

HANA

HANA Studio Developer Demo

Contents

[Introduction 2](#_Toc339874740)

[Creating a Workspace and Project 2](#_Toc339874741)

Basic Project Files [9](#_Toc339874742)

[Creating Data 13](#_Toc339874743)

[Accessing Data](#_Toc339874743) 14

[Server Side JavaScript](#_Toc339874743) 19

[Application Site](#_Toc339874743) 21

Migrating Content28

Documentation Links33

## Introduction

This tutorial will demonstrate how to develop an application in SAP HANA Studio using SAP UI5 and creating data that can be easily transported and referenced through OData calls. The goal is to create a simple web application showing Hello World in a table in several different languages. Prerequisites for the tutorial are:

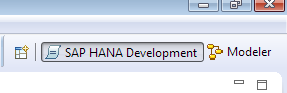
* An SAP HANA Database Server with at least revision 48
* SAP HANA Studio installed with the same revision as the Database Server
* The SAPUI5 toolkit installed in HANA Studio
* The HANA system added to HANA Studio

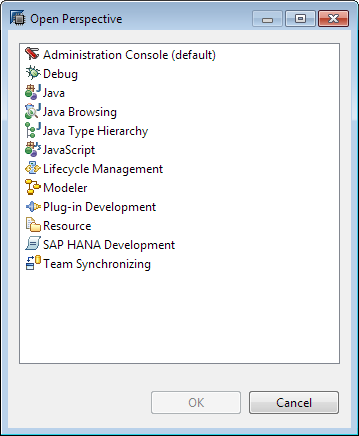
Instructions on preparing HANA Studio for this project can be found in the HANA Studio Installation and Setup Demo.

## Creating a Workspace and Project

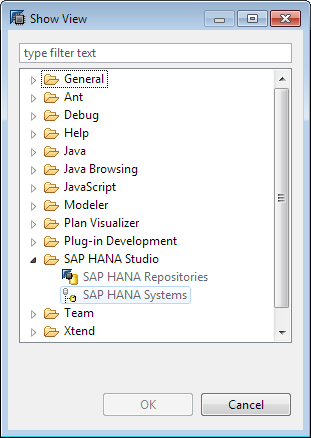
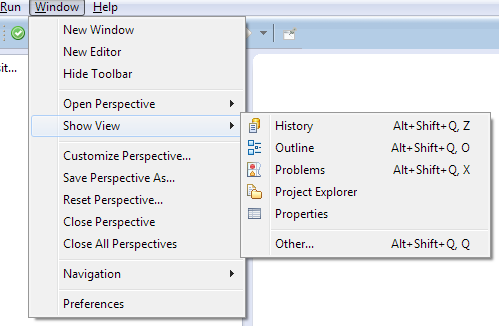
The first step is to create a repository workspace in HANA Studio. This is the location for all development files that allows for version control when sharing between developers, and easy transport by placing the files in a package in the HANA environment.

With HANA Studio open, switch to the SAP HANA Developer perspective. In the upper right hand corner of the main screen, all perspectives that have been used should be displayed. If the Developer perspective has not yet been used, selecting the box with a yellow plus sign will allow you to add another perspective to that list.

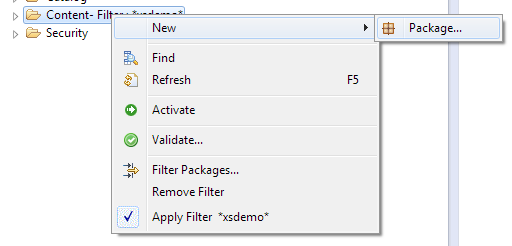




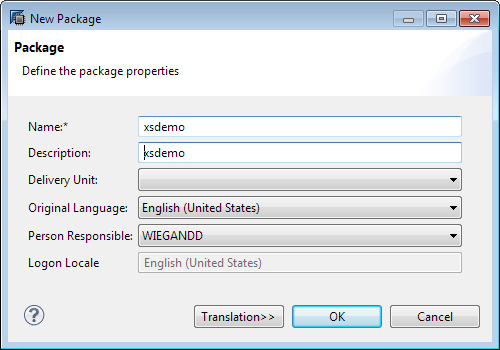
In the Developer perspective, the SAP HANA Systems, Project Explorer, and SAP HANA Repositories should be visible. In versions of HANA Studio prior to revision 55, SAP HANA Systems will instead be the Navigator. If any are not present, go to the Window menu and Show View. If any are not showing on the immediate menu, click “Other…”. Project Explorer can be found in the “General” folder, and SAP HANA Repositories and Navigator/SAP HANA Systems can be found under SAP HANA Studio. There is also a Navigator option in the General folder, but it is not the same Navigator that is used in older versions.

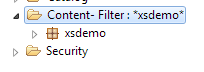


The workspace will allow you to place all files in a HANA package for ease of transport. Before we create a new workspace, we should make a new package to contain our XS project. In the SAP HANA Systems pane, expand the system and right click the Content node and go to “New” -> “Package”.

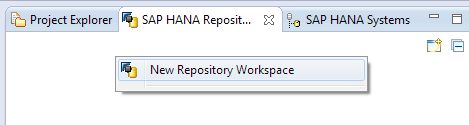


Fill in the name of the package you want to create; ignore the delivery unit for now. The package will now be visible in the content folder in the Navigator.

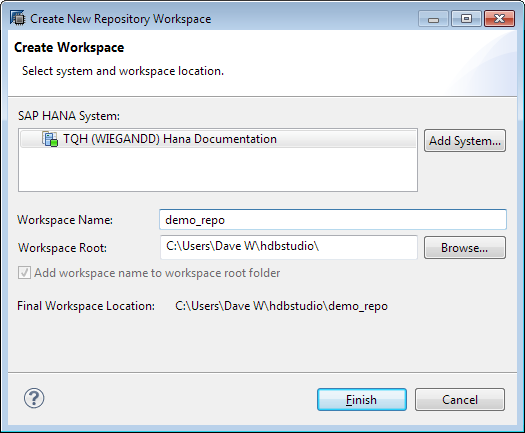




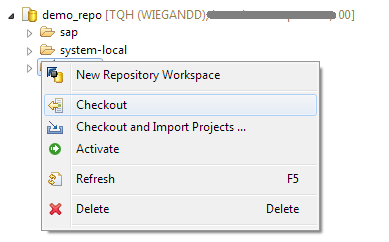
Once the package is created, the repository is next. Go to the SAP HANA Repositories tab, right click, and select New Repository Workspace.



Select the system to create the workspace on, choose a workspace name, and select a location for the files to be placed on the local machine.



Expand the new repository and right click the folder for the package that was created for the project, then select Checkout.

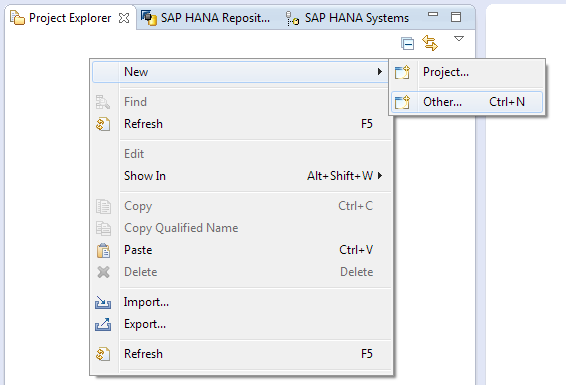


A checkmark will appear on the folder. Any content that was already in the folder (there should be none) will be copied to the local machine, and any future content will have a file on the local machine in a folder for the selected package in the location selected for the workspace.

