Code Jam Mini Edition: First look at Node.js as a Part of SAP HANA Extended Application Services

DEV602

Exercises / Solutions

Thomas Jung / SAP Labs, LLC

Please Note: This is an early look at the new programming model and techniques of HANA Extended Application Services in SPS 11. This is not the final version of the software. This was an early internal build created 3 months before the product release. The purpose of this workshop is to provide an early insight into some of the architectural and programming model changes to XS in SPS 11, but not to give a definite guide to exactly how things will work in the final software.

Contents

[Exercise 1 – Hello World 3](#_Toc427937220)

[Exercise 1 – Solution 4](#_Toc427937221)

[Exercise 1.1: Application Router - Hello World 4](#_Toc427937222)

[Exercise 2 –Database Artifact Development 12](#_Toc427937223)

[Exercise 2 – Solution 13](#_Toc427937224)

[Exercise 2.1: Database Content 13](#_Toc427937225)

[Exercise 3 –XSJS and XSODATA services 18](#_Toc427937226)

[Exercise 3 – Solution 19](#_Toc427937227)

[Exercise 3.1: Node.js Services – XSJS and XSODATA 19](#_Toc427937228)

Exercise 1 – Hello World

Objective

In this first exercise, we will connect to the remote system, start the necessary services, and then create an application router to serve as the application endpoint and proxy all of our services and static HTML content. At the end of this exercise you will be able to connect to your HANA XS server via web browser and see a Hello World message.

Exercise Description

Exercise 1 – Solution

## Exercise 1.1: Application Router - Hello World

|  |  |
| --- | --- |
| Explanation | **Screenshot** |
| 1. Begin by clicking on the **Start** button. Type **remote** into the search window. Then click on **Remote Desktop Connection**. |  |
| 1. Input the IP Address for the server you have been assigned by your instructor. Then press **Connect**. |  |
| 1. Enter the credentials as follows:  User: .\Administrator Password: HANARocks2015  Press **OK** | C:\Users\I809764\AppData\Local\Temp\SNAGHTML4e502ce5.PNG |
| 1. From the desktop, launch the SAP Management Console. We will use this tool just to make sure that your HANA server is already running. |  |
| 1. After you expand the system list, we want to see that all the processes are green. This means your HANA server is up and ready to use. If you don’t see green next to each process, please alert your instructor. |  |
| 1. Next we need to launch XS, advanced. This is the service that offers all the new capabilities of XS in SPS 11. In the final product, this service will be fully integrated into the HANA start up and monitoring framework alongside XS, classic. However because we are working with an early, internal build we need to run it as a standalone service from the Command prompt.   Begin by Launching the Windows Command Prompt from the desktop icon. |  |
| 1. From the Command Prompt, type: xs-start   After a minute or so, the startup process will complete. The XS, advance service is now running and connected to your HANA database.  **Please Note**: Closing this command Prompt window will cause the XS, advanced service to stop. Please just minimize this window and forget about it. You won’t need it any longer in this workshop. |  |
| 1. Please open another Command Prompt window. This is where we will issue all of our command to XS, advanced.   First we need to login. Type the command: xs login  The username is: xsadmin and password is: manager. |  |
| 1. Now we are ready to start doing some development. We will create and edit files directly on our local file system using the Sublime Text editor. The final version of XS, advanced will also feature Web based tooling, but at the time of the preparation of this workshop it wasn’t ready to use yet.  Navigate to the c:\xs\apps folder in Windows Explorer. Here you will already find the completed solution for this workshop. |  |
| 1. Create another folder in c:\xs\apps\ called DEV602. This is the folder where you will perform all of your development tasks. |  |
| 1. We will start development by creating a manifest.yml file in the root of our project folder. This file defines all the parts or micro-services within our application. It is used to deploy the application and configures bindings within our application. . |  |
| 1. Type the following into your manifest.yml file.  We will start by defining one application named DEV602. The application content will be in the sub-folder called web.   We also define external dependencies here.   The destination ui5 will map to where our SAPUI5 libraries are located. This way our web application doesn’t have to hard code the actual path or server port.  We also list uaa as a dependent service. This is because we have user authentication required to access our application.  Save your file. | ---  applications:  - name: DEV602  path: web  env:  destinations: >  [  {"name":"ui5", "url":"http://localhost:8000/sap/ui5/1/"}  ]  services:  - uaa |
| 1. In the DEV602-Solution folder there is a sub-folder called web. Go ahead and copy this into your DEV602 folder. |  |
| 1. This web folder contains the App Router. The App Router is a node.js application which will serve as the entry point for your application. It manages all static HTML resources (in the resources folder) and performs the task of reverse proxy for all other internal services. This way you have a single HTTP endpoint and avoid any CORS issues.  We’ve already placed a simple index.html with “Hello World” in the resources folder. |  |
| 1. The App Router is configured via two files. First is the package.json. package.json is the root file of any node.js application which performs basic configuration. It tells the node runtime which JavaScript it should execute first and it lists any dependent modules. In this case we list the approuter module as the dependency which will cause it to be installed during deployment. We also tell node to launch the approuter.js upon startup of this application. |  |
| 1. The other configuration file is the xs-app.json. In this file we map the routes to destinations we defined in the manifest.yml. We have three routes already defined for you. One will point back to the SAPUI5 destination you’ve already created. The other two (xsjs and xsodata) are for use later once we’ve created those services. |  |
| 1. Our initial development is done and we are ready to deploy our application onto the XS, advanced server. Return to the Command Prompt where you are logged into xs.  Change directory to c:\xs\apps\dev602\ |  |
| 1. Issue the command  xs push  This will upload your development objects to the server, package them along with any dependent modules and with the node.js runtime. All of this content then gets deployed onto the server and assigned a port.   If successful, you should see the application reach the state RUNNING and see the HTTP port to which it has been assigned. |  |
| 1. Open Chrome and go to [http://localhost:<your](http://localhost:%3cyour) port from previous step>. |  |
| 1. You should see a login screen. Use the XS, advanced application user: xsadmin and the password: manger to login.   Authentication at the XS level is no longer done against HANA database users. Instead XS, advanced uses separate application users which generally come from an external user store. XS, advanced then uses a container service technical user to connect to the HANA database. |  |
| 1. After successful authentication, you should see your index.html with the Hello World message. |  |
| 1. Congratulations! You just wrote your first XS, advanced Node.js application. |  |

