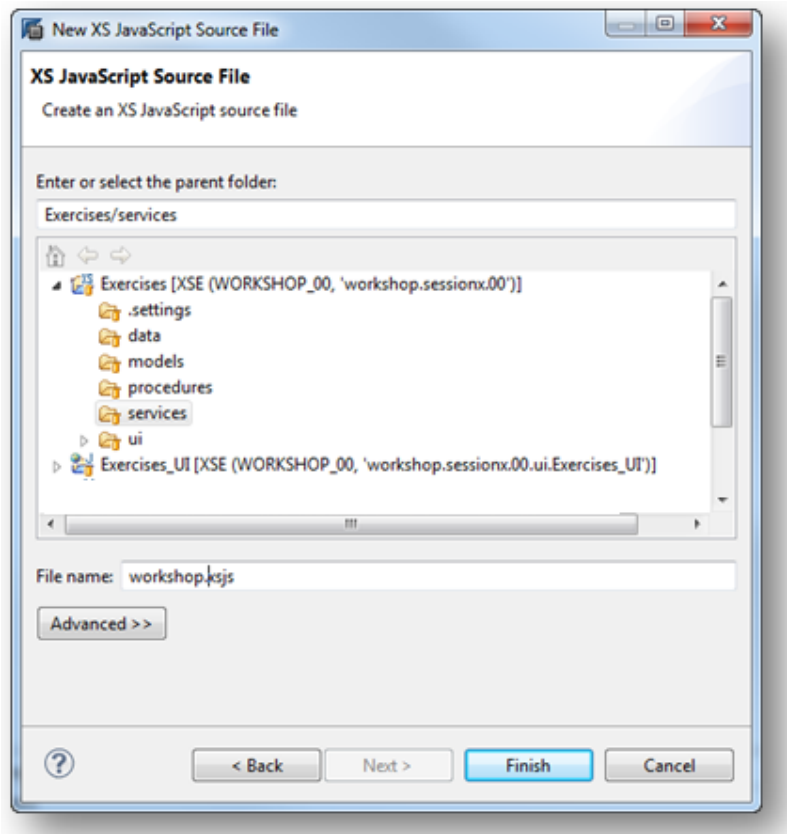
1. Use **New->Other** from the services package of your project.



2. From the New Wizard, choose XS JavaScript Source File from the SAP HANA Development folder.



3. Name your new source file: workshop.xsjs.



5. We want to create a service, which will accept two input integer values and multiply them together. The results of this math will be passed back as text in the body of the response object. If you do not want to type it, copy from here:

Source Code:

```
function performMultiply(){
        var body = '';
        var num1 = $.request.parameters.get('num1');
        var num2 = $.request.parameters.get('num2');
        var answer;

        answer = num1 * num2;

        body = answer.toString();

        $.response.setBody(body);
        $.response.status = $.net.http.OK;
}

var aCmd = $.request.parameters.get('cmd');
switch (aCmd) {
        case "multiply":
                performMultiply();
                break;
        default:
                $.response.status = $.net.http.INTERNAL_SERVER_ERROR;
                $.response.setBody('Invalid Command: '+aCmd);
}
```

6. We will explain it block by block in a moment. We begin the XSJS file with the function which will handle the primary action of our service handler – performing the multiplication. It reads the two input parameters from the request object. After multiplying them together, it places the result back into the response body. It also sets the response return code to OK.

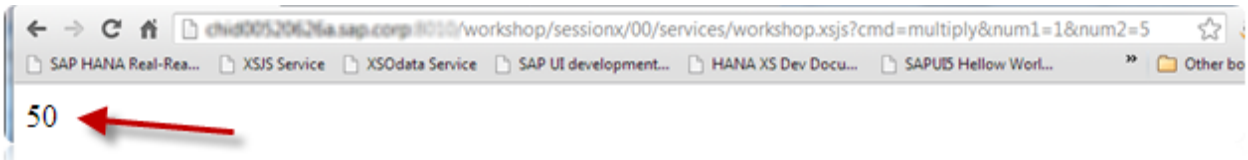
```
function performMultiply(){
    var body = '';
    var num1 = $.request.parameters.get('num1');
    var num2 = $.request.parameters.get('num2');
    var answer;

    answer = num1 * num2;

    body = answer.toString();

    $.response.setBody(body);
    $.response.status = $.net.http.OK;
}
```

7. This block is actually the entry point to the service handler but must appear after all referenced functions. Here we can query the request URL and determine which action to take. This way a single XSJS file might have multiple services or query parameters. We will look for the URL parameter called **cmd** to decide which path to take. We only have the ***cmd=multiply***. These are just examples, as you have complete control over the definition of all URL parameters and how they are handled by the service.

```
var aCmd = $.request.parameters.get('cmd');
switch (aCmd) {
case "multiply":
    performMultiply();
    break;
default:
    $.response.status = $.net.http.INTERNAL_SERVER_ERROR;
    $.response.setBody('Invalid Command: '+aCmd);
}
```

8. Save, commit and activate the new service handler file.

9. Test the service in your browser. The URL to run your service would be http<host>:<port>/workshop/session<session>/<group>/services/workshop.xsjs?cmd=multiply&num1=10&num2=5. For example if your session letter was x and your group number was 00 then the URL would be: http://<host>:<port>/workshop/sessionx/00/services/workshop.xsjs?cmd=multiply&num1=10&num2=5