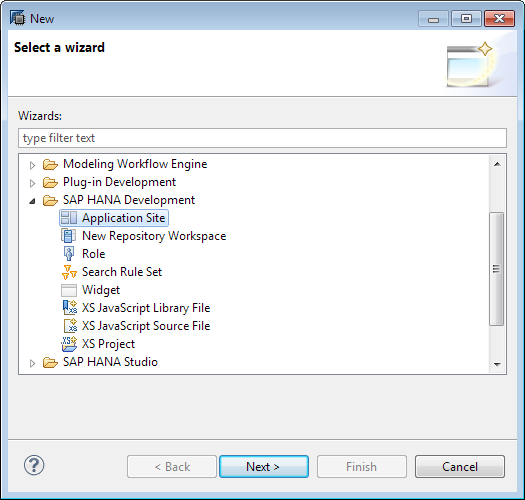


Now that we have a repository workspace, we need to create a project within it. Projects group together all application-related artifacts, containing folders and files for the application. Multiple projects can be placed into one repository workspace.

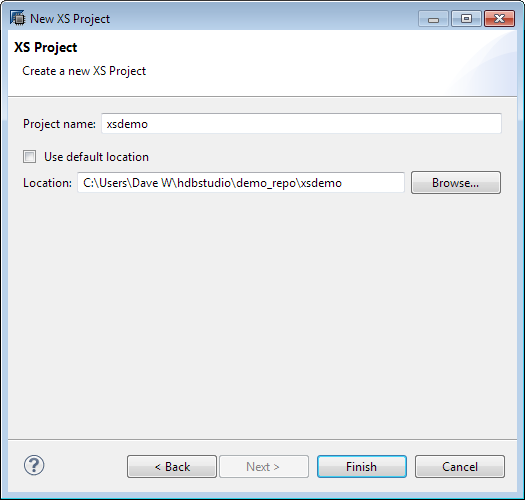
To create the project, go to the Project Explorer tab, right click, and select “New” -> “Other”.



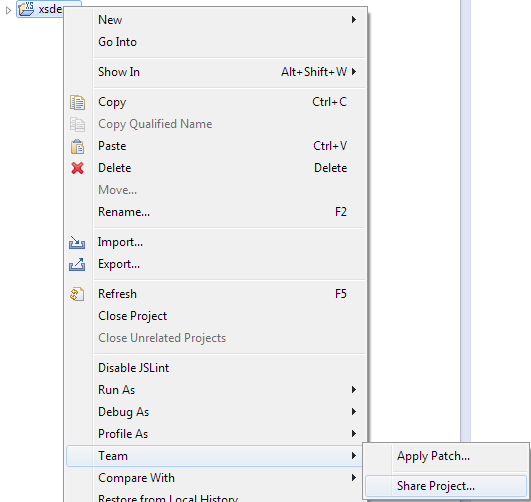
Expand the SAP HANA Development folder and select XS Project.



Choose a name for the project. Uncheck “Use default location” and navigate to the folder on your local machine for the checked out package; this will allow all content for the project to be associated with that package in HANA Studio.



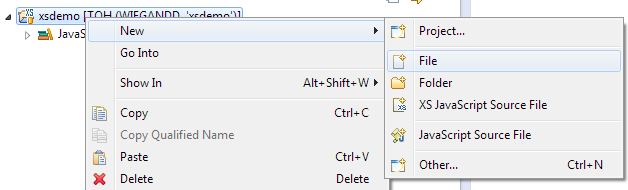
To connect the project to the HANA server so that files can be committed to the repository and activated, right click the project name and go to “Team” -> “Share Project…”.

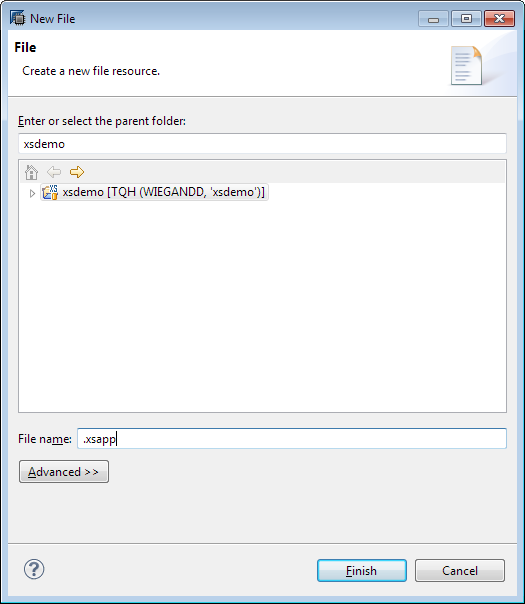


Now the project is created, and we are ready to start adding content.

## Basic Project Files

Before we can add any actual functionality to the application, we need to add files that will let the server recognize that there is XS application content in the project. The first file necessary is the .xsapp file. Right click the project name and go to “New” -> “File”, and type .xsapp in for the name. When you click create, it will open in a text editor, but this file should remain completely blank, so you can close the text editor.





An .xsaccess file will also need to be created to control access to the project. Again, create a new file, this time called .xsaccess. This will need some content; the code is in JSON format and should be as follows:

{

"exposed":true,

"authentication": [

{

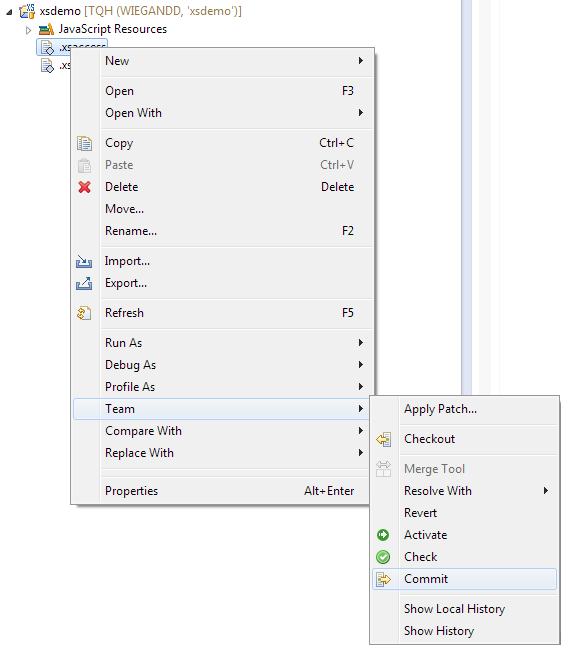
"method":"Basic"

} ]

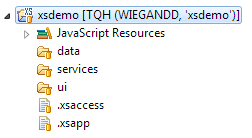
}

This exposes the application to the web server, and enables basic authentication, which will allow users to log in with their HANA credentials. For other options such as form-based authentication or SAP logon tickets, refer to the SAP HANA Developer’s Guide.

When the .xsapp and .xsaccess files are saved, they both need to be committed to the server and then activated. Right click the file name, select “Team” -> “Commit”, then right click and select “Team” -> “Activate”. You can either commit and activate each file individually, or select multiple using shift or control. It is also possible to select activate or commit on the project name, and that will perform the command on all files within the project.

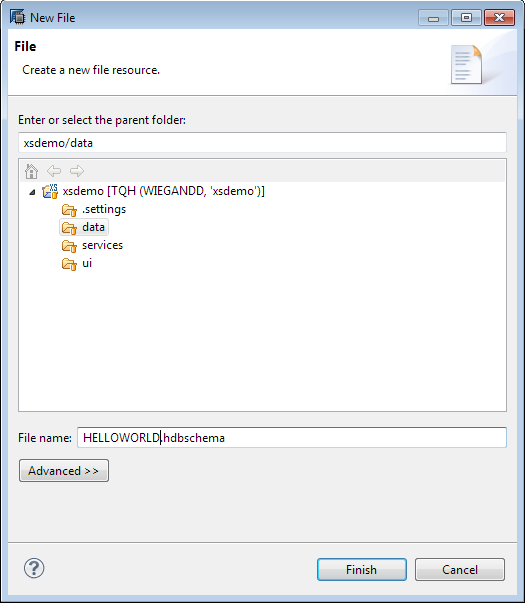


Also before adding content, it will help to create folders to organize the parts of the application. The folders we will create are data, services, and UI. They are created by right clicking the project and going to “New” -> “Folder”.