Exercise 2 –Database Artifact Development

Objective

In this exercise, we will continue to develop our application. We will data database table and stored procedure definitions to our applications. We will see how we use a node.js application to deploy these database artifacts using the new container-based, schema-less HDI (HANA Deployment Infrastructure) concepts.

Exercise Description

* Database Tables via HDBCDS
* Stored Procedures via HDBPROCEDURE
* Initial table data load via CSV
* Deploy to HANA via HDI

Exercise 2 – Solution

## Exercise 2.1: Database Content

|  |  |
| --- | --- |
| Explanation | **Screenshot** |
| 1. New to HANA in SPS 11 is the concept of HDI – HANA Deployment Infrastructure. The goal of HDI is manage database artifacts from design time objects but in a way that allows multiple “branches” of the same core objects to be used on the same HANA database at the same time.  Therefore all development objects are created into a container. The container in turn dynamically generates the Schema, Database User who owns all objects, and a password for that database user. XS, advanced applications then only need access to the container and never need to know the actual Schema, technical user, or password. All of that information is stored within the container definition. . |  |
| 1. From our command prompt we want to issue the command: xs create-service hana hdi-shared dev602-hdi-container.  This will create a container for our application named dev602-hdi-container of type hana, hdi.. |  |
| 1. Now edit your manifest.yml file in the DEV602 directory again. Add a second application definition named DEV602-db. It should point to the db folder. It doesn’t need to be accessible via HTTP, so use the no-route: true option. Finally bind this application to the HDI container you created in the previous step by listing it as a dependent service. | - name: DEV602-db  path: db  memory: 128M  no-route: true  services:  - dev602-hdi-container |
| 1. To save you some coding time, we’ve already prepared the database artifacts in the db folder of DEV602-Solution. Copy this folder into your DEV602 folder. |  |
| 1. In the root of the db folder you just copied there is also a package.json file just like in the web folder. This file serves the same purpose, except instead of telling node.js how to start the App Router, it instead starts the HDI Deployment application. |  |
| 1. The db/src folder is where your actual database development objects belong. There are two configuration files in the root of this folder. The .hdiconfig file maps the file extensions to the specific server side activation plug-ins. This way you can choose any file names you wish to use as long as you map them to the correct plug-ins. However we will use the default mappings for now.   The .hdinamespace file configures the package namespace for your development objects. As we no longer use the HANA Repository to hold design time objects, this file provides the same service as the folder structure in the Repository used to. |  |
| 1. In the data folder you will see several development artifacts. hdbcs is the new file extension replacing hdbdd. It contains the table and view definitions. We are creating a simple Purchase Order Header and Item data model.   The syntax is the same as CDS-based development objects previously. |  |
| 1. We also have the hdbtabledata development object. This is the replacement for the old hdbti development object. Although the syntax of this object is new, the purpose is the same – to allow the loading of initial data from CSV files it target tables during their creation. |  |
| 1. In the procedures folder we have an hdbprocedure file. The syntax for stored procedures hasn’t changed from previous levels of HANA. |  |
| 1. Now that we have our database development objects, we are ready to push the application again. This time pushing it will execute a node.js application which will call over to HANA and deploy these database artifacts into their container.   From the command prompt issue the command: xs push DEV602-db  We added the DEV602-db part to the command this time so that only this one of the two applications in the manifest.yml will be deployed. There is no reason to re-deploy the complete manifest file because we’ve not made any changes to the web content yet. |  |
| 1. At the end of deploying you should see that the DEV602-db is running. This doesn’t mean it’s done yet. It has just started the process of creating the database objects. To see the progress of their creation you must look at the logs for this application. |  |
| 1. Issue the command: xs logs DEV602-db –recent  This will show you the logs for this application. You want to see something like the screen shot in the output. If you don’t see a deployment to a container yet, it might not be finished and you will need to re-issue the logs command. |  |
| 1. That long GUID that is the container name in the log is actually the database schema in HANA. Remember the first few letters of that container (your value will be different than what is shown here) and then open the HANA Studio.   From the HANA Studio, expand the catalog folder and then find the Schema with the same name as the container showed in your log. There will be a very long list of schemas, so it’s probably best to use the filter feature. |  |
| 1. You should then be able to see the development artifacts successfully deployed into your container schema. However, it’s perfectly normal that even the SYSTEM user won’t have authorization to query or execute any of these objects. Only the technical user created as part of the container has access to these objects. |  |