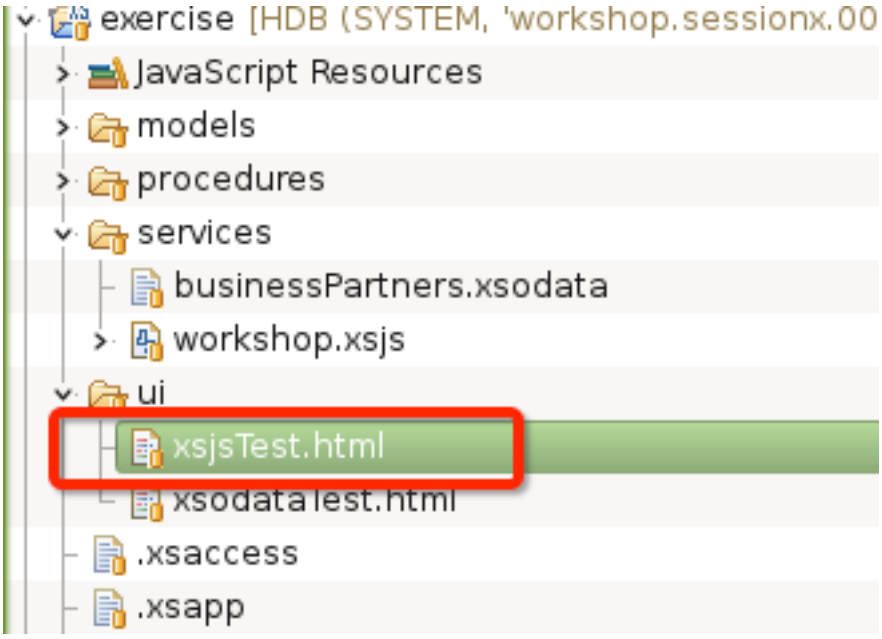


Calling the XSJS Service From the User Interface

1. While the XSJS was relatively easy to test from the web browser directly via URL parameters, it is also beneficial to see what kind of client side JavaScript could be used to integrate such a service call into the user

interface. In this exercise we will build a user interface with two input fields for the numeric values which are multiplied. We will then call the service on the server when the user changes the value in either field. Note: of course you could perform simple math on the client side. We are using this very simple example in order to focus on the techniques for creating and calling the XSJS services without needing to get into too much detail.

2. Like the previous exercise, create a new html file called **xsjsTest.html** in the **ui** package.



3. We will introduce the code by blocks. At first, we defined a new function **onMultiply** and we make an ajax call to the server side java scripts with the values of the parameters get from the ui elements.

```
function onMultiply() {
    var aUrl = '../services/workshop.xsjs?cmd=multiply'+'&num1='+escape(oVal2.getValue())+'&num2='+escape(oVal1.getValue());
    jQuery.ajax({
        url: aUrl,
        method: 'GET',
        dataType: 'json',
        success: this.onCompleteMultiply
    });
}
```

4. Then we create the ui elements, 2 text fields to allow user to input the parameters and one button, which will trigger the call to the services and perform the calculation.

5. To avoid having to do too much typing, we have prepared some template code for you. You can copy from here:

Source Codes:

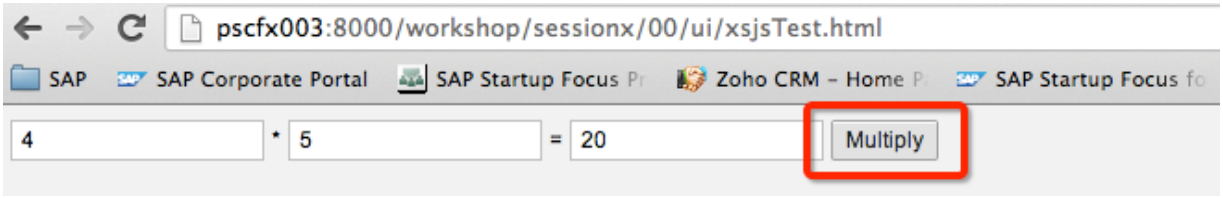
```
<!DOCTYPE HTML>
<html>
<head>
<meta http-equiv="X-UA-Compatible" content="IE=edge">

<script src="/sap/ui5/1/resources/sap-ui-core.js" id="sap-ui-bootstrap"
       data-sap-ui-libs="sap.ui.commons"
       data-sap-ui-theme="sap_goldreflection">
</script>

<script>
       var layoutNew = new sap.ui.commons.layout.MatrixLayout({
              width : "auto"
```

```
});
var oVal1 = new sap.ui.commons.TextField("val1", {
	tooltip : "Value #1",
	editable : true
});
var oMult = new sap.ui.commons.TextView("mult", {
	tooltip : "Multiply by",
	text : " * "
});
var oVal2 = new sap.ui.commons.TextField("val2", {
	tooltip : "Value #2",
	editable : true
});
var oEqual = new sap.ui.commons.TextView("equal", {
	tooltip : "Equals",
	text : " = "
});
var oResult = new sap.ui.commons.TextField("result", {
	tooltip : "Results"
});

var myButton = new sap.ui.commons.Button("btn");
myButton.setText("Multiply");
myButton.attachPress(function(){
	onMultiply();
});

function onMultiply() {
	var aUrl = '../services/workshop.xsjs?cmd=multiply' +
	'&num1='+escape(oVal1.getValue()) +
	'&num2='+escape(oVal2.getValue());

	jQuery.ajax({
		url: aUrl,
		method: 'GET',
		dataType: 'json',
		success: this.onCompleteMultiply
	});
}

function onCompleteMultiply(myTxt){
	var oResult = sap.ui.getCore().byId("result");
	oResult.setValue(myTxt);
}

layoutNew.createRow(oVal1,oMult,oVal2,oEqual,oResult,myButton);
layoutNew.placeAt("content");
</script>

</head>
<body class="sapUiBody" role="application">
	<div id="content"></div>
</body>
</html>
```

6. Save, commit and activate the file you've just created.

7. Test your **xsjsTest.html** application from your web browser. The URL to run your test application would be http://<host>:<port>/workshop/session<session>/<group>/ui/xsjsTest.html. For example if your session letter was x and your group number was 00 then the URL would be: http://<host>:<port>/workshop/sessionx/00/ui/xsjsTest.html

   Input the number values in the first two text fields and click the Multiple button to see the result the 3rd text field.