Next, we will need to create a schema to contain the data files we will create, and create roles with access to that schema so that users can access it. A schema can easily be created using SQL, but that will not be easily transportable; in the project, we can create a .hdbschema file that can be exported and imported with the rest of the project. Our schema will be called HELLOWORLD; to create it, right click the data folder and create a new file. The name of the file needs to be the name of the schema, so name it HELLOWORLD.hdbschema. When it opens in the text editor, input the following syntax:

schema\_name="HELLOWORLD";



Once that is saved, commit and activate the schema, and we will move on to the roles and privileges for the schema and application. In the root folder of the application, create a new file called .xsprivileges. We will make a basic set of privileges, and a set of administrative privileges as follows:

{

"privileges":[

{

"name":"Basic",

"description":"Basic user privileges"

},

{

"name":"Admin",

"description":"Administrative privileges"

}

]

}

This simply creates a list of different types of privileges; actual access can be controlled by other files. The .xsaccess file can limit authorization to the package based on privileges; refer to the SAP HANA Developer’s Guide for information on how to do this. Commit and activate the privileges file, then create the roles. In the data folder, we will make hwUser.hdbrole and hwAdmin.hdbrole. Here is the syntax for the user:

role <basepackage>.data::hwUser {

catalog schema "HELLOWORLD": SELECT;

application privilege: <basepackage>::Basic;

}

In this, the <basepackage> refers to the base package of the project, where it was placed in the repository. In the tutorial, it was created in package “xsdemo”; the location can be seen in single quotes next to the name of the project, after the user name, if the project is shared.

Z:\dwiegand\Documents\HANA Documentation\e2e demo\022_package_name.png

This role has select privileges only on the schema we created, and basic application privileges as created in the .xsprivileges file.

The hwAdministrator will have all privileges that the hwUser has (extending the role), as well as Admin privileges from the .xsprivileges file and more access to the schema we created.

role <basepackage>.data::hwAdmin

extends role <basepackage>.data::helloUser

{

catalog schema "HELLOWORLD": INSERT, UPDATE, DELETE, DROP;

application privilege: <basepackage>::Admin;

}

Once those roles are created, the system user (or any user with “GRANT\_ACTIVATED\_ROLE (\_SYS\_REPO)” SQL privileges) will need to grant them to users.

## Creating Data

Now that the schema is created, we will add a table to it. We will create a very simple table called hello\_world, with two data columns and a primary key. Create a new file named hello\_world.hdbtable with this syntax:

table.schemaName = "HELLOWORLD";

table.tableType = COLUMNSTORE;

table.description = "Hello World";

table.columns = [

{name = "ColID"; sqlType = NVARCHAR; nullable = false; length = 5; comment = "Column ID"; },

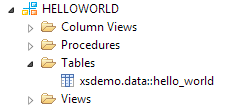
{name = "Hello"; sqlType = NVARCHAR; nullable = false; length = 10; comment = "Hello"; },

{name = "World"; sqlType = NVARCHAR; nullable = false; length = 10; comment = "World"; }

];

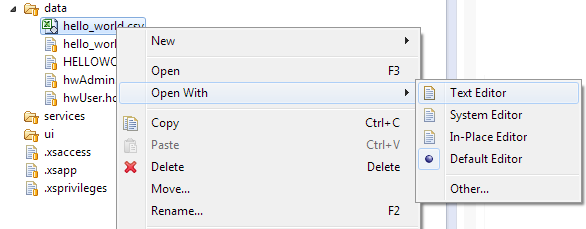
table.primaryKey.pkcolumns = ["ColID"];

This makes a simple table with 3 columns – an ID column, one for Hello, and one for World. Commit and activate the table; it will now be visible in the catalog.



While the easiest way to load small quantities of data into the table would be a SQL insert statement, we will show how to load a CSV file into the table via the repository. This requires creating three files – the .csv to be loaded, a .hdbtid file that points to the csv file, and .hdbtim file that points to the table to import into.

First is hello\_world.csv. It will likely try to open in excel; the easiest way to edit the file is to close excel and right click the file and select “Open With” -> “Text Editor”.



The data to load is:

001,Hello,World

002,Hallo,Welt

003,Ciao,Mondo

Then create the file hello\_world.hdbtid. Here is the syntax:

implements xsdemo.data:hello\_world.hdbtim;

csvFiles = ["xsdemo.data:hello\_world.csv"];

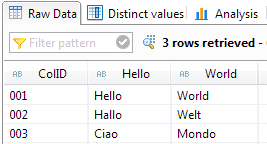
Finally, the file hello\_world.hdbtim:

listCsvFile csvFiles;

import csvFiles "HELLOWORLD" "xsdemo.data::hello\_world";

Where HELLOWORLD is the schema name and “xsdemo.data::hello\_world” is the target table.

The files can then be committed and activated; the .hdbtim file must be activated before the .hdbtid file or else the .hdbtid file activation will throw an error. Once all three files are active, the csv data will load into the table; it can be viewed with a SQL query or by data preview.



## Accessing Data

Once the table is created, we will make an OData file to access it. In the services folder, create a file called hello.xsodata. The syntax follows:

service namespace "xsdemo.services" {

"HELLOWORLD"."xsdemo.data::hello\_world" as "HelloWorld";

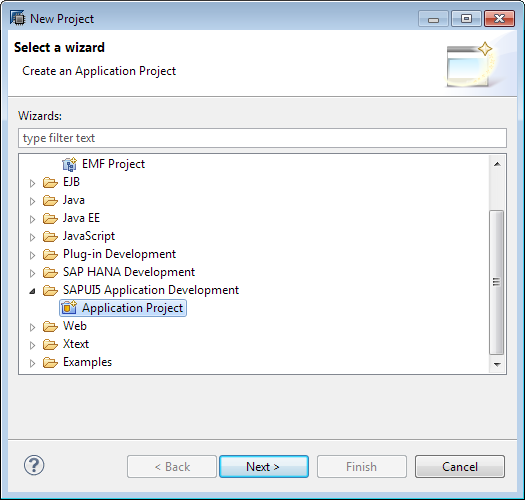
}

This just refers to the table we created by the alias HelloWorld, and allows us to connect to it via the XS engine.

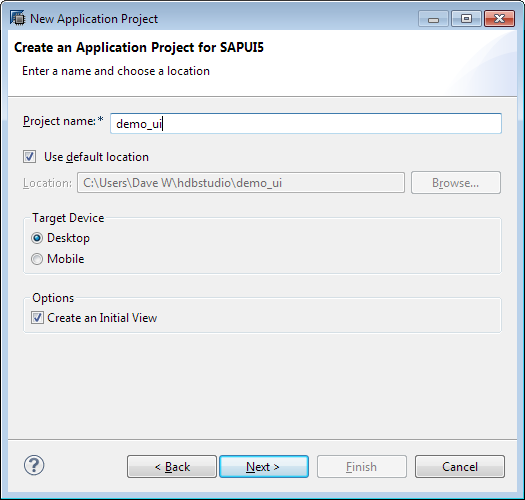
Some objects such as csv files and OData calls can be checked via a web browser. Chrome and Firefox are recommended for this purpose. The URL for XS objects is http://<domain>:80<instance #>/<package>/<filename>. Many other files, such as the hdbrole and hdbtable files, will not be accessible in this manner. The output for many of the files is not easily readble; for more information on how to read the data through the browser and arguments to add to the URL, refer to the HANA Developer’s guide.

The next step will be to create an application that shows the data from the table via the OData call. We will use the standard HANA application site, which can be used to create detailed applications with many widgets.

