SAP HANA Cloud Platform End-to-End-Development Scenarios

Getting Started

Develop Your First SAP HANA Native Application on SAP HANA Cloud Platform Using the SAP HANA Web-based Development Workbench

Version 1.0 | June 2014 | Bertram Ganz, Jens Glander, Monika Kaiser, SAP AG

NOTE April 2016: THIS TUTORIAL IS OUTDATED AND WILL NOT BE MAINTAINED ANY MORE

Table of Contents

Source Code

Figures

1 Introduction

1.1 What Do You Get Here?

This guide describes step by step how you create your first HANA native application with a shared SAP HANA database instance on the SAP HANA Cloud Platform trial landscape. You develop a *PersonsList* web application end-to-end: from HANA database table definition to application logic implementation up to the SAPUI5 user interface on frontend side.

- **Platform**: You build and run your application within a trial developer account on the SAP HANA Cloud Platform
- **Database**: A shared SAP HANA database is used to create one column-based table to store Person data
- **Runtime**: Your application runs on SAP HANA extended application services (shortly *SAP HANA XS,* see Glossary), a small footprint application server, including a web server and basis for an application development platform inside SAP HANA.
- **Design time**: As development tool you purely use the SAP HANA Web-based Development Workbench (shortly *HANA Web IDE*) that allows to develop the whole application in a browser without the need to install any development tools (like the Eclipse IDE) on your client.
- **Frontend**: You will use the UI development toolkit for HTML5 (shortly *SAPUI5*), SAP's HTML5 library that enables to build responsive business applications for all devices.

1.2 Intended Audience

This guide is intended for all developers and programmers who may be new to the SAP HANA Cloud Platform and/or SAP HANA native development based on *SAP HANA extended application services* (also known as *SAP HANA XS*).

1.3 Applicable Releases

SAP HANA SPS 07 available as Trial SAP HANA instance on SAP HANA Cloud Platform where everyone can get free access to. The described web-based HANA XS development scenario works also on a corresponding on-premise SAP HANA instance, where you should better follow the on-premise tutorial version.

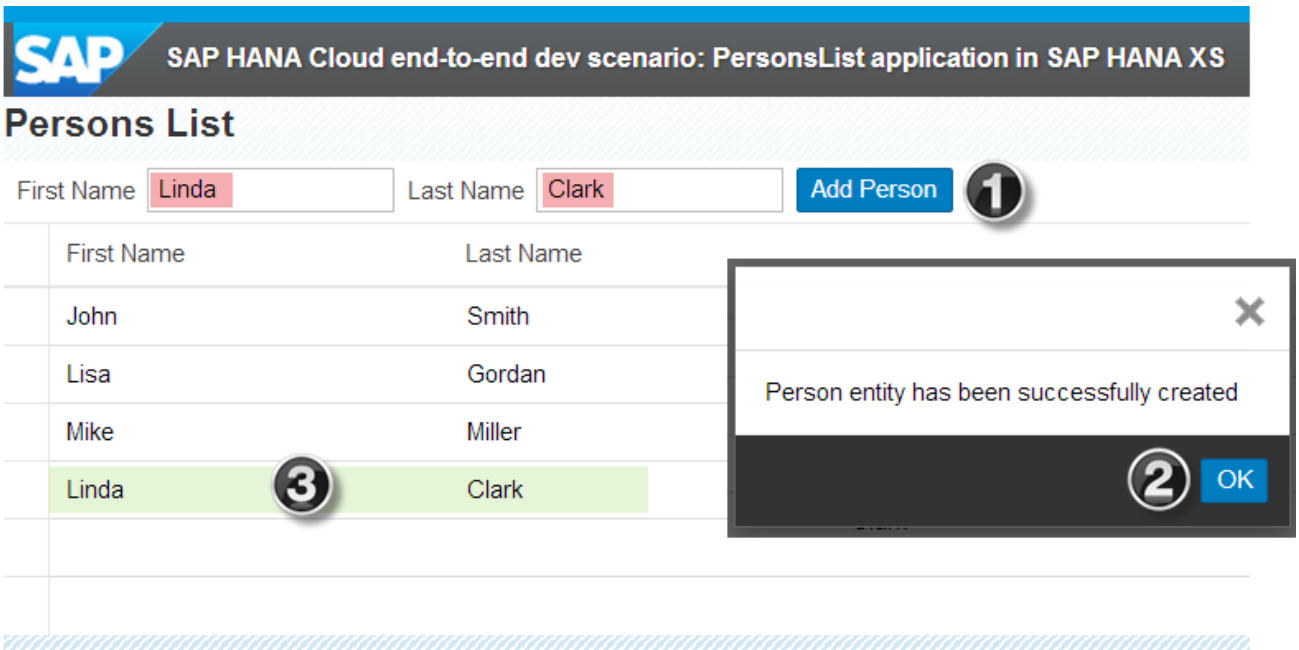
1.4 End-2-End Development Scenario Overview

The application frontend you build in this tutorial looks very simple. Person entries that are retrieved from the SAP HANA database are displayed in a table control:

(1) You can enter first and last name of a new Person entry and write it to the backend with an input form inside the table toolbar.

(2) On creation of a new Person entry a success message gets displayed.

(3) And the new entry gets instantly displayed in the table UI.

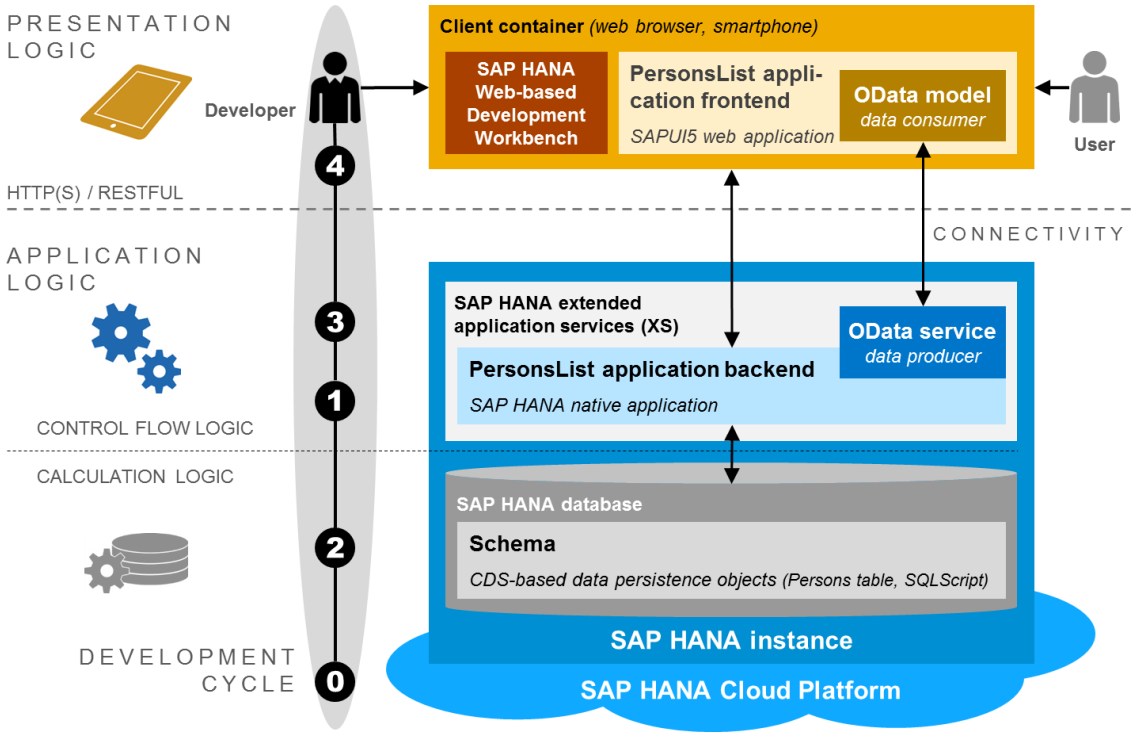


1.5 Development Steps in the Corresponding HANA Layers

Figure 1 illustrates the whole development scenario in a simple overview. The development process applied in tutorial has been divided in five main development steps:

(0) You start with account sign-up and HANA database creation on SAP HANA Cloud Platform
(1) Create a bare-bone SAP HANA XS application
(2) Define a CDS-based data persistence object in the SAP HANA database
(3) Develop the application logic: create a OData-service to read and write person table records from and to the HANA database
(4) Develop application frontend side: implement of a simple SAPUI5 user interface

Figure 1: Overview of the development steps for the cloud-based SAP HANA native PersonsList application



The main development steps and the corresponding sections of the end-to-end tutorial

Step	Section	What You will Learn
Step 0	**Prerequisites for Web-based Development: account setup**	• How to sign up for a free developer account on SAP HANA Cloud Platform (trial account) • How to create a new trial instance of the SAP HANA database within your developer account • How to start the SAP HANA Web-based Development Workbench from your SAP HANA Cloud Platform cockpit.
Step 1	**Create new SAP HANA native application within SAP HANA Web-based Development Workbench**	• Start the SAP HANA Web-based Development Workbench from SAP HANA Cloud Platform cockpit • Create blank SAP HANA XS application • Run blank SAP HANA XS application in web browser • Set up the application package structure in SAP HANA repository
Step 2	**Set up the persistence model in SAP HANA database**	• Create sub-package *data* • Define data type *Person* within CDS document • Load initial mock data from .csv file into database table • Define new user role and grant it to your user
Step 3	**Build the application backend with SAP HANA extended application services**	• Expose the Person database entity by means of OData service • Implement application logic to write new records into Persons table (with OData modification exit that calls a SQLScript procedure)
Step 4	**Build & run the SAPUI5 application frontend UI consuming an OData service to read and write Persons data**	• How to build the SAPUI5 frontend with a simple input form to add new Person entries and a table control to list stored ones. • How to consume an OData service with read and write access in SAPUI5.

1.6 How to Run This Tutorial

For executing this tutorial successfully we strongly recommend to read the following list of hints:

Hint 1: Which Web Browser Should You Use?

The HANA Web IDE tutorial has been proven to work best with **Google Chrome browser** in the newest available version. We therefore recommend using Google Chrome when working through this tutorial. Nevertheless the HANA Web IDE is also supported for latest Mozilla Firefox and Microsoft Internet Explorer (10+) browsers.