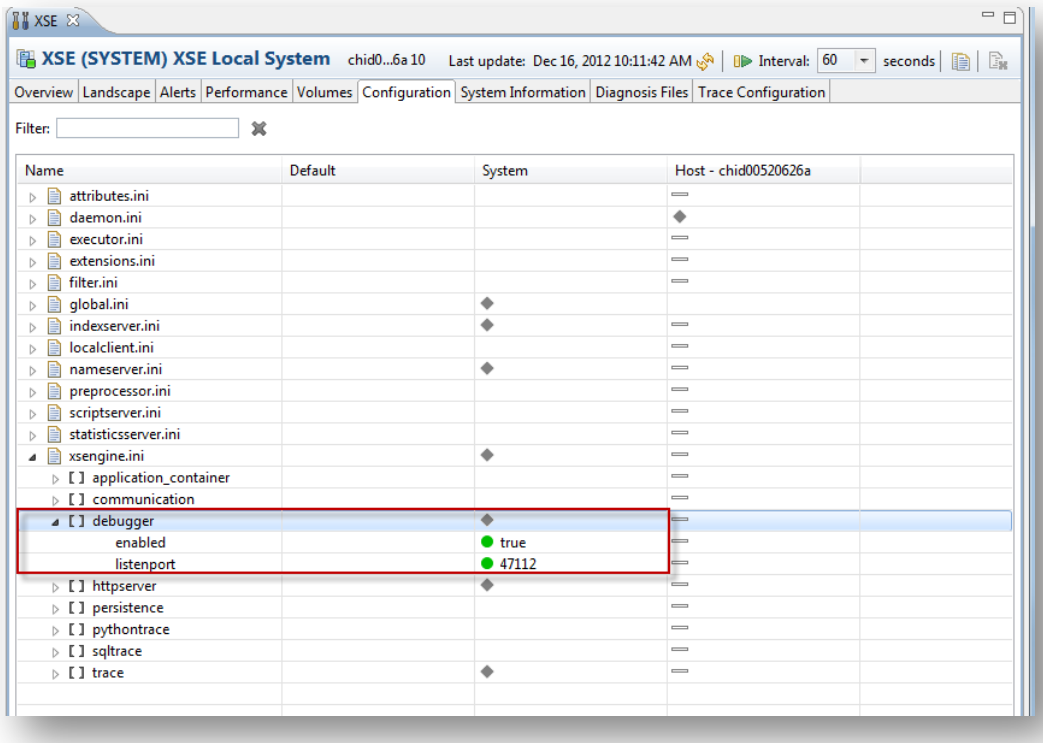


Optional: XSJS – Server Side JavaScript Debugging

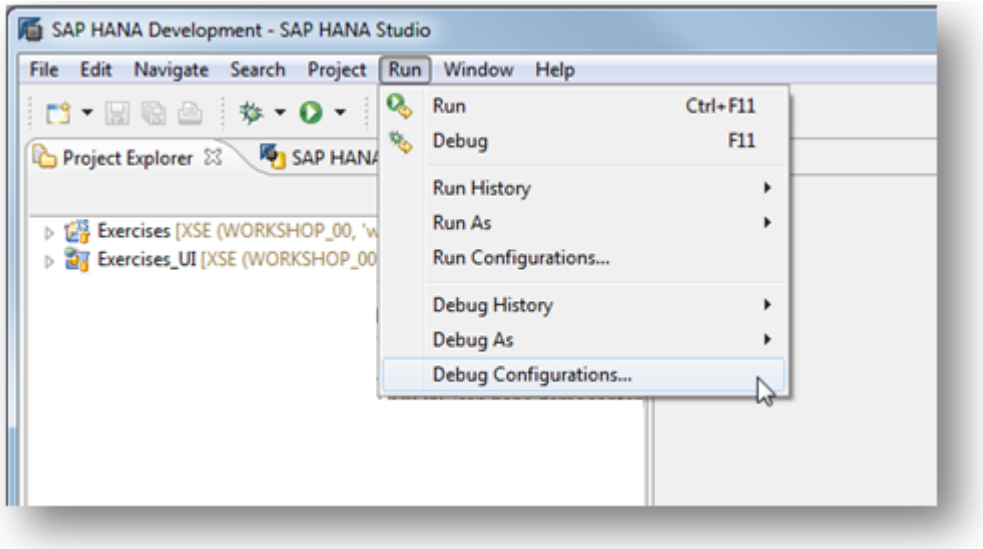
Configuring the Debugger

For each new system, there is a one-time setup required to create the debug configuration in your SAP HANA Studio. Just continue a few steps and enable yourselves to debug Server Side JavaScript. Isn't it cool?

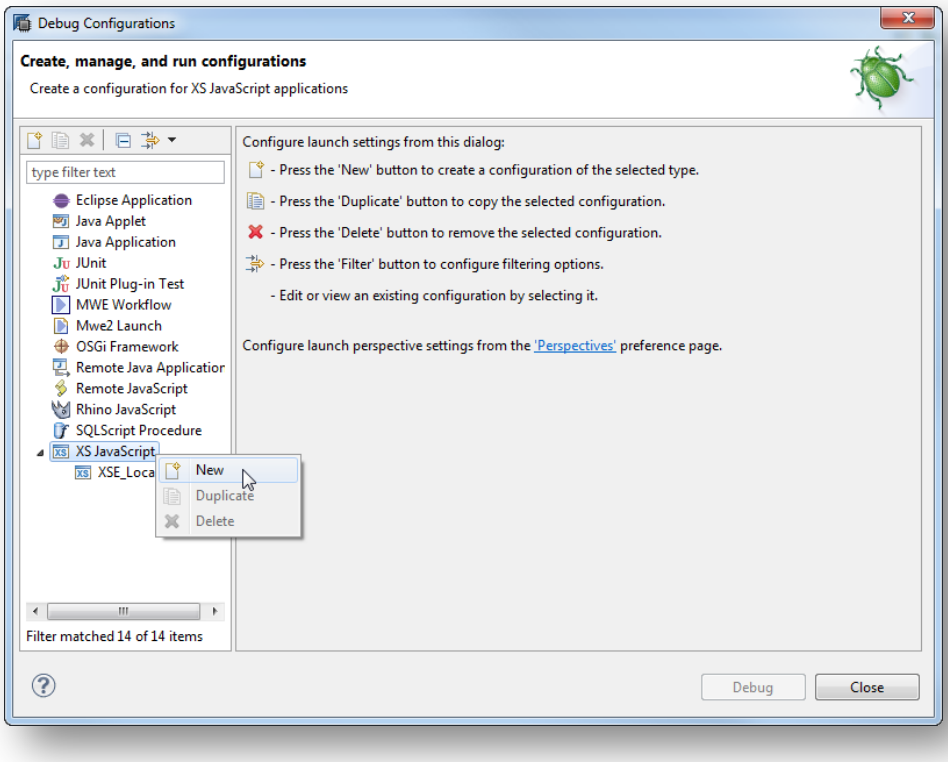
1. Although the system configuration for debugging might be already completed on the system, it might be useful for future to knowledge to see where this is done. You can follow these steps to enable debugging in your system:
   I. From the System Administration tool, go to the **Configuration** tab.
   II. Under **xsengine.ini** you have to create a new section called **debugger**
   III. Add a parameter named **enabled** with the value **true**.
   IV. Add a second parameter named **listenport**. The default value is **47112**.