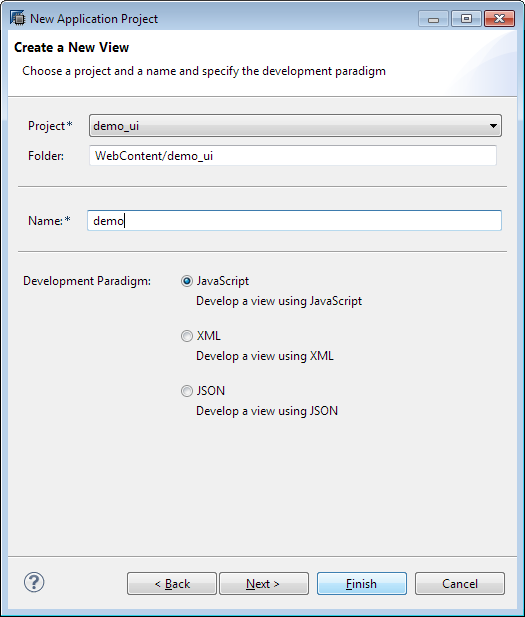
First, create an SAPUI5 application project. Right click in Project Explorer, select new, and click Project. Browse to SAPUI5 Application Development and select Application Project.

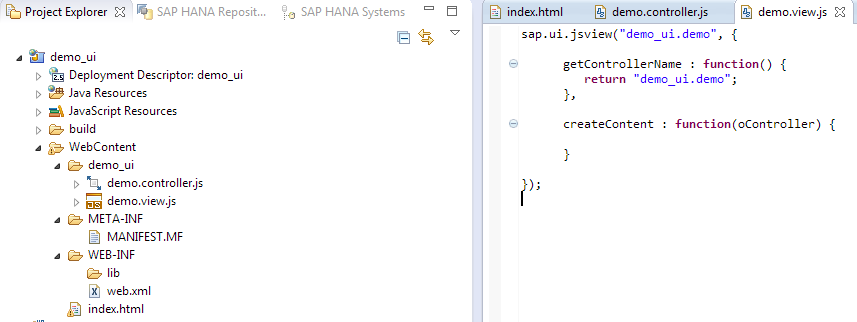


On the next screen, choose a name, demo\_ui. We will keep it as a desktop project, and have it create an initial view.

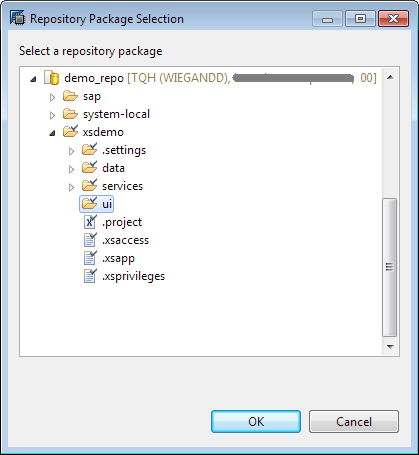


Then select a name (demo), keep it as a JavaScript view, and click finish. If HANA Studio asks if you want to open it with the Java EE perspective, select No.





Once the project is created along with an initial view, it needs to be shared into the workspace. Right click the name in the Project Explorer, go to Team, then click Share Project. Choose SAP HANA Repository and click next. Select your workspace and browse to the package you want to put the UI project in (the ui package of the xsdemo project).



To later create another new view in the same project, you would right click the project name, go to New, and select View.

In the index.html file, the location of the UI5 library source needs to be changed; add “/sap/ui5/1/” to before “resources/sap-ui-core.js”. Also, in order to show table information from our OData call in the view, we must add sap.ui.table to the list of libraries. Where it says “data-sap-ui-libs="sap.ui.commons””, add a comma after commons and add “sap.ui.table”. The entire first <script> block in the header will be as follows:

<script src=*"/sap/ui5/1/resources/sap-ui-core.js"*

id=*"sap-ui-bootstrap"*

data-sap-ui-libs=*"sap.ui.commons, sap.ui.table"*

data-sap-ui-theme=*"sap\_goldreflection"* >

</script>

Then, in the view (demo.view.js), we will need to add a model that refers to the OData call. Step by step, here is the code to include in the view:

First an ODataModel needs to be created. The arguments are the absolute path on the server of the xsodata file being called (/xsdemo/services/hello.xsodata), then the formatting of the data (true = JSON, false = atom XML; could be either in this case). There can also be arguments for username and password, which are not necessary here.

**var** oModel = **new** sap.ui.model.odata.ODataModel('/xsdemo/services/hello.xsodata', **false**);

Then, a table needs to be created for the data. The first argument is an ID string, and the second contains any additional settings within the {}. Some common settings are visibleRowCount, firstVisibleRow, rowHeight, and width. If you set a specific number of visible rows, it will show that many, even if some are blank.

**var** oTable = **new** sap.ui.table.Table("helloTable", {visibleRowCount: 5});

Now to add columns, there will first be a TextField control object to use as a template (which we will bind the column name from the OData to). Then, the column needs to actually be added. There is one argument with the settings in brackets; a label can be created to be a header for the column, and the template needs to be set to the control object that was initially created. Some of the other possible settings include width, sortProperty, and resizable (true/false).

oControl = **new** sap.ui.commons.TextField().bindProperty("value","ColID"); oTable.addColumn(**new** sap.ui.table.Column({label:**new** sap.ui.commons.Label({text:"Column ID"}),

template: oControl, sortProperty: "ColID"}));

Then columns need to be created for the other columns of the table.

oControl = **new** sap.ui.commons.TextField().bindProperty("value","Hello");

oTable.addColumn(**new** sap.ui.table.Column({label:**new** sap.ui.commons.Label({text:"Hello"}),

template: oControl, sortProperty: "Hello"}));

oControl = **new** sap.ui.commons.TextField().bindProperty("value","World");

oTable.addColumn(**new** sap.ui.table.Column({label:**new** sap.ui.commons.Label({text:"World"}),

template: oControl, sortProperty: "World"}));

Once the table is complete, it needs to be attached to the model we created initially.

oTable.setModel(oModel);

Then data from the OData file needs to be bound to the table. We also create a sorter object to determine the default sorting of the table.

**var** sort1 = **new** sap.ui.model.Sorter("ColID");

oTable.bindRows("/HelloWorld",sort1);

Then we set the title of the table.

oTable.setTitle("Hello World");

And finally, the table needs to be returned so that it will be inserted into index.html.

**return** oTable;

So, the full code to be included is as follows:

**var** oModel = **new** sap.ui.model.odata.ODataModel('/xsdemo/services/hello.xsodata', **false**);

**var** oTable = **new** sap.ui.table.Table("helloTable", {visibleRowCount: 5});

oControl = **new** sap.ui.commons.TextField().bindProperty("value","ColID");

oTable.addColumn(**new** sap.ui.table.Column({label:**new** sap.ui.commons.Label({text:"Column ID"}),

template: oControl, sortProperty: "ColID"}));

oControl = **new** sap.ui.commons.TextField().bindProperty("value","Hello");

oTable.addColumn(**new** sap.ui.table.Column({label:**new** sap.ui.commons.Label({text:"Hello"}),

template: oControl, sortProperty: "Hello"}));

oControl = **new** sap.ui.commons.TextField().bindProperty("value","World");

oTable.addColumn(**new** sap.ui.table.Column({label:**new** sap.ui.commons.Label({text:"World"}),

template: oControl, sortProperty: "World"}));

oTable.setModel(oModel);

**var** sort1 = **new** sap.ui.model.Sorter("ColID");

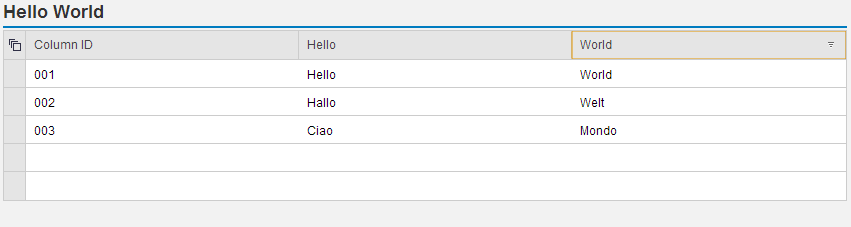
oTable.bindRows("/HelloWorld",sort1);

oTable.setTitle("Hello World");

**return** oTable;

The controller does not need any additional coding for this example. Commented descriptions of the different functions in the controller are included in the file by default.