Hint 2: First set-up Your Developer Account

Before you can start with the web-based HANA XS application development **you need to setup your free developer account** on SAP HANA Cloud Platform as described in detail in chapter 2: *Prerequisites of Web-based Development: Account Setup.*

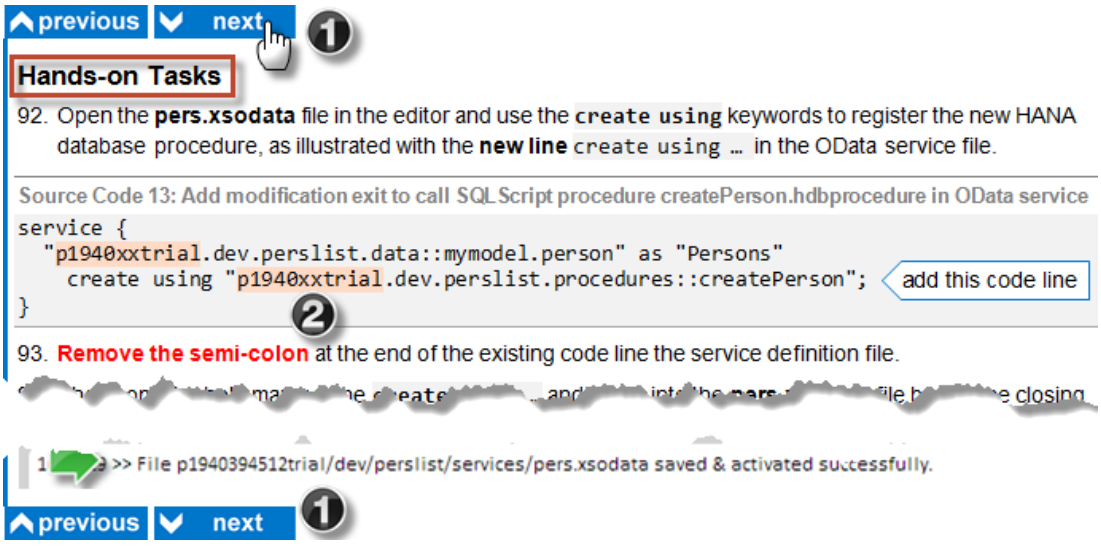
Hint 3: For Quick Success just Follow the Hands-On Tasks

Many developers don't like to read lengthy tutorials and often just want to execute the relevant steps to get the tutorial application running. In case you want to process the whole tutorial for fast success you could only follow the "**Hands-on Tasks**" sections that are highlighted with a **blue border at the left side.**

To jump from one hands-on task to another click the **blue "previous"/"next" link buttons** (see Screenshot 1, item 1) or search for "Hands-on Tasks". For just quickly building the HANA XS Getting Started application of this tutorial it is enough to execute the steps inside the Hands-on Tasks section.

To succeed even faster you can run the **interactive online version of this PDF tutorial**; see SCN blog Try out Web-based HANA XS Development on SAP HANA Cloud Platform for more details ….

Screenshot 1: Hands-on Task section highlighted with blue border line and navigation links



NOTE: For a deeper understanding of SAP HANA XS application development on SAP HANA Cloud Platform we recommend reading the complete tutorial. Also see the Glossary with terms and short terms used in the tutorial.

Hint 4: Copy & Paste File and Package Names

During the tutorial you need to create many files and packages. Those files and package names that can be copied and pasted without editing change are **highlighted with a blue color** like this example: `createPerson.hdbprocedure`. Avoid typos by entering these blue highlighted strings with copy and paste function.

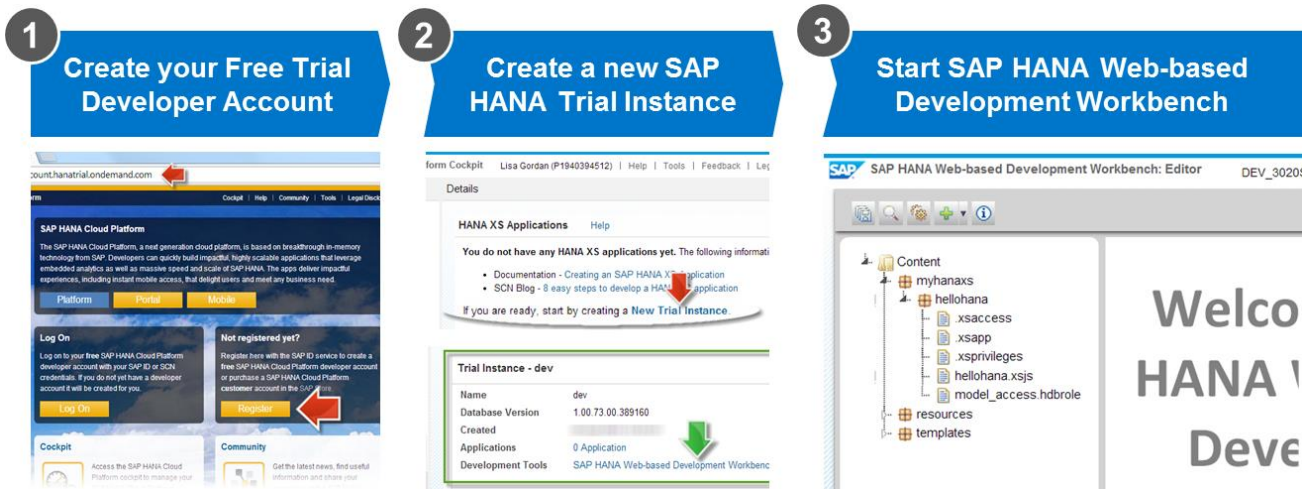
Hint 5: Copy Source Code and Replace User Name

During the tutorial you need to copy and paste source code where user specific data is contained (mainly the namespaces of your HANA artifacts will contain you user name).

When copying source code **you need to replace** the **highlighted, user-specific red color string** `p1940xxtrial` with your own user name (e.g. `p1940394512`, see Screenshot 1, bullet item 2).

2 Prerequisites of Web-based Development: Account Setup

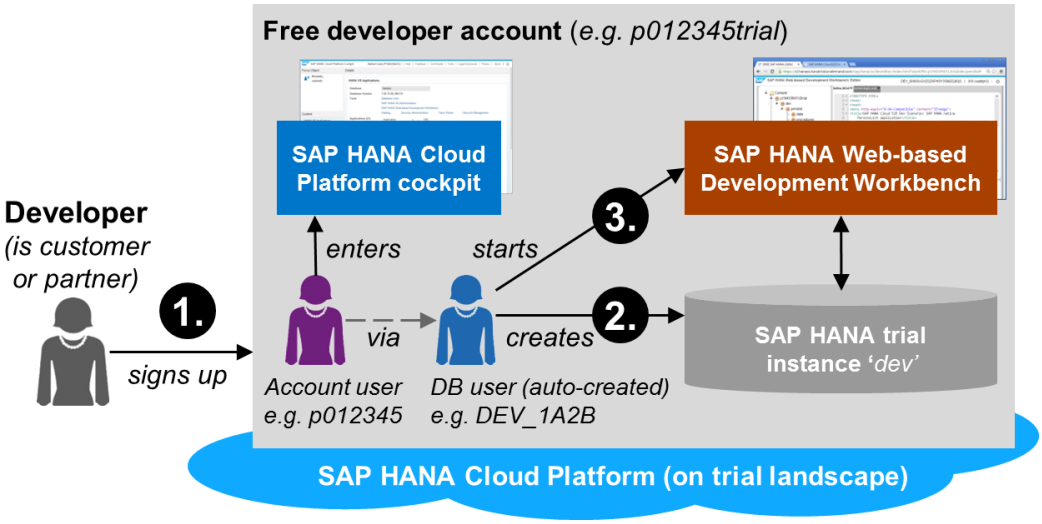
To start HANA native development within the HANA Web IDE you need to follow some setup steps before: firstly the developer account creation on SAP HANA Cloud Platform trial landscape and secondly the SAP HANA database creation.



The following sections 2.1, 2.2 and 2.3 describe how you as a developer …

1. create your **free trial developer account on SAP HANA Cloud Platform,**
2. login to the SAP HANA Cloud Platform cockpit with your trial account user and create a new **instance of the SAP HANA database,**
3. start the **SAP HANA Web-based Development Workbench** (in short HANA Web IDE).

Figure 2: Three steps how a developer starts HANA XS development on a *trial* SAP HANA instance by means of SAP HANA Web-based Development Workbench



NOTE: In a free developer account on SAP HANA Cloud Platform trial landscape you do not need to create a **HANA database user** on your own. For sake of simplicity and security a new **HANA database user** that is required for HANA native development is *auto-created* for you by the SAP HANA Cloud Platform when you create a new developer account. This HANA database user has all pre-defined privileges that are needed for HANA native development in the trial SAP HANA Cloud Platform landscape (with some specifics imposed by user isolation requirements). In a productive SAP HANA Cloud Platform account or in an on-premise SAP HANA system you need to create and configure the HANA database user manually.

Further information:

SAP HANA Cloud Platform Documentation: Using a Trial SAP HANA Instance

2.1 Create your Free Trial Developer Account

For developing and running Web applications on SAP HANA Cloud Platform you have to register once for a SAP HANA Cloud Developer Account. A developer account offers you access to the trial landscape for an unlimited period and is free of charge. You are restricted to one developer account.

Start Hands-on

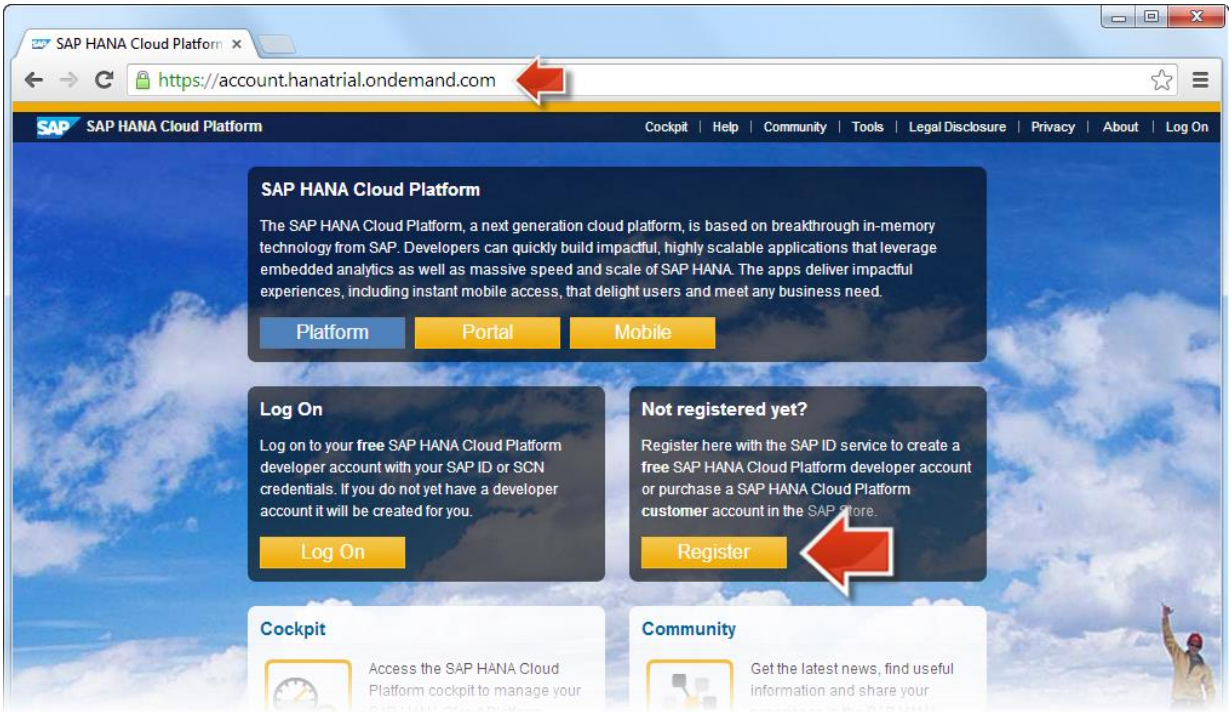
Prerequisites

Depending on whether or not you already have an SAP ID user, the creation of a new developer account on SAP HANA Cloud Platform differs.