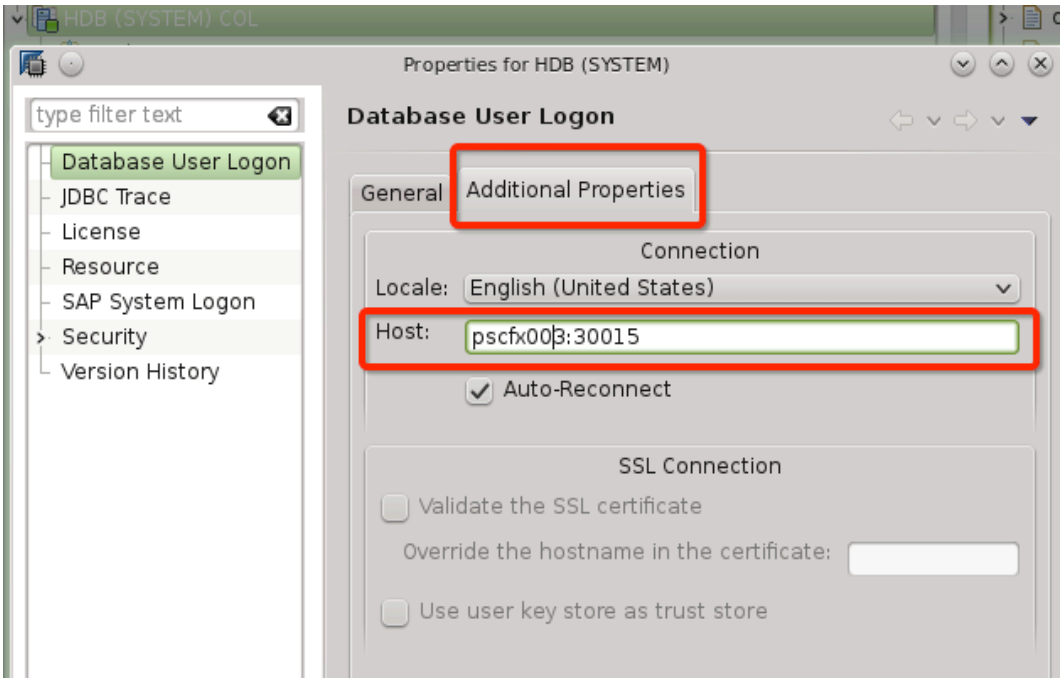
2. Then let's configure the Debug Configurations, Choose **Run -> Debug Configurations…**



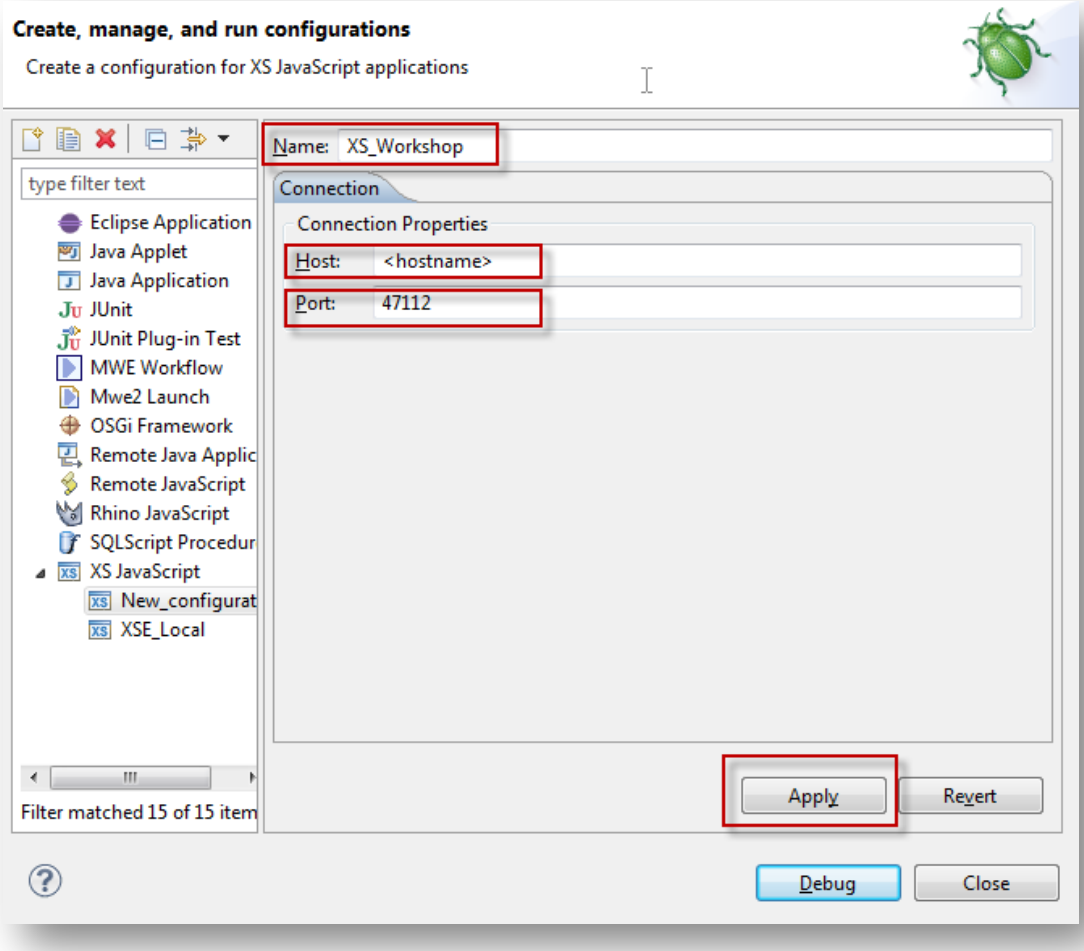
3. In the Debug Configurations dialog, choose XS JavaScript. Right mouse click on this entry and choose New.