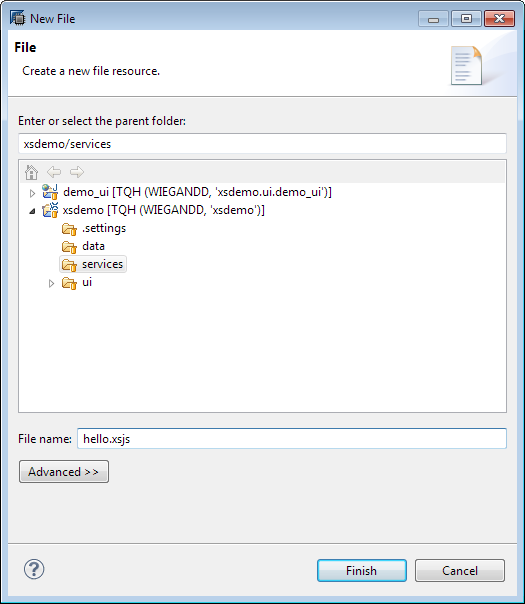
This creates a very simple HTML page with a table showing the data from our Hello World table. Once the controller, view, and index.html page are committed and activated, index.html should be accessible at http://<domain>:80<instance #>/xsdemo/ui/demo\_ui/WebContent/index.html



OData can also be used to create, update, or delete data from the HANA system.

## Server Side JavaScript

An .xsjs file can also be used to access data or perform operations. To show how arguments can be passed to an .xsjs file, we will create a script to multiply two numbers that are included in the URL of the file. Create a new file in the services folder, name it multiply.xsjs, and the text editor will open.



The syntax begins with passing the parameters, word1 and word2, into variables.

**var** word1 = $.request.parameters.get('word1');

**var** word2 = $.request.parameters.get('word2');

Then a variable for the answer should be declared.

**var** answer = '';

Then the words will be concatenated into the answer variable and then put in the body of the page.

answer = word1 + word2;

$.response.setBody(answer);

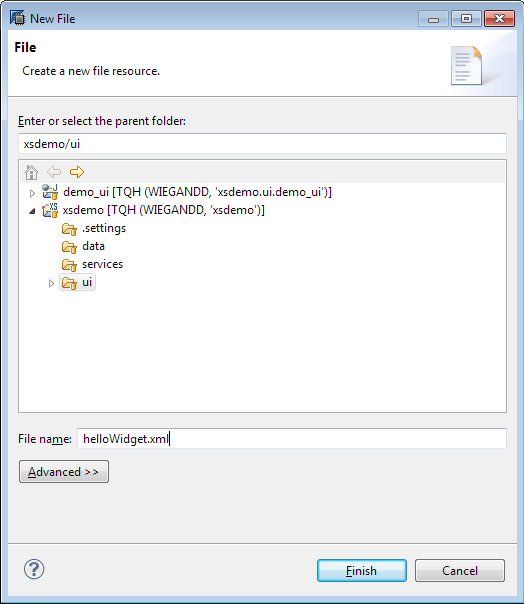
Save and activate that file, and then you can add two numbers via the web server. The format for the URL is http://<domain>:80<instance #>/xsdemo/services/hello.xsjs?word1=Hello&word2=World. The question mark after the URL indicates the beginning of the parameters, and each additional parameter is indicated with an ampersand.

Z:\dwiegand\Documents\HANA Documentation\e2e demo\034_xsjs_result.png

There is a great deal more functionality available for XSJS pages, including reading data through SQL and almost anything that standard JavaScript could do.

## Application Site

We will use the built in SAP HANA Application Site functionality to expose the OData page to users. This entails creating a widget to place the page in, and an application site to hold the widget. First, we need to make an XML specification file for the widget. In the UI folder of the XS project, create a file called helloWidget.xml.



If it opens in an XML editor, close that and open the file with a text editor. To refer to the html file we already created, the following code is used:

<Module>

<ModulePrefs title="Hello World">

<Require feature="sap-context"/>

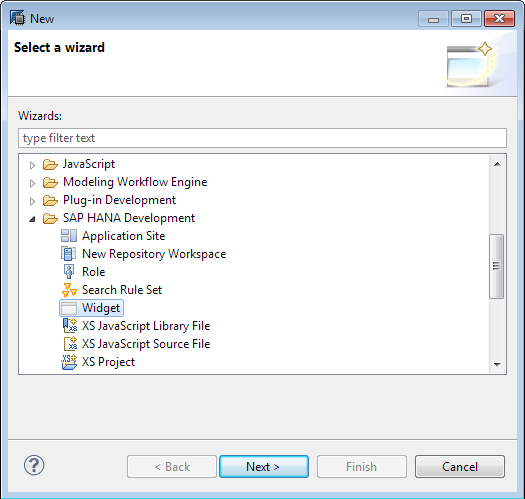
</ModulePrefs>

<Content type="html" href="/xsdemo/ui/demo\_ui/WebContent/index.html">

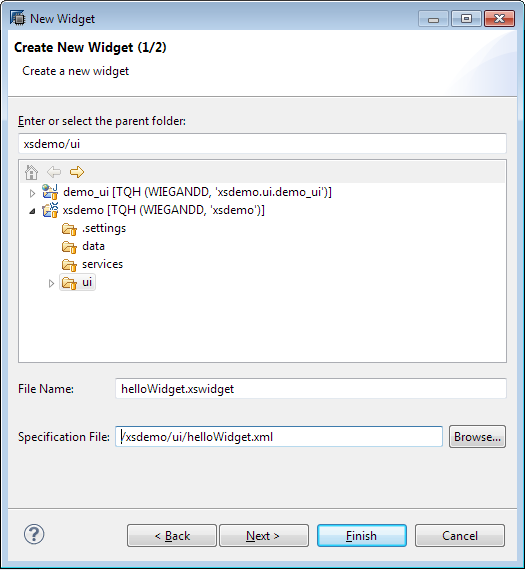
</Content>

</Module>

Once the XML specification is made, committed, and activated; the actual widget must be created. Right click the location (the UI folder), select New, and Other. Widget is under SAP HANA Development; select it and click Next.

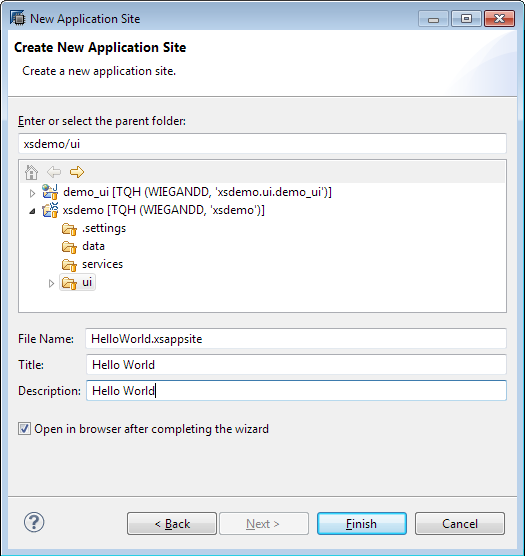


For Specification File, browse to the XML file just created. Put the name (helloWidget) in the File Name box; it will autofill the extension. Click Finish.