Hands-on Tasks

Execute the following steps to register for your own SAP HANA Cloud Developer Account.

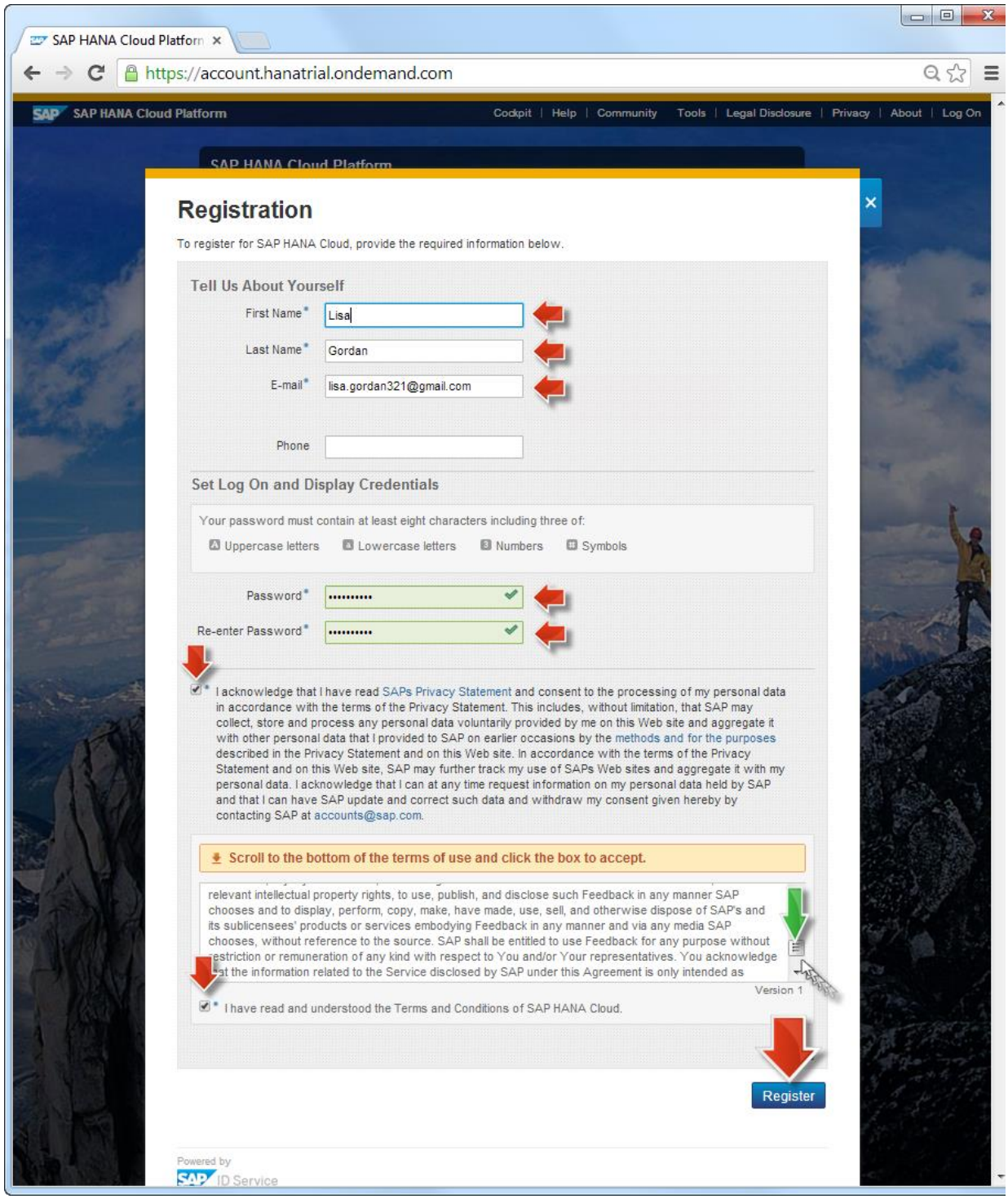
1. Open the SAP HANA Cloud Platform landing page https://account.hanatrial.ondemand.com in your Google Chrome web browser.
2. If you do not yet have an SAP ID user choose **Register** to open a registration form. If you already have an SAP ID user in place follow the below NOTE description.



NOTE: If you already have an own SAP ID user in place you can directly log on to your free SAP HANA Cloud Platform developer account with your SAP ID or SCN credentials. If you do not yet have a developer account it will be created for you. Carry out the following steps:

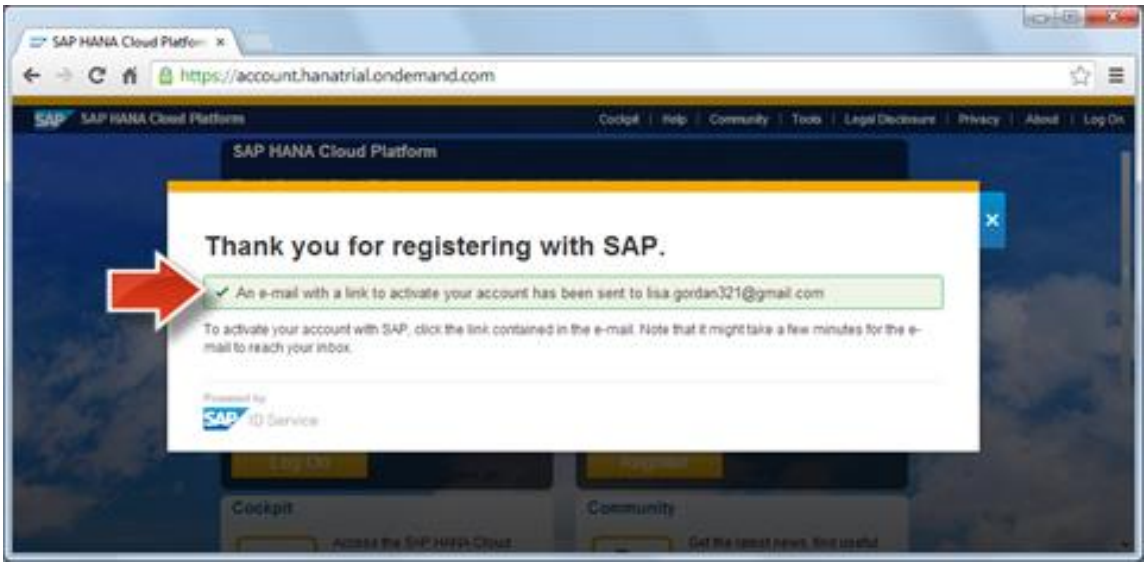
- Click **Log On** and sign in with your SAP ID or SCN credentials.
- Read and accept the SAP HANA Cloud Developer Edition License Agreement.
- Skip the next tutorial steps and move to the Result section of this hands-on task.

3. On the registration form enter the corresponding fields, read the *Terms of use* and then select the option that you have read and understand the Terms and Conditions of SAP HANA Cloud.