4. Name your Debug Configuration XS_Workshop. Supply the hostname and the default debug port of 47112. If you forget the host name, right click your system, select the **Properties** from the context menu, go to the **Additional Properties** and you will see the host name as highlighted in red. You will just need the host name only, not the port 30015.



5. Finally press **Apply**.



Interactive Debugging

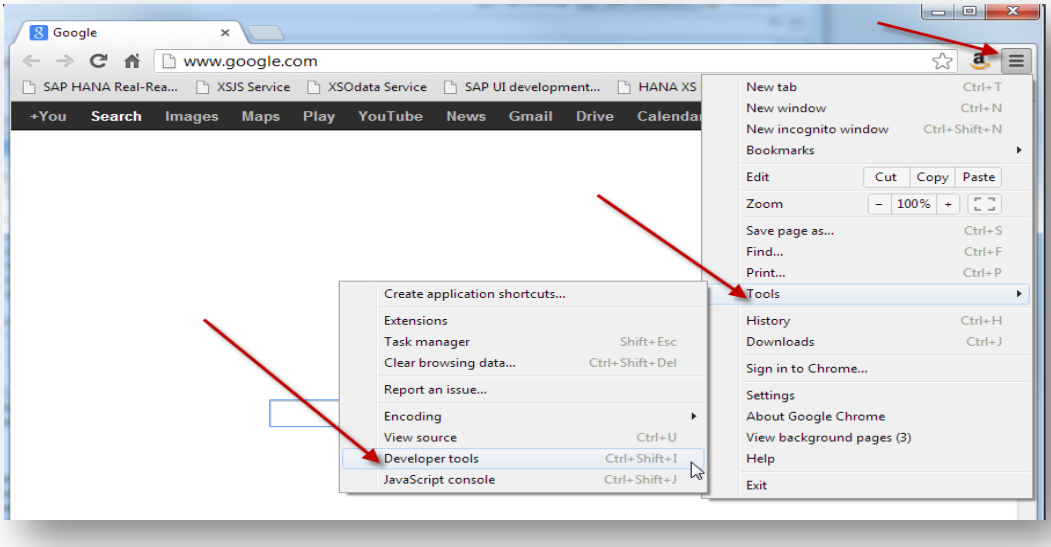
1. We can now use the interactive debugger against the xsjs service which we created in the previous exercise. workshop.xsjs in package workshop.sessionx.00.services.

2. Open the workshop.xsjs service in the editor and navigate to the **performMultiply** function. You can set breakpoints by double clicking in the bar to the far left.

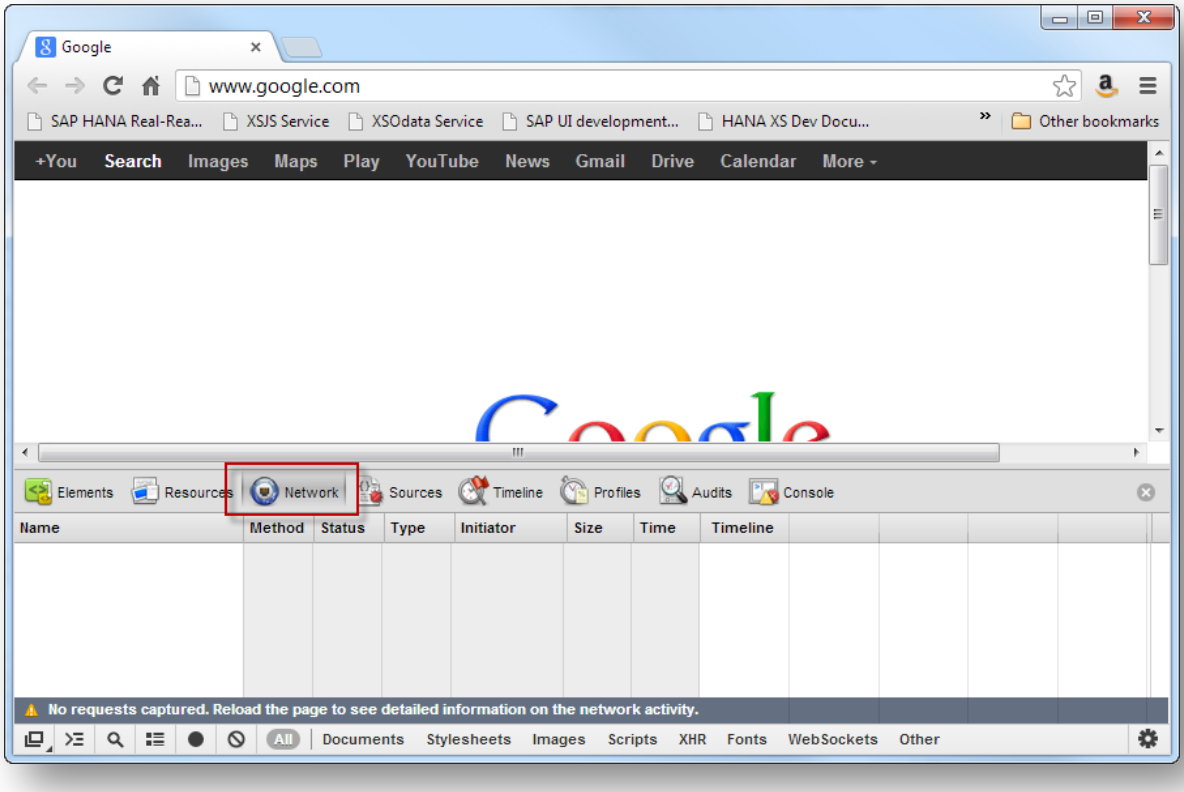
*HDB - SQL Console 2 | xsjsTest.html | workshop.xsjs

```
function performMultiply(){
    var body = '';
    var num1 = $.request.parameters.get('num1');
    var num2 = $.request.parameters.get('num2');
    var answer;

    answer = num1 * num2;

    body = answer.toString();

    $.response.setBody(body);
    $.response.status = $.net.http.OK;
}

var aCmd = $.request.parameters.get('cmd');
switch (aCmd) {
case "multiply":
    performMultiply();
    break;
default:
    $.response.status = $.net.http.INTERNAL_SERVER_ERROR;
    $.response.setBody('Invalid Command: '+aCmd);
}
```