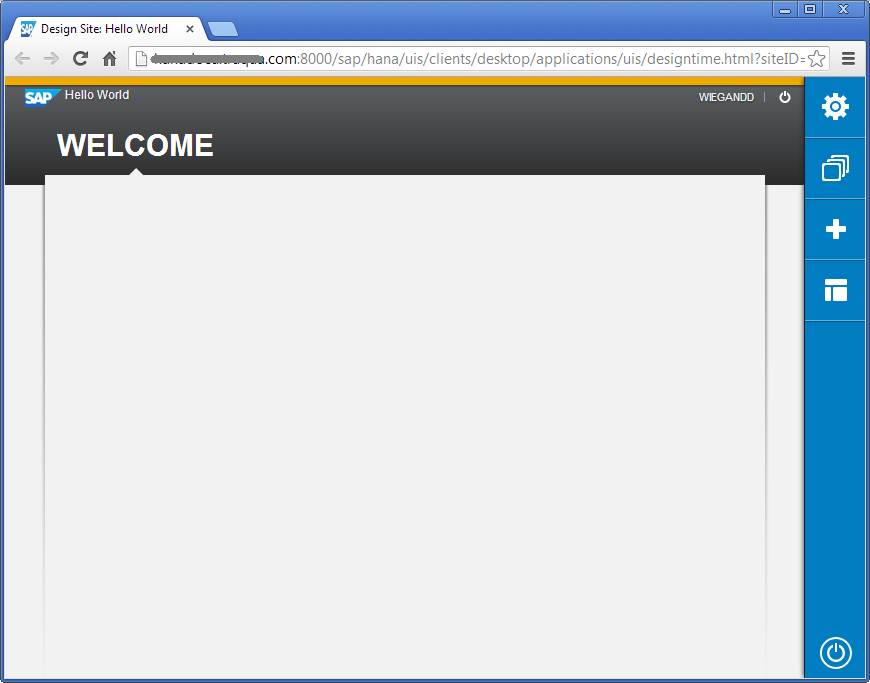
Commit and activate the widget.

Finally, we can create the application site. Right click the UI folder, select “New” -> “Other”, and select Application Site under SAP HANA Development. Choose a name, title, and description, then click Finish.

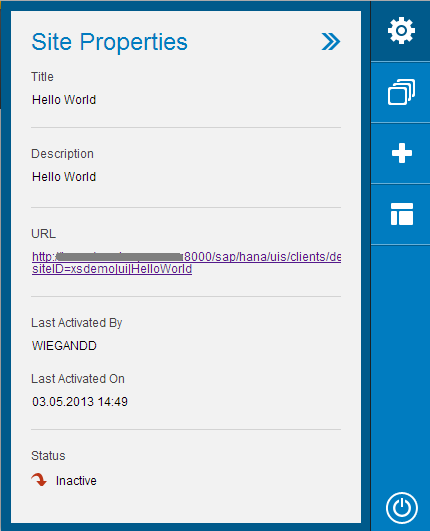


Either check the box labeled “Open in browser after completing the wizard” or, if it cannot be checked, commit the appsite and double click it to open it in a browser after the file is created.

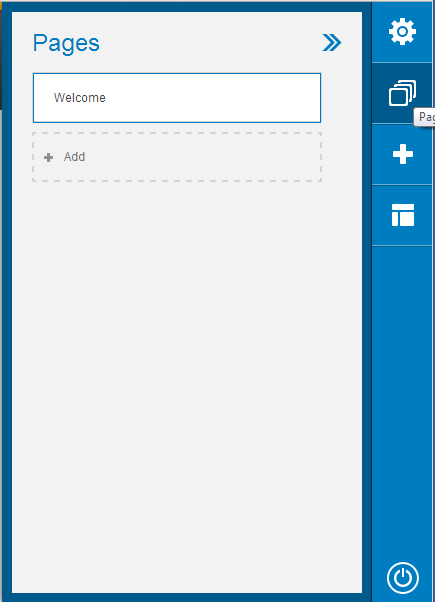
The appsite is edited in a browser. Once you have logged in, the following screen should show:



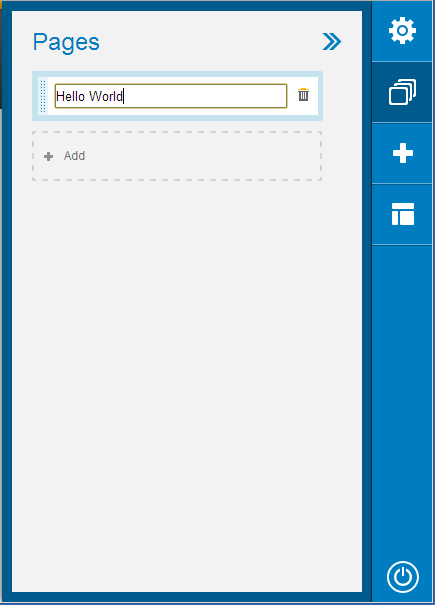
The blue buttons on the bar to the right allow you to control what objects are in the appsite. The gear shows the properties of the site, including the production URL associated with it.



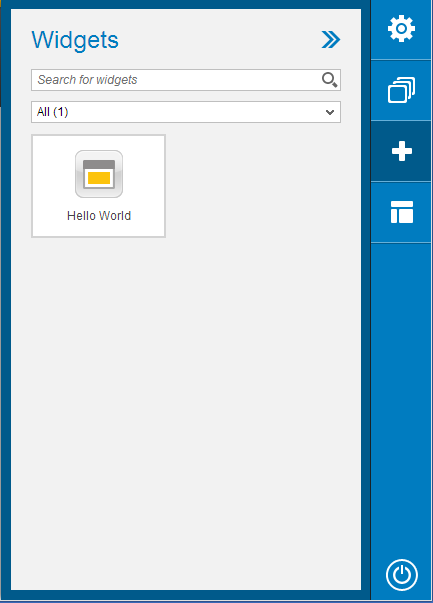
The white squares allow you to add, remove, and edit pages.



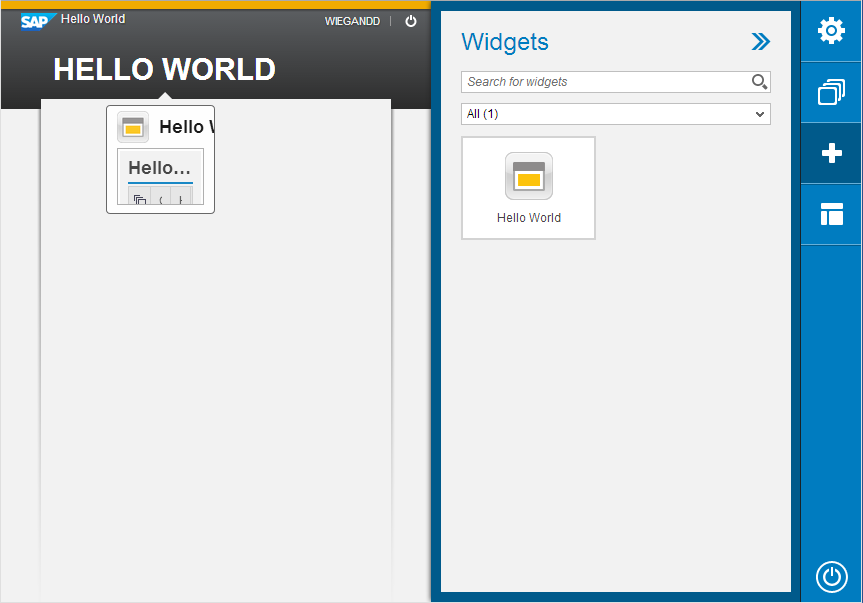
In the pages section, we will change the name of the Welcome page to Hello World. Click the name (Welcome) and it will allow you to type a new name.