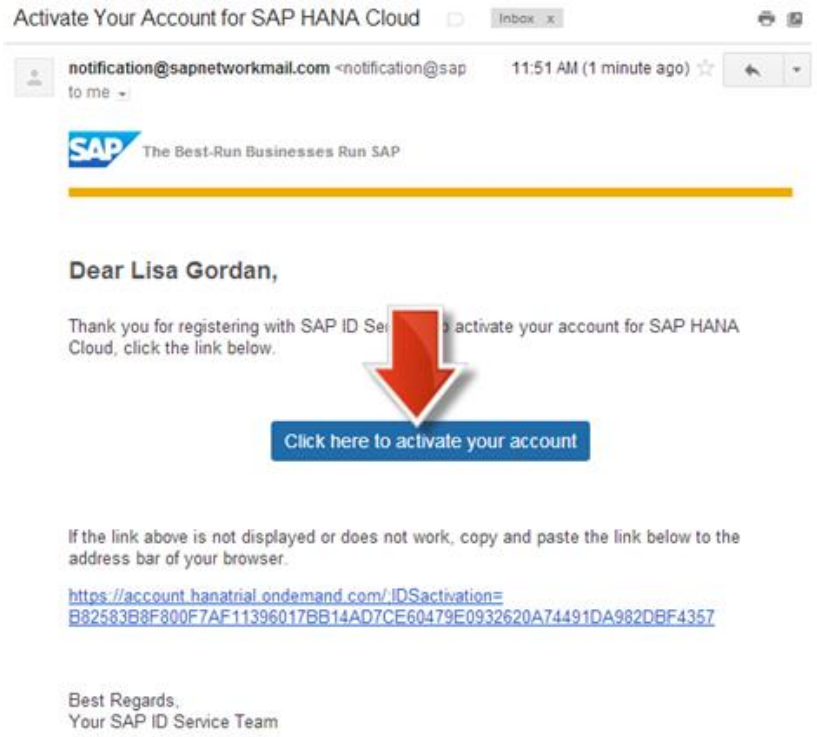


4. Press the **Register** button.

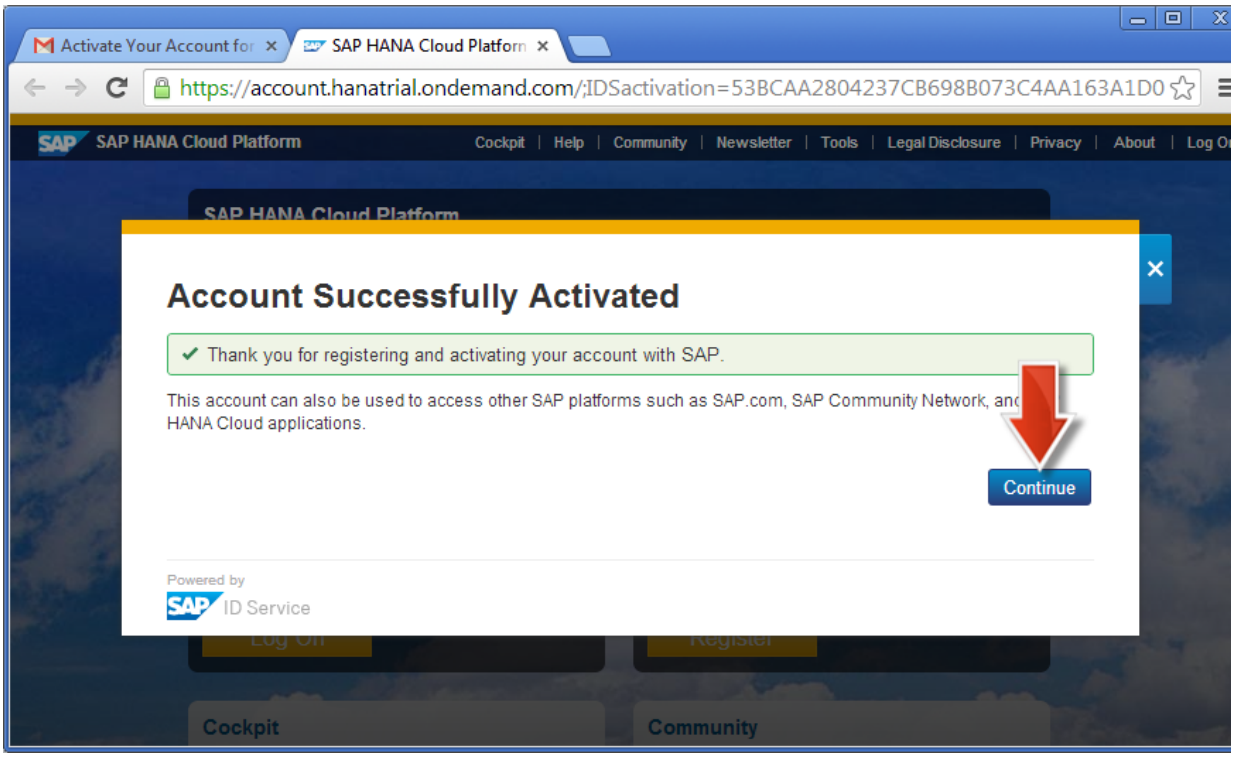
5. A registration confirmation message appears. An e-mail which contains the activation link of your cloud developer account will be sent to you.



6. Open this mail which was sent to your e-mail account

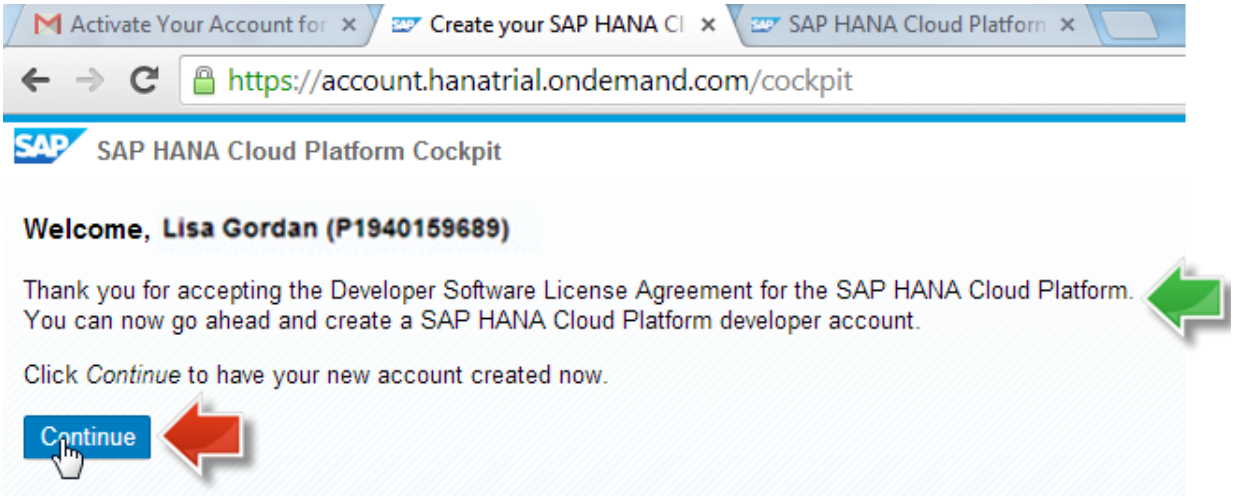


7. Choose the **Click here to activate your account** button and an *Account Successfully Activated message* will appear.



8. Choose **Continue** to launch the SAP HANA Cloud Cockpit of your newly created developer account.

9. As you launch your SAP HANA Cloud Platform Cockpit the first time a welcome page thanks you for accepting the Developer Software License Agreement for the SAP HANA Cloud Platform.



10. Confirm the SAP HANA Cloud Platform Cockpit welcome message by clicking **Continue.**

Result

Finally the **SAP HANA Cloud Platform Cockpit** tool of your newly created SAP HANA Cloud Platform ***Trial Developer Account*** opens with its Account Dashboard.

You have created your SAP HANA trial developer account data:

- Trial account name : **`p0123456trial`**
  (in the entire tutorial we use example account `p1940394512trial`)
- Trial account user : **`p0123456`**
  (in the entire tutorial we use example user `p1940394512`)
- Trial account password : **********

NOTE: With your trial account data you can directly log on to your developer account cockpit via URL https://account.hanatrial.ondemand.com/cockpit .

Read more: SAP HANA Cloud Platform Documentation > Sign Up for a Developer Account

2.2 Create a new SAP HANA Trial Instance

Execute the following steps to create a new trial instance of the SAP HANA database on your trial SAP HANA Cloud Platform account.

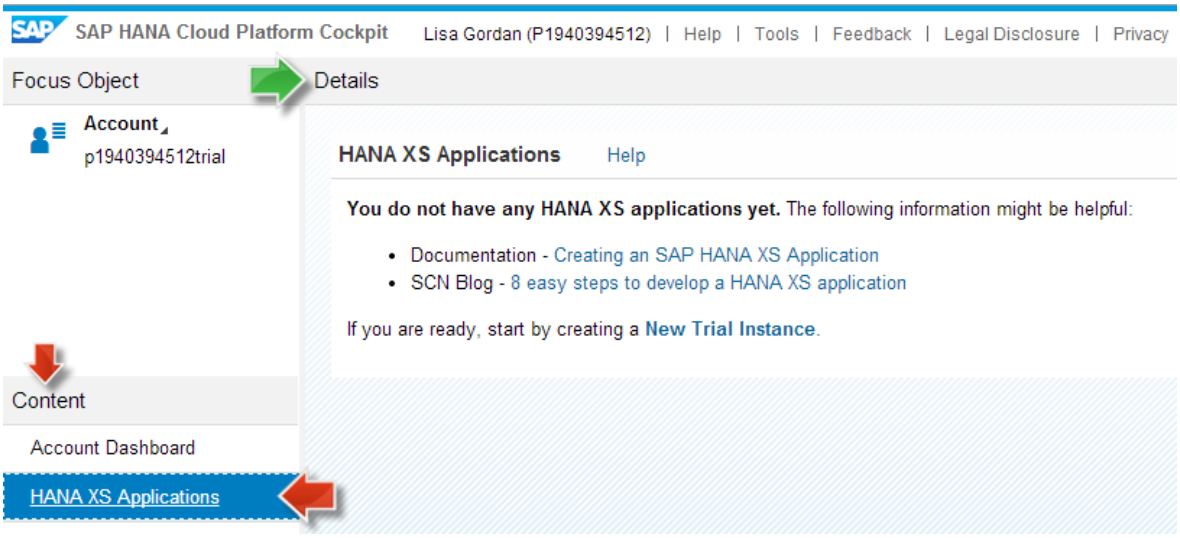
previous next

Prerequisites

- You have your free developer trial account data at hand.

Hands-on Tasks

11. **Open** your SAP HANA Cloud Platform cockpit: https://account.hanatrial.ondemand.com/cockpit.
    If the Log on page is displayed then log on with your E-mail (or user ID) and password you specified during your trial account registration.
12. In **Content** area select *HANA XS Applications* so that the corresponding Details page is displayed.



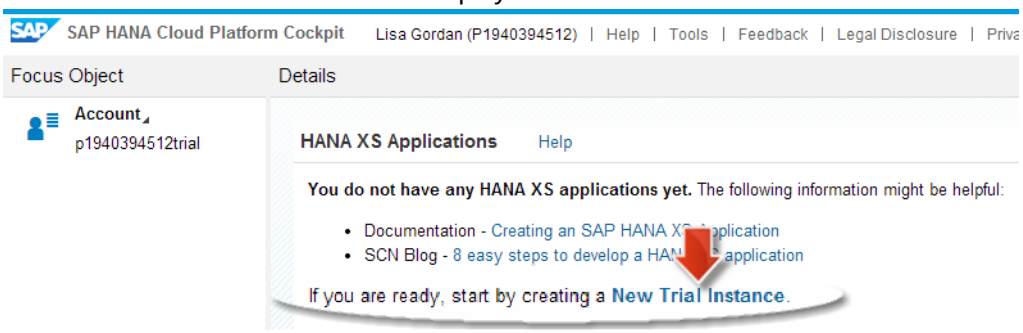
13. Execute this step only if you have already one or more Trial Instances of the SAP HANA database (e.g. you have executed already the Eclipse-based HelloWorld SAP HANA XS tutorial before). In this case we recommend that you delete your existing Trial Instance(s), because the package structure in Web IDE will otherwise not be displayed correctly.

    Then click **Delete** on your existing Trial Instances(s) to delete it.

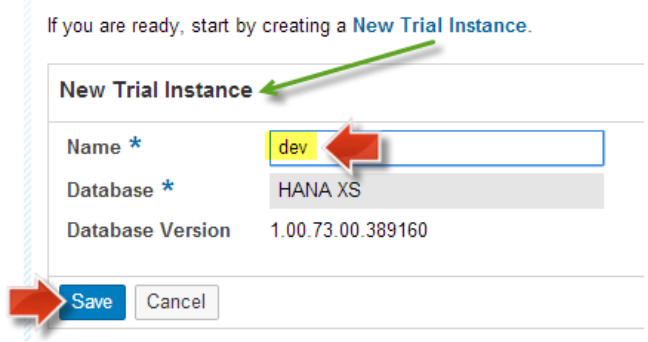
IMPORTANT NOTE: Before you delete your Trial Instance(s) save the related project data (i.e. your developed project content on trial), if this is relevant.

    After deletion of all existing Trial Instances(s) the *creating New Trial Instance* link is displayed again.