3. Before we can start the debugger we must first test the service once in a web browser in order to know our session id. The session id must be supplied to the debugger on start up so it knows which session to connect to and debug. Launch Chrome and open the developer tools.

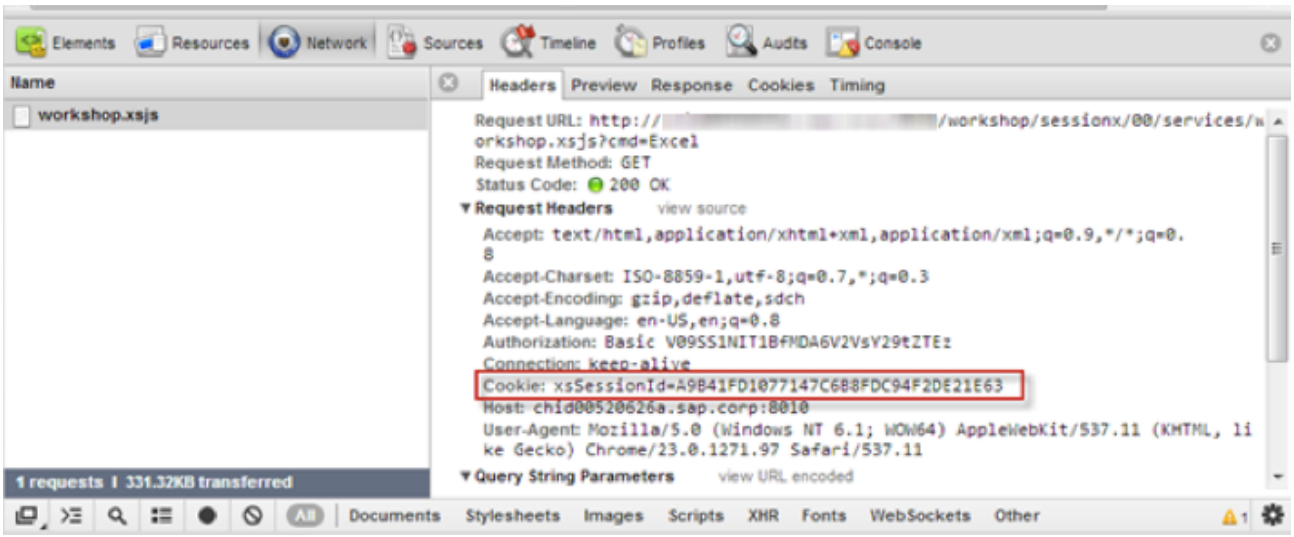


4. Make sure you are on the Network tab of the developer tools.

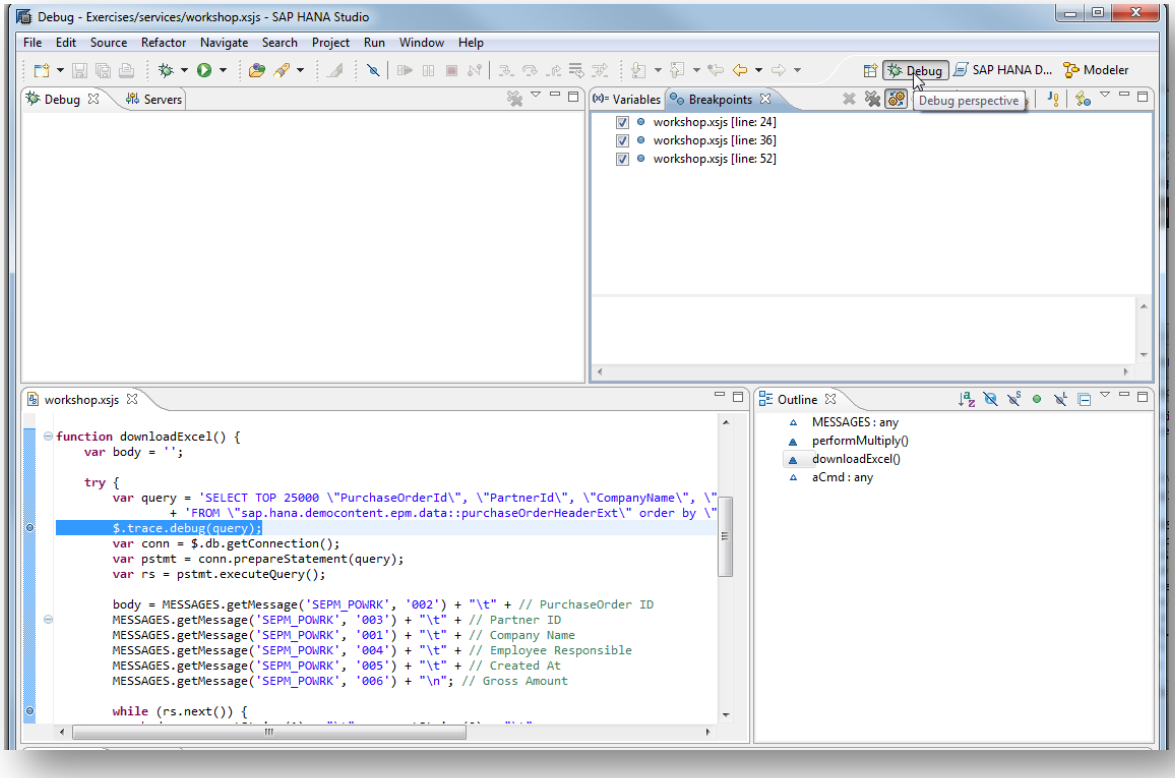


5. Test the application in your browser. For example if your session letter was x and your group number was 00 then