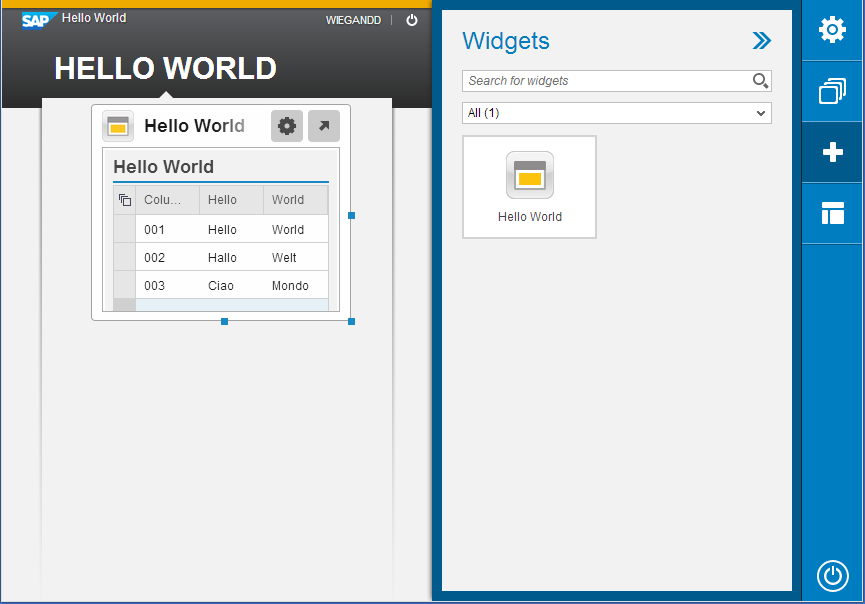
The plus sign allows you to add widgets to pages.



You can drag the Hello World Widget to the page in the main screen of the appsite.



You can resize the widget on the page by moving the mouse over it until blue boxes appear on the right and bottom sides, as well as the corner, and clicking and dragging those boxes. The arrow in the upper right makes the widget fill the page, and the gear allows you to remove the widget.

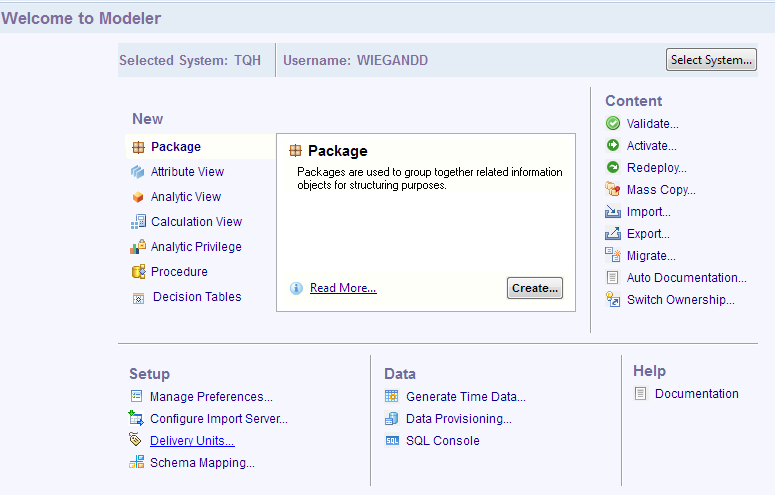


The final side button, layout, changes the size of the navigation bar where the page names are listed.

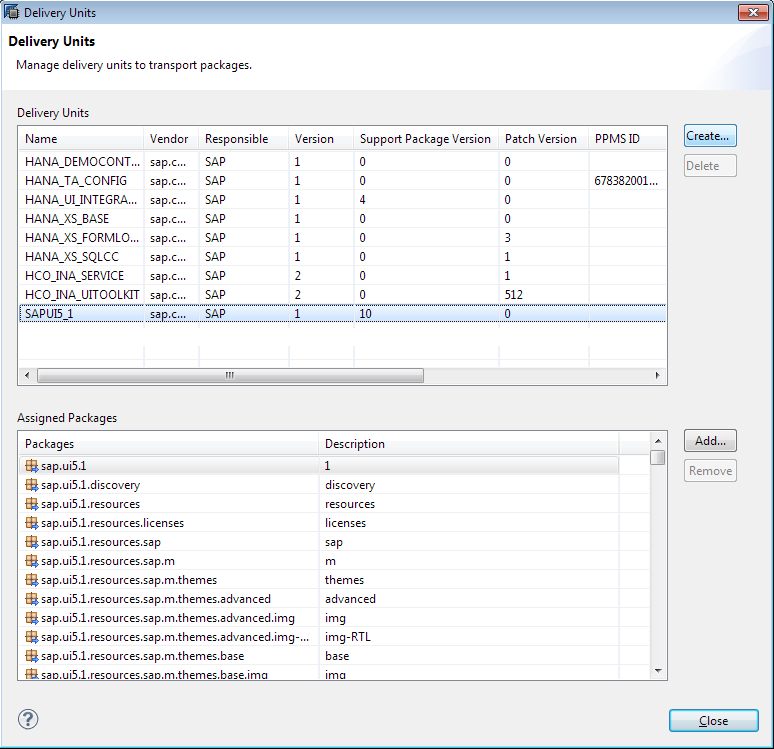
Once the application page has content, go back to HANA Studio and activate it. Return to the browser and get the URL for the page from the Site Properties menu, then you can view your live application site.

## Migrating Content

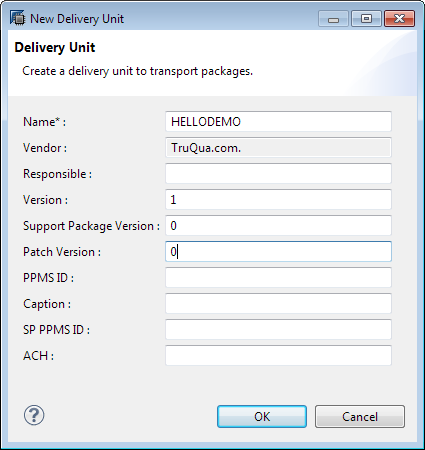
Once the application site is created, you may also want to transport the site to another system, perhaps to a production server from testing. Creating all tables and files within the project makes this very easy. Back in HANA Studio, switch to the Modeler perspective. If the quick launch screen does not come up, go to the Help menu and select Quick Launch.



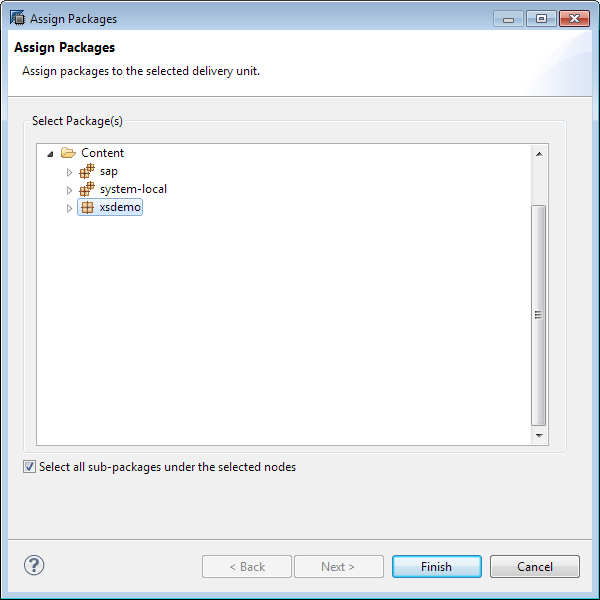
On the quick launch screen, select Delivery Units under Setup. It will bring up a screen showing all delivery units currently in the system; there are a number of default SAP delivery units.