14. Click on **New Trial Instance** link to display the instance creation section.



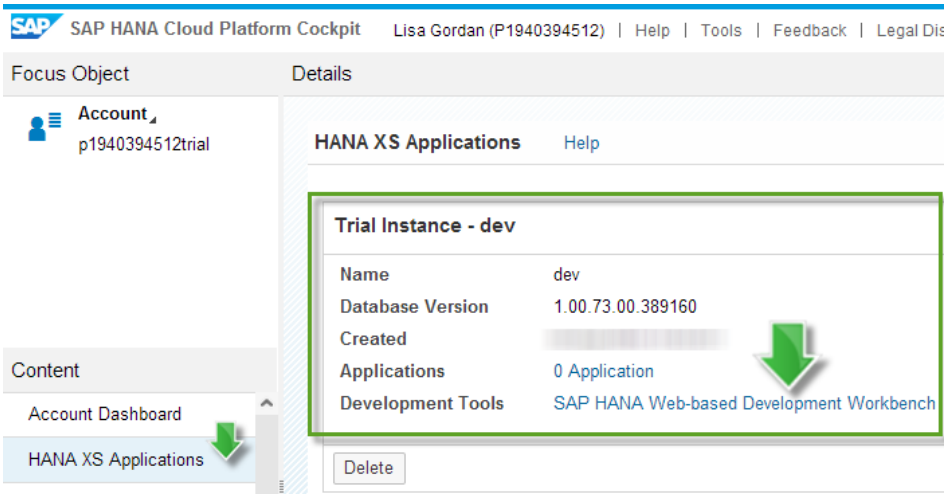
15. In the displayed section specify a name for your new Trial Instance (e.g. use **dev**) and then click **Save** to create it.



Result

Your SAP HANA Trial Instance with the before specified name (e.g. **'dev'**) has been created.

In the displayed Trial Instance section you find the SAP HANA Web-based Development Workbench link to launch your SAP HANA Web IDE tool.



Read more:

- SAP HANA Cloud Platform Documentation > Cockpit
- SAP HANA Cloud Platform Documentation > Creating a Trial SAP HANA Instance

previous next

2.3 Start SAP HANA Web-based Development Workbench

For development of our SAP HANA native PersonsList application you can now launch the SAP HANA Web-based Development Workbench from the SAP HANA Cloud Platform cockpit.

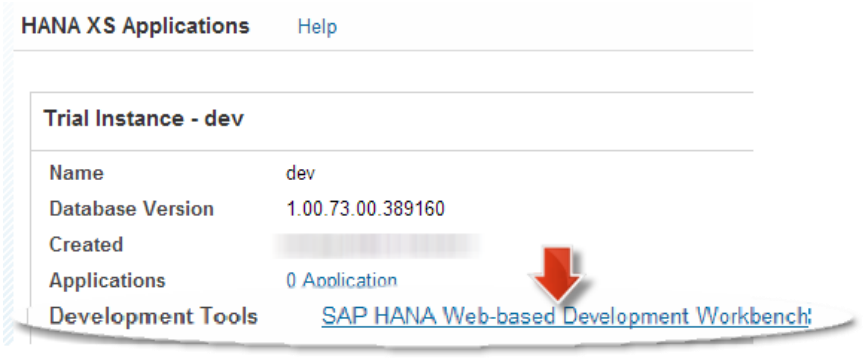
previous next

Prerequisites

- You entered your trial account in the SAP HANA Cloud Platform cockpit.
- You created a new trial instance of the SAP HANA database.

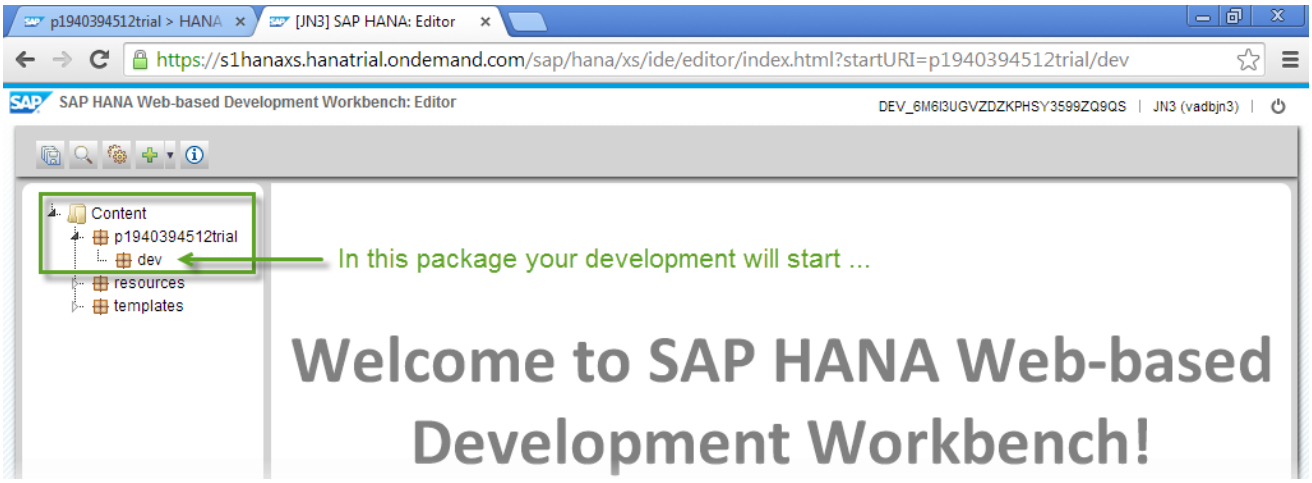
Hands-on Tasks

16. In order to develop a new SAP HANA native application from scratch select *Content* node **HANA XS Applications** and click link **SAP HANA Web-based Development Workbench** in section *Details* > *HANA XS Applications* > *Trial Instance – dev* > *Development Tools*.



Result

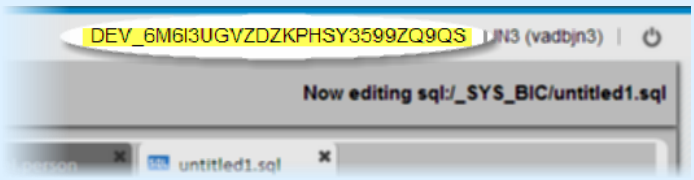
The Editor of the SAP HANA Web-based Development Workbench starts in a new browser tab. Under repository root node *Content* you see the initial package structure `<trial-account-name>/<trial-instance-name>` e.g. `p1940394512trial/dev`:



In the tutorial the developed SAP HANA XS project will be located in sub-package `perslist` under your trial account specific package. According to your own trial developer account name and the trial Instance name this results in `Content/<trial-account-name>/<trial-instance-name>/<project-name>` e.g. `Content/p1940394512trial/dev/perslist`.

NOTE: Where is the SAP HANA Database User?

In a free developer account on SAP HANA Cloud Platform trial landscape the **HANA database user** is *auto-created* for you by the SAP HANA Cloud Platform when you create a new developer account. When starting the HANA Web IDE from the SAP HANA Cloud Platform cockpit you login with the HANA database user but not with your account user. For this reason you do not get your account user name `p1940394512` displayed in the header of the HANA Web IDE but see a HANA database user ID like `DEV_6M6I3U…` with an GUID-suffix behind `DEV_`. When working through this tutorial you do not need to know/use this HANA database user ID.



previous next

What is the SAP HANA Web-based Development Workbench?

The *SAP HANA Web-based Development Workbench* (short term *HANA Web IDE*) is a browser-based integrated development environment (IDE) to build SAP HANA native applications with a particular emphasis on the following development experience qualities:

- **Zero-installation**: only a browser client is needed to start coding
- **Simple and fast development roundtrip**: just press 'run' and the app will start in the same browser instantly
- **Integrated debugging experience**: set a breakpoint and start hunting your bug immediately
- **Template-based development**: build and launch simple applications in less than 60 seconds
- **Database citizen**: explore the SAP HANA database catalog and security artifacts and content easily
- **Multi-device support**: work on your code on the go on your favorite tablets

Further information

- SAP Help: SAP HANA Developer Guide > 3.9 Developing Applications in Web-based Environments
- SAP HANA Cloud Platform Documentation: Developing with SAP HANA Web-based Tools
- SCN blog What´s New? SAP HANA SPS 07 Web-based Development Workbench, Thomas Jung, SAP AG, Dec 3, 2013
- SCN blog HANA XS development with the SPS07 Web IDE (focus on debugging), Kai Christoph Mueller, SAP AG, Nov 27, 2013

- Video Web-based Development Workbench, Thomas Jung, SAP AG, Dec 18, 2013

2.4 Remember Your Data

After executing successfully the above described steps you have to remember your accounts data as you need them later several times in the tutorial:

1.	**Free SAP HANA trial developer account**	Trial account name Trial account user Trial account password	: `p1940394512trial` : `p1940394512` : `********`
2.	**SAP HANA trial instance**	Database instance	: `dev`
3.	**SAP HANA XS repository location of PersonsList project**	Repository path:	: `Content/p1940394512trial/dev/perslist`
4.	**SAP HANA database schema**	Schema name:	: `_SYS_BIC`

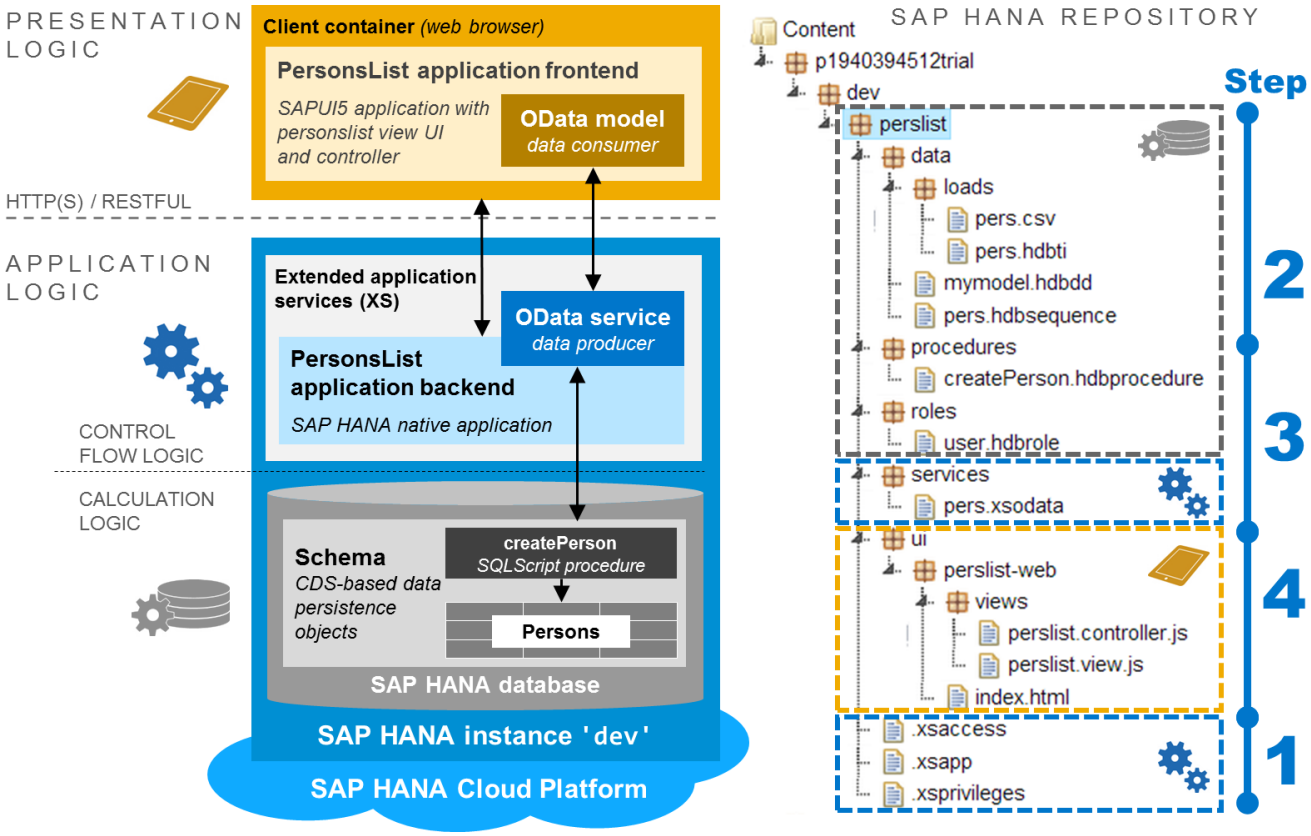
3 Build a Simple SAP HANA Native PersonsList Application