the URL would be: http://<host>:<port>/workshop/sessionx/00/services/workshop.xsjs?cmd=multiply&num1=10&num2=5

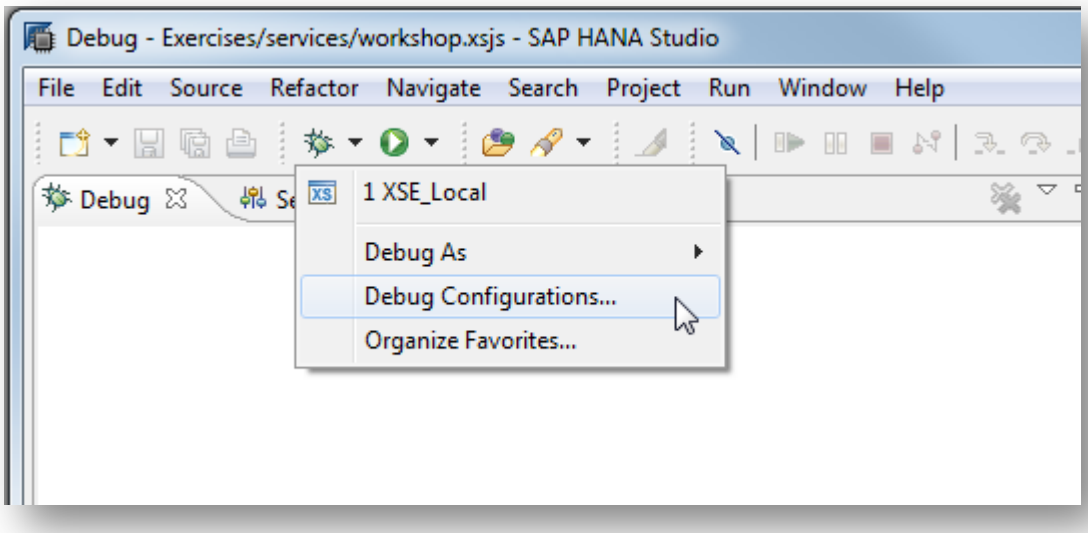
6. After calling the service you can display the Headers details of the request to workshop.xsjs. In the Request Headers there should be a cookie called xsSessionId. The value of this cookie will be the value you in the next steps.



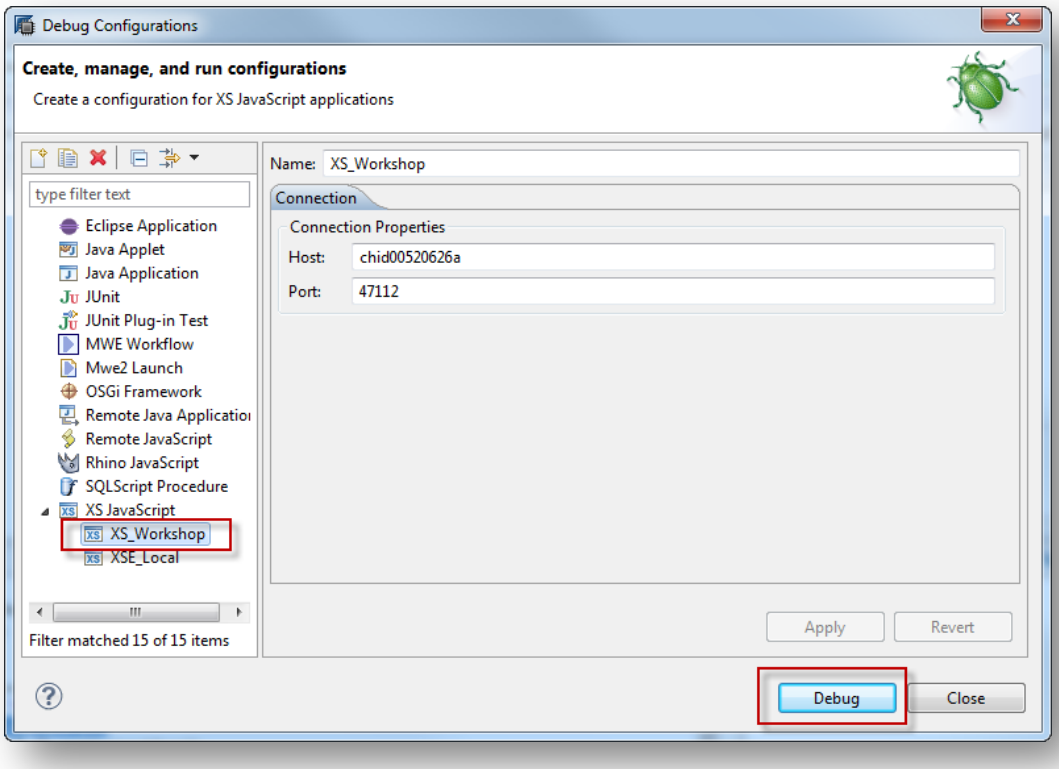
7. Switch back to SAP HANA Studio **without closing your web browser**. Switch to the **Debug** perspective.