Exercise 3 –XSJS and XSODATA services

Objective

For this exercise we will now build the XSJS and XSODATA services used to expose our data model to the user interface. Although XS, advanced runs on node.js, SAP has added modules to node.js to provide XSJS and XSODATA emulation. Therefore you can use the same programming model and much of the same APIs from XS, classic even within this new environment. .

Exercise Description

* Node.js XSJS Bootstap
* XSJS Services
* XSJS Services which branch back into Node.js Modules
* XSODATA Services

Exercise 3 – Solution

## Exercise 3.1: Node.js Services – XSJS and XSODATA

|  |  |
| --- | --- |
| Explanation | **Screenshot** |
| 1. Like the previous exercises, we will start by extending the manifest.yml. Add a new application section named DEV602-js.It should point to a js sub-folder and will run on port 3005. It should also be bound to the dev602-hdi-container we created in the previous exercise. This will allow our node.js coding to connect to HANA and read/execute the database artifacts we created previously without coding any destination information. | - name: DEV602-js  path: js  port: 3005  services:  - dev602-hdi-container  - uaa |
| 1. We also need to expose these new XS services through the application router. Therefore please return to the DEV602 entry in the manifest.yml and add a second destination for nodejs that points to port 3005 (the same port we just decided to run our node.js services on). | {"name":"nodejs", "url":"http://localhost:3005"} |
| 1. If you remember back to exercise 1 – we had two additional routes in the xs-app.json of the App Router. Now you see why we did this previously. This is where we are configuring that any file request with the extension .xsjs or .xsodata should be rerouted internally to the nodejs destination that we just defined in the manifest.yml. |  |
| 1. Like in the previous exercises, we will copy the **js** folder from the DEV602-Solution into our DEV602 folder. |  |
| 1. Like the other applications, this one also starts with a package.json file. Different this time is the fact that the startup script is not an SAP provided central node application, but one that we’ve created. |  |
| 1. This xsjs.js is the node.js bootstap for XSJS compatibility mode. It uses the SAP provided xsjs module and starts it with a few basic parameters. However remember all the HANA database connectivity options come from the HDI container which we bound to this service via the manifest.yml. |  |
| 1. The src folder then contains xsjs and xsodata files that for the most part have pretty much just been copied from a pre-SPS 11 HANA system. We will discuss what each does in a little more detail after we deploy them and can execute them. |  |
| 1. We can now deploy the application. This time we will do a complete push because we have changes to 2 of the 3 applications in the manifest.   xs push  You will see all three application deploy/redeploy – which could take a minute or two. |  |
| 1. If you need to refresh your memory as to which port your App Router is running on, you can issue the command: xs app DEV602 |  |
| 1. Open your Chrome web browser to localhost and this port once again. If prompted for authorization again, its xsadmin/manager |  |
| 1. We can now change the URL to test the routes to our new XSJS services. For instance http://localhost:<port>/xsjs/hello.xsjs |  |
| 1. /xsjs/hdb.xsjs reads data from our new Purchase Order table we created in HANA in the previous exercise and exports it as an Excel text file. |  |
| 1. /xsjs/os.xsjs shows how you can call node.js modules even from XSJS code. |  |
| 1. /xsjs/procedures.xsjs shows calling a stored procedure that we created in the previous exercise via XSJS and outputting the results as JSON. |  |
| 1. /xsodata/purchaseOrder.xsodata gives you access to a full OData service for the Purchase Order header and item tables we created in the previous exercise. |  |
| 1. Finally if you want to see how all the pieces fit together, run [http://localhost:<your](http://localhost:%3cyour) port>/odataTest.html  This is a Fiori application that displays the OData service results in two tables and also calls the XSJS service to download the data to Excel. |  |



© 2015 SAP SE or an SAP affiliate company. All rights reserved.

No part of this publication may be reproduced or transmitted in any form or for any purpose without the express permission of SAP SE or an SAP affiliate company. SAP and other SAP products and services mentioned herein as well as their respective logos are trademarks or registered trademarks of SAP SE (or an SAP affiliate company) in Germany and other countries.   
Please see [http://www.sap.com/corporate-en/legal/copyright/index.epx#trademark](http://www.sap.com/corporate-en/legal/copyright/index.epx%23trademark) for additional trademark information and notices.