Preview

In Figure 3 you see architecture and content preview of the whole SAP HANA native PersonsList application you will develop step-by-step within the following four main sections of the tutorial:

- **Step 1: Create HANA XS application**: how to create a new and minimalistic SAP HANA native application using the SAP HANA Web-based Development Workbench.
- **Step 2: Set up the persistence model**: how to set up the persistence model for the *PersonsList* application in the SAP HANA database including database schema creation, CDS-based table definition and initial table load from a static *pers.csv* file.
- **Step 3: Code the backend:** how to build the application backend with SAP HANA extended application services including OData service exposure for read and write access and a SQLScript procedure *createPerson()* as modification exit for OData write requests. With this step your PersonsList application implements the *CR (=**C**reate & **R**ead)*-functionality of a full CRUD-application (**C**reate, **R**ead, **U**pdate, **D**elete).
- **Step 4: Build the SAPUI5 frontend:** how to build the application frontend with the UI development toolkit for HTML5 (SAPUI5).

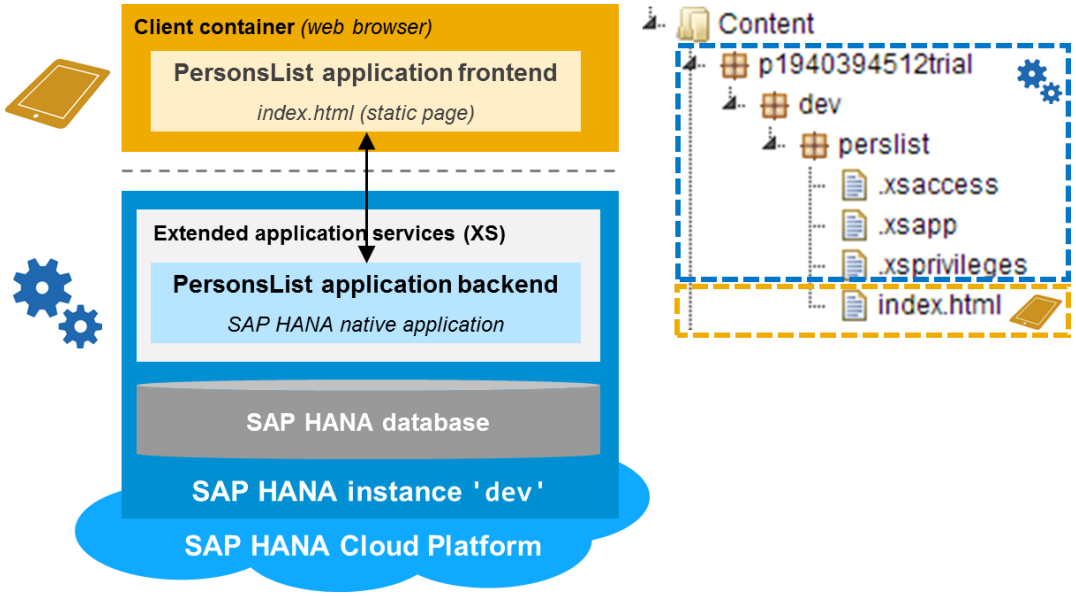
Figure 3: Architecture, artifacts and development steps of final SAP HANA native PersonsList application


PRESENTATION LOGIC
Client container (web browser)
PersonsList application frontend
SAPUI5 application with personslist view UI and controller
OData model
data consumer
HTTP(S) / RESTFUL
APPLICATION LOGIC
Extended application services (XS)
OData service
data producer
PersonsList application backend
SAP HANA native application
CONTROL FLOW LOGIC
CALCULATION LOGIC
Schema
CDS-based data persistence objects
createPerson
SQLScript procedure
Persons
SAP HANA database
SAP HANA instance 'dev'
SAP HANA Cloud Platform
SAP HANA REPOSITORY
Content
p1940394512trial
dev
perslist
data
loads
pers.csv
pers.hdbti
mymodel.hdbdd
pers.hdbsequence
procedures
createPerson.hdbprocedure
roles
user.hdbrole
services
pers.xsodata
ui
perslist-web
views
perslist.controller.js
perslist.view.js
index.html
.xsaccess
.xsapp
.xsprivileges
Step
2
3
4
1

3.1 Step 1: Create new SAP HANA native Application within Web IDE

Preview

Figure 4: Step 1 – initial creation of a bare bone SAP HANA native application



Design-Time Application Artifacts Created in this Step

File extension	Object	Description
Package	A container in the SAP HANA repository for development objects.	Packages are represented by folders. The package that contains the application-descriptor file becomes the root path of the resources exposed by the application you develop.
.xsapp	SAP HANA XS application descriptor	An application-specific file in a repository package that defines the root folder of a native SAP HANA application. All files in that package (and any sub packages) are available to be called via URL.
.xsaccess	SAP HANA XS application access file	An application-specific configuration file that defines permissions for a native SAP HANA application, for example, to manage access to the application and running objects in the package.
.xsprivileges	SAP HANA XS application- privilege file	A file that defines a privilege that can be assigned to an SAP HANA Extended Application Services application, for example, the right to start or administer the application.
index.html	Default entry page	Default entry (index) file with HTML markup to start SAP HANA application in a web browser.

3.1.1 Create blank SAP HANA XS Application

To realize native applications on SAP HANA, the *SAP HANA repository* is used to manage the different resources that altogether define the behavior and appearance of the application: either static content, libraries like SAPUI5 control and runtime libraries or the defined application resources containing stored procedures, table (entity) definitions, JavaScript-code and the like.

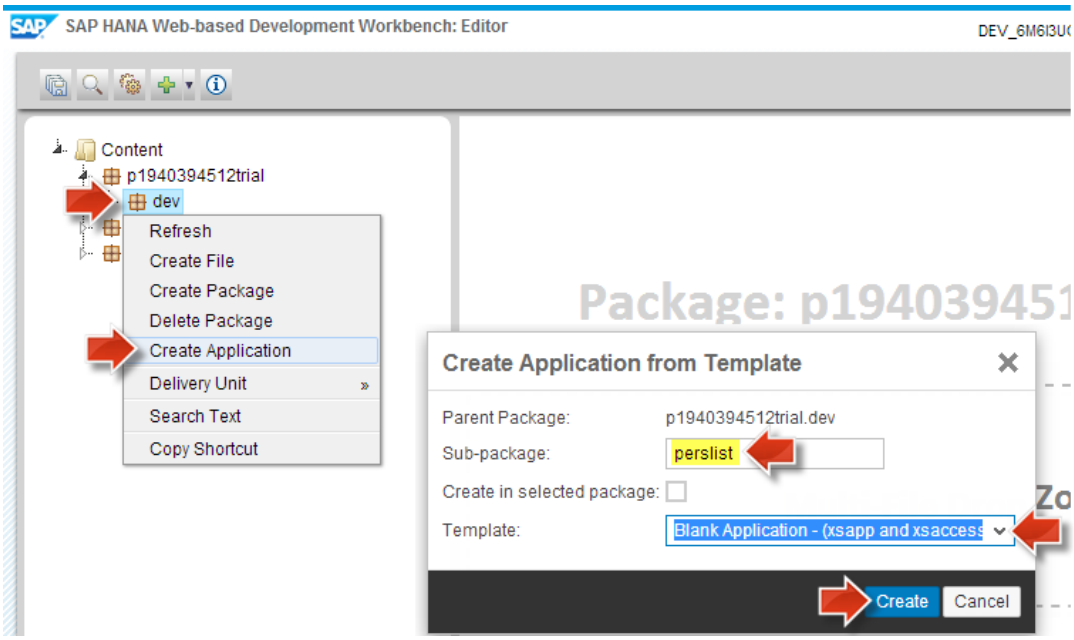
To create a new SAP HANA XS application inside the HANA repository we use the editor's "*Create Application from Template*" function that adds an initial application skeleton to a new package.

Prerequisites

- You have launched the *SAP HANA Web-based Development Workbench Editor* from *the SAP HANA Cloud Platform cockpit*. If you are inactive for some time, your session in the SAP HANA Web-based Development Workbench will become invalidated. To start a new session, go back to the SAP HANA Cloud Platform cockpit and make sure your session is active there.
- Under root node **Content** the initial package structure **/<trial-account-name>/<trial-instance-name>** e.g. `Content/p1940394512trial/dev` is displayed.