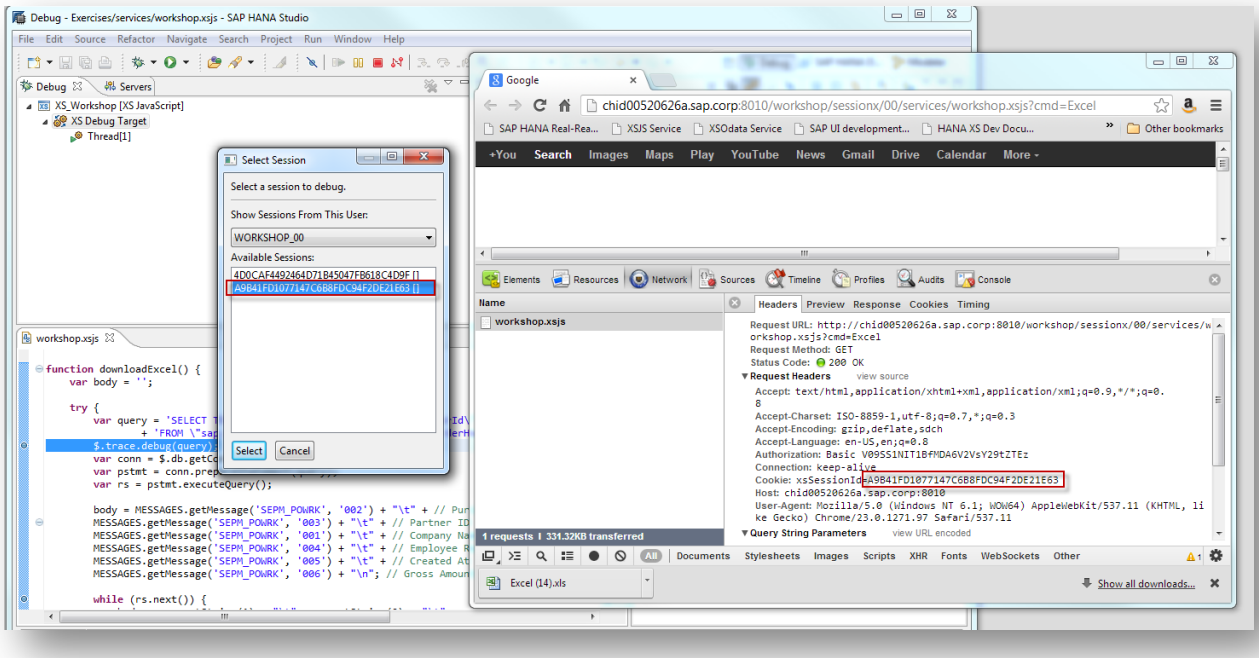
8. Under the debug icon click the expand arrow and then choose Debug Configurations.



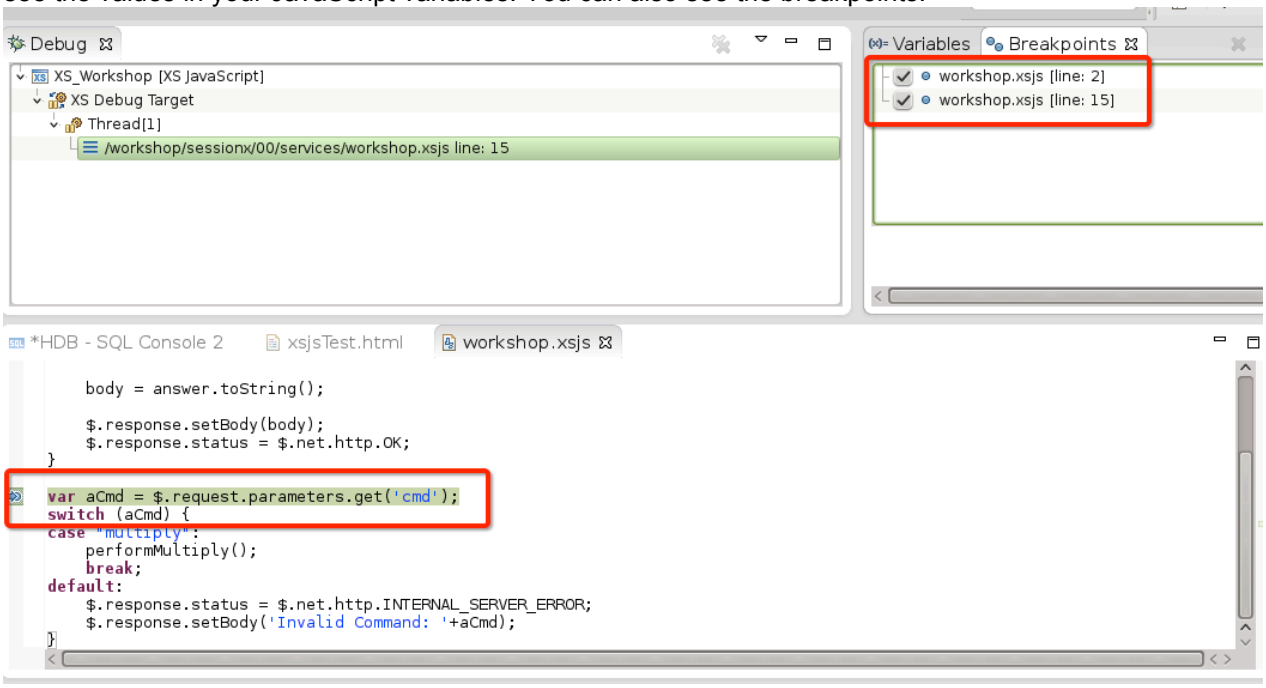
9. Choose the **XS_Workshop** debug configuration you created in the previous part of this exercise and then press Debug.



10. A dialog will come up asking you to select your session. Make sure that your user is selected in the first drop down list box. Then match the session id in the dialog to the one in your web browser.



11. You can now refresh your web browser to make another request to this same service. This should start the interactive debugger. You should see the debugger stopping processing on the first breakpoint. You can also see the values in your JavaScript variables. You can also see the breakpoints.



The Best-Run Businesses Run SAP™