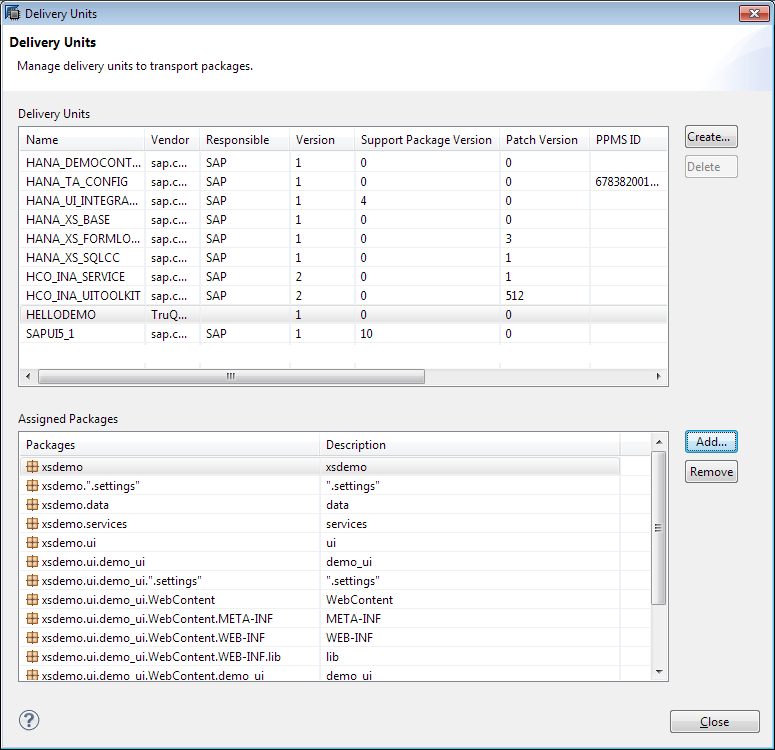
Click the “Create…” button in the upper right.



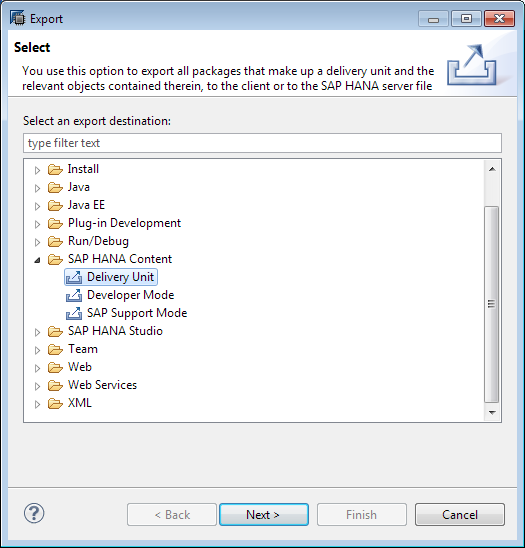
Give the delivery unit a name. The Vendor is autofilled and cannot be changed; everything else is optional. Click OK. The delivery unit will be craeted with no packages assigned; click the “Add…” button to select a package to include in the delivery unit.



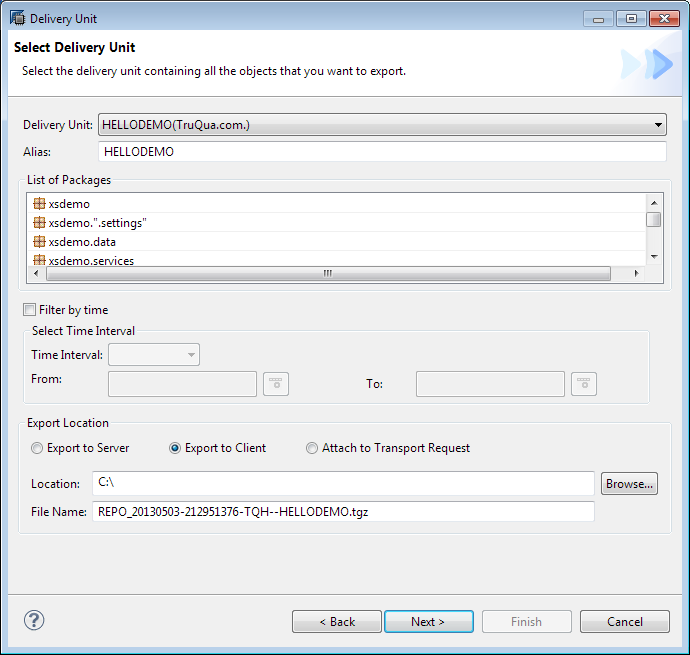
All the packages in the xsdemo package will now be shown as associated packages.



Close the delivery unit view, and select Export under Content. In SAP HANA Content, you will be able to select Delivery Unit.

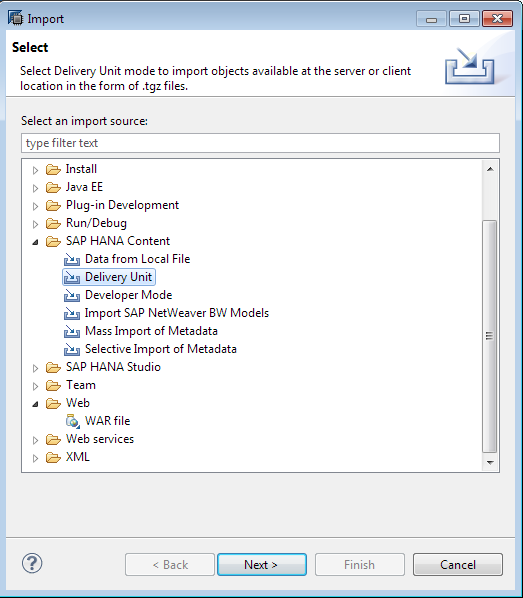


Click next. On the Select Delivery Unit screen, choose the new delivery unit from the dropdown and give it an alias. You can now export it to the current server or export it to the client computer to a location of your choice.

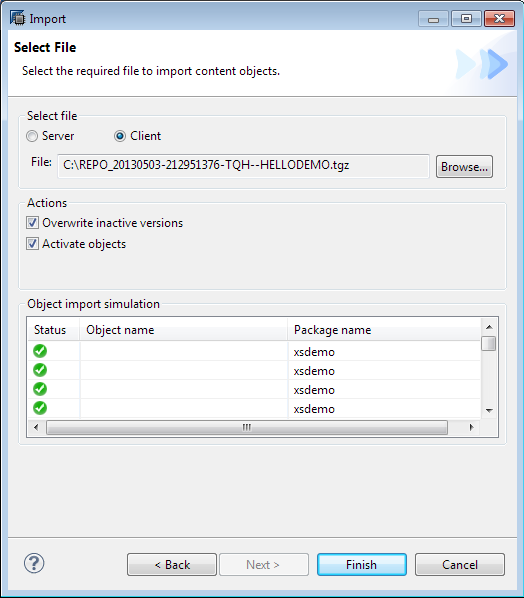


Verify the settings are correct on the next screen, then click finish to begin the export.

Importing delivery units is also very simple. Select Import on the quick launch screen, and choose delivery unit.



On the next screen, if the file is on the server, select Server and choose the file from the dropdown menu. If you exported to the client machine or have a delivery unit exported elsewhere downloaded to your client machine, select client and browse to the location of the file.



Click Finish to finish importing the delivery unit. You can choose to have the imported items overwrite any inactive versions of the files on the server, and to automatically activate them after importing using the checkboxes in the Actions section.

## Documentation Links

[SAP HANA Developer Guide](http://help.sap.com/hana/hana_dev_en.pdf)

[SAP HANA Academy](http://www.saphana.com/community/implement/hana-academy)

[HANA on help.sap.com](http://help.sap.com/hana_appliance/)

[SAP HANA SQLScript Reference](http://help.sap.com/hana/hana_dev_sqlscript_en.pdf)

[Thomas Jung’s Native HANA Development Workshop](http://www.youtube.com/playlist?list=PLoc6uc3ML1JSzl5CQMw24ul0xe9OTG6pl)