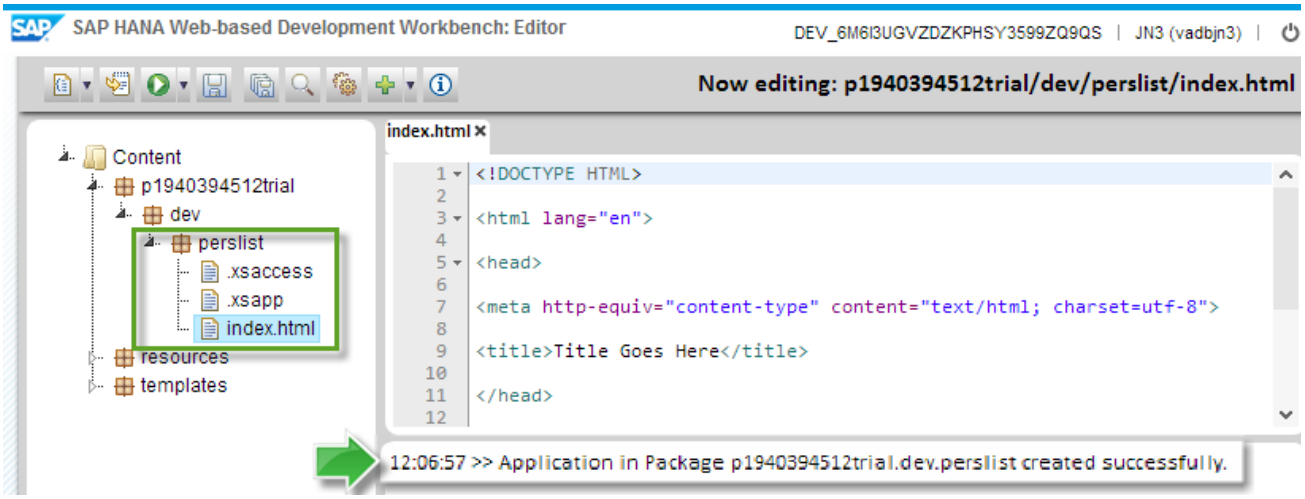
Hands-on Tasks

17. In the SAP HANA Repository tree select node item **Content → p1940xxtrial → dev** .
18. Open context menu on selected package **dev** and select item **Create Application**



19. In the *"Create Application from Template"* popup dialog enter sub-package name **`perslist`**.
20. In *Template* dropdown list select item **Blank Application – (xsapp and xsaccess)**
21. Press **Create.**
22. The system creates the files **index.html**, **.xsaccess** (HANA XS application access file), and **.xsapp** (HANA XS application descriptor), and automatically opens the **index.html** file in a new editor tab.

Result



As a result, the following SAP HANA application artifacts are created and listed under your newly added package:

- **application root package:** New package **Content/<trial account name>/<SAP HANA trial instance>/perslist** (here `Content/p1940394512trial/dev/perslist`) in SAP HANA repository that comprises the initial application..
- **.xsaccess:** Expose your package content, meaning it can be accessed via HTTP, and assign access controls, for example, to manage who can access content and how.
- **.xsapp:** Each application that you want to develop and deploy on SAP HANA extended application services (SAP HANA XS) must have an application descriptor file. The application descriptor is the core file that you use to describe an application's framework within SAP HANA XS.
- **index.html:** web page to start your application in a browser client. The **index.html** initially contains only a page title and one paragraph. We will later replace it with content needed to run a SAPUI5 application frontend.

previous next

3.1.2 Run Blank SAP HANA XS Application in Web Browser

After having created the blank XS application under package **p1940xxtrial/dev/perslist** with the editor's template creation function we can run it in the web browser.

previous next

Hands-on Tasks

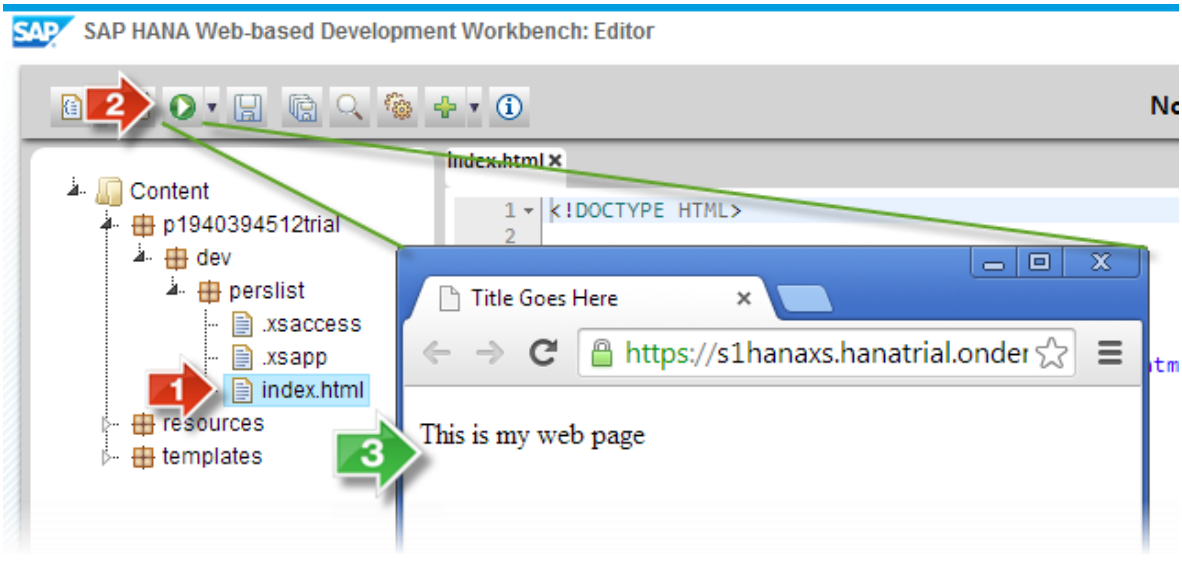
23. To test the blank application in the web browser, open your application package in the editor's Content tree: **Content → p1940xxtrial → dev → perslist → index.html**.

NOTE: The editor toolbar functions get automatically adapted to the selected item in the Content tree. The toolbar button "**Run on server (F8)**"  gets displayed for a selected **index.html** file to open the application frontend in a browser window.

24. Press button  **"Run on server (F8)"** to start the blank application in a web browser by requesting the **index.html** file.

Result

The SAP HANA XS Web application frontend (with the **index.html** content) is displayed in the web browser.



3.1.2.1 Application Descriptor .xsapp

Each SAP HANA native application must have an **application descriptor** file (a file without file name and extension **.xsapp**). The application descriptor is the core file that is used to describe an application's availability within SAP HANA extended application services. The package that contains the application descriptor file becomes the root path of the resources exposed by the application. The file content must be valid JSON for compatibility reasons (like **{}**) but does not have any content used for processing.

Terminology: JSON, or *JavaScript Object Notation*, is an open standard format that uses human-readable text to transmit data objects consisting of attribute–value pairs. It is used primarily to transmit data between a server and web application, as an alternative to XML.

3.1.3 Define Application Privileges in File .xsprivileges

In SAP HANA extended application services (SAP HANA XS), the application-privileges **(.xsprivileges**) file can be used for access authorization in the **.xsaccess** file or checked programmatically to secure parts of an XS application (examples: to start the application or to perform administrative actions on an application). It does not have a name and is formatted according to JSON rules. Multiple **.xsprivileges** files are allowed, but only at different levels in the package hierarchy.

Hands-on Tasks

25. In HANA Web IDE editor navigate to the SAP HANA Repository package **Content → p1940xxtrial → dev → perslist** and select package node **perslist**.
26. Choose context menu item **Create File**.
27. Enter `.xsprivileges` as filename (to replace the default string **0_NewFile.js** highlighted in blue).
28. In the editor tab for file **.xsprivileges** paste the following content:

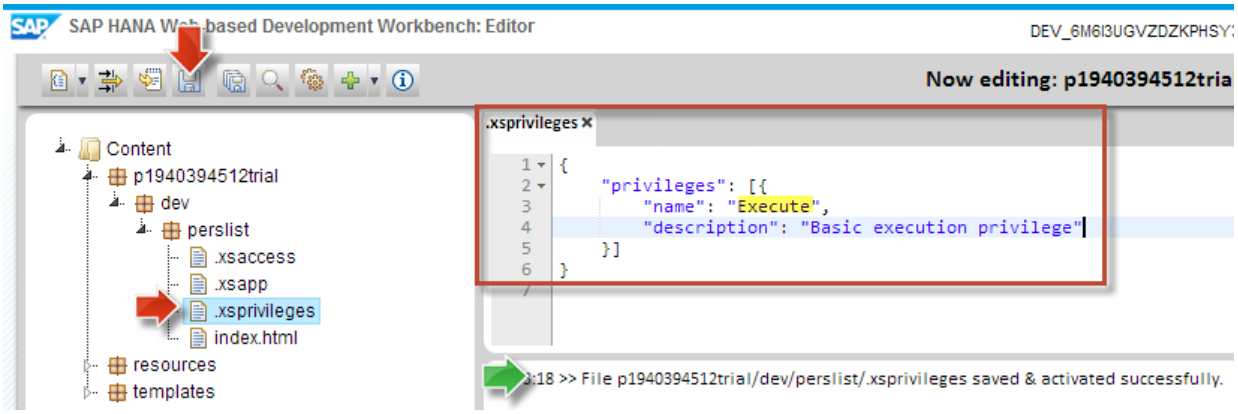
Source Code 1: .xsprivileges file to define the application privilege p1940xxtrial.dev.perslist::Execute

```
{
    "privileges": [{
        "name": "Execute",
        "description": "Basic usage privilege"
    }]
}
```

29. Save the **.xsprivileges** file with **Ctrl+S** or toolbar button **Save** to activate it.

Result

The new file with name **.xsprivileges** was created, saved and activated under package **perslist**.



Inside the **.xsprivileges** file, a privilege is defined by specifying an entry name with an optional description. This entry name is then automatically prefixed with the package name to form the unique privilege name, for example **p1940xxtrial.dev.perslist::Execute**.

previous next

3.1.4 Define Application Access in File .xsaccess

The **.xsprivileges** file lists the *authorization levels* that are available for access to an application package. The **application access file** with name **.xsaccess** (suffix only) defines which *authorization level* is assigned to which application package (i.e. what privileges are required to access content of a package and its sub-packages). It also defines data (content) exposure to clients, defines the authentication method, specifies URL rewrite rules and defines if SSL is required.

Similar to the **.xsapp** file, also the .**xsaccess** file is using the JSON format. **.xsaccess** files can be located in the root package of the application or in any sub-package. When accessing the application via an URL, the **.xsaccess** file in the sub-package or up the package hierarchy to the root package is used.

previous next

Hands-on Tasks

30. In SAP HANA repository tree select the application access file **Content → p1940xxtrial → dev → perslist → .xsaccess** to edit it in a new editor tab.

31. In the editor tab for file **.xsaccess** replace the entire file content `{"exposed" : true ,"authentication" : [{"method" : "Execute"}]}` with Source Code 2:

NOTE: The copied source code contains the string **`p1940xxtrial`** to be replaced with your own **trial account name, e.g p1940394512trial.**

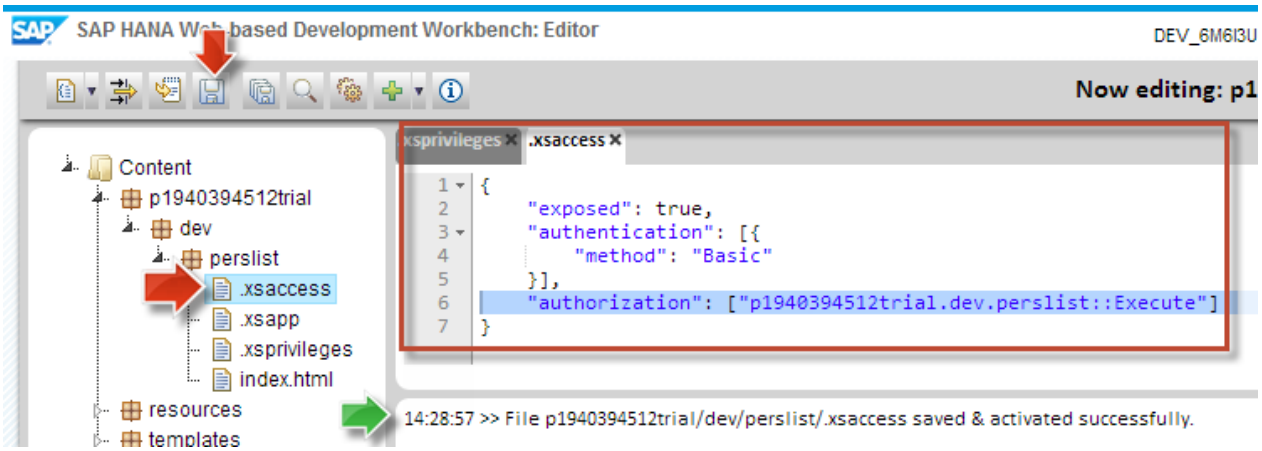
Source Code 2: .xsaccess file to define application access permissions

```
{
    "exposed": true,
    "authentication": [{
        "method": "Basic"
    }],
    "authorization": ["p1940xxtrial.dev.perslist::Execute"]
}
```

32. Save the **.xsaccess** file with **Ctrl+S** or toolbar button **Save** to activate it.

Result

The `authorization` keyword in the **.xsaccess** file enables you to specify which authorization level is required for access to a particular application package, e.g. to **p1940xxtrial.dev.perslist**. A user must now have this privilege to access the application package where the **.xsaccess** file resides (see section 3.2.7 Grant Role user.hdbrole to Your User).



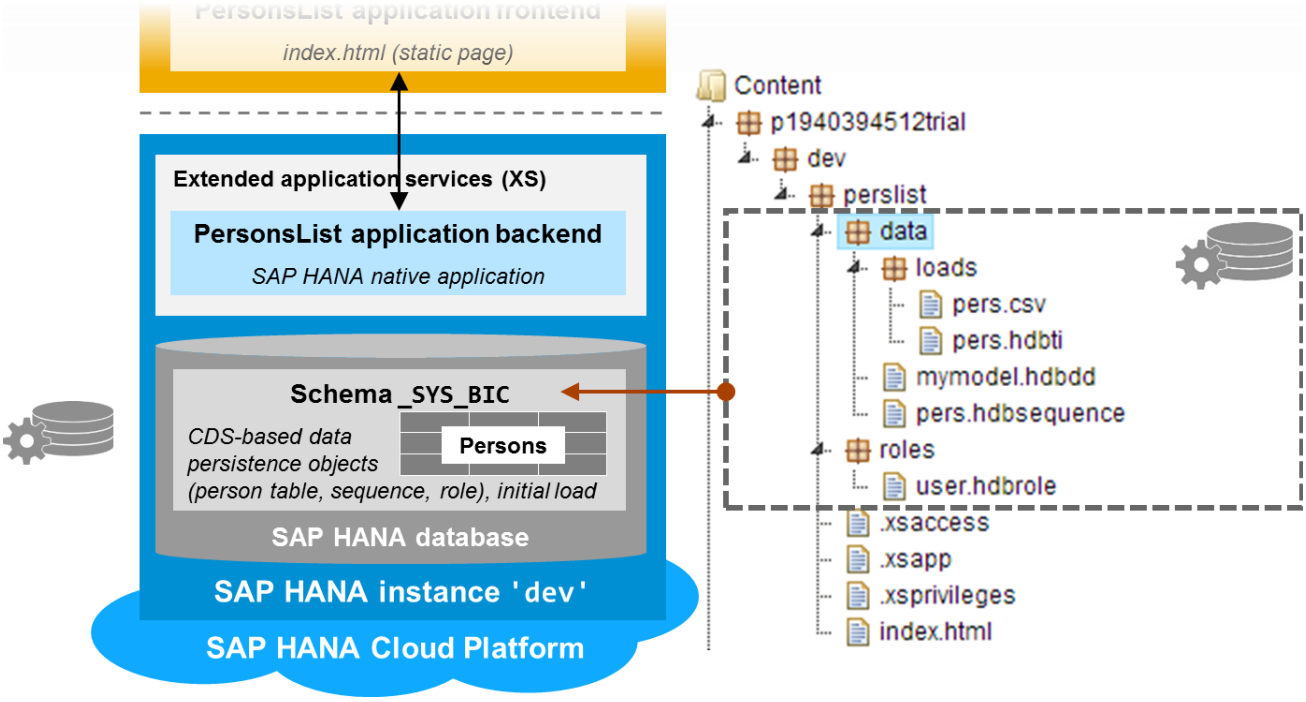
Read more: SAP HANA Developer Guide: The Application-Access File

previous next

3.2 Step 2: Set up the Persistence Model in SAP HANA Database

Preview

Figure 5: Step 2 - Backend part in SAP HANA database with CDS-based data persistence objects and initial load



Design-Time Application Artifacts created in this Step

File extension	Object	Description
`.hdbdd`	CDS (Core Data Services) data definition document	A file containing a design-time definition of a CDS-compliant data-persistence object (for example, an entity or a data type) using the Data Definition Language (DDL).
`.hdbprivilege`	Application Privilege	A file that defines a privilege that can be assigned to a HANA XS application, for example, the right to start or administer the application.
`.hdbrole`	Role	A file containing a design-time definition of an SAP HANA user role.
`.hdbsequence`	Sequence	A design-time definition of a database sequence, which is set of unique numbers, for example, for use as primary keys for a specific table.
`.hdbti`	Table import definition	A table-import configuration that specifies which .csv file is imported into which table in the SAP HANA system

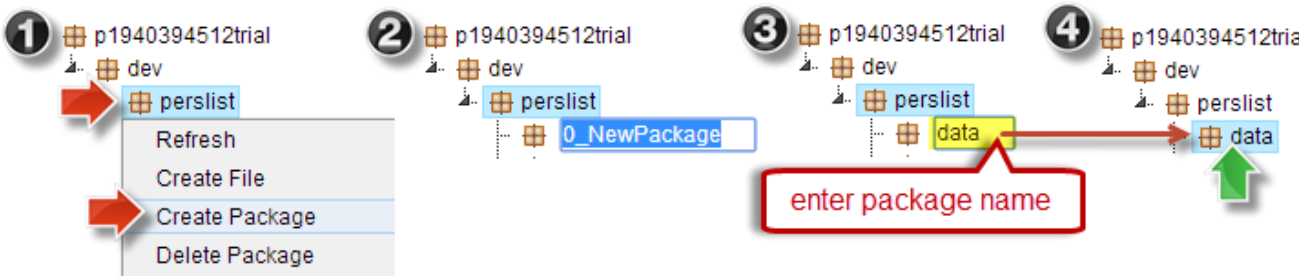
3.2.1 Create Sub-Package *data*

To logically structure content of our whole *PersonsList* application inside the SAP HANA repository we create sub-packages for related artifacts under the application's root package **p1940xxtrial/dev/perslist**. Files related with the persistence model are stored in sub-package **data** that can easily be added within three steps:

previous next

Hands-on Tasks

33. In the Editor's repository content tree select package node **Content → p1940xxtrial → dev → perslist**
34. Select context menu item **Create Package.** The newly created sub-package is named *0_NewPackage* by default.
35. Enter `data` as name of the new package



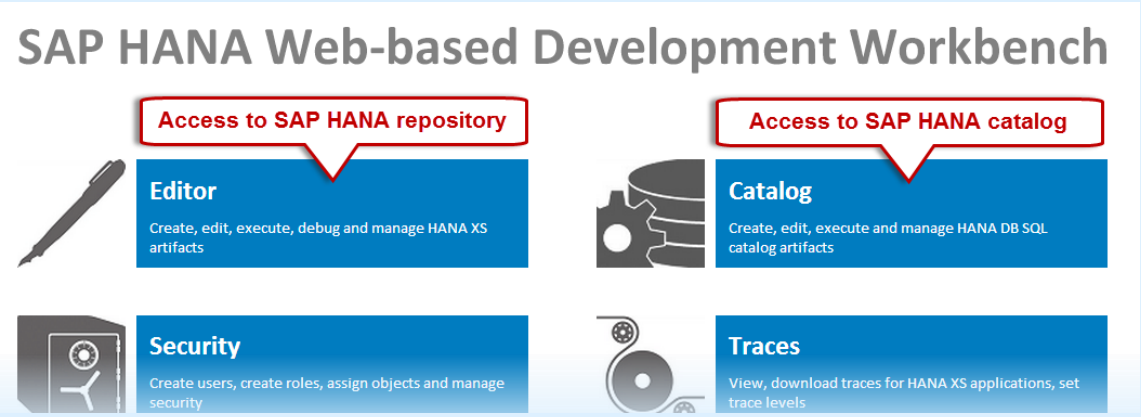
Result

The new sub-package with name **data** was created under package **p1940xxtrial/dev/perslist**:

previous next

SAP HANA repository: The *SAP HANA repository* is the central component of the SAP HANA development infrastructure and an integral part of the SAP HANA system. The repository is used for central storage and versioning of software artifacts, and it is also the foundation for lifecycle management for SAP HANA content and for the translation of SAP HANA applications. The repository provides the export and import functions needed for shipping applications to customers and for transporting development results between SAP HANA systems.

The repository supports concurrent development in teams. As the central storage, it enables sharing of development artifacts with other developers. The repository also supports concurrent development with conflict resolution, for example by merging conflicting versions.



SAP HANA catalog: SAP HANA native applications persist their content in the SAP HANA repository and, depending on the content type, compile artifacts into the runtime *catalog*.

Database schema: The SAP HANA database contains a catalog that describes the various elements in the system. The catalog divides the database into sub-databases known as *schema*. A database schema enables you to logically group together objects such as tables, views, and stored procedures. Without a defined schema, you cannot write to the catalog.

3.2.2 Define Data Type *Person* within CDS Document

In the past, database objects could only be created via SQL directly in the database catalog. However this meant they couldn't be transported via delivery units like all other repository objects. As part of SAP HANA native development, we now have tools named *Core Data Services (CDS)* to create database objects in SAP HANA via a repository representation which generates the catalog objects upon activation.

To make up the data-persistence model for our application, we define a new model entity ***Person*** within a so-called *HANA data-persistence object definition file* by using the CDS Data Definition Language.

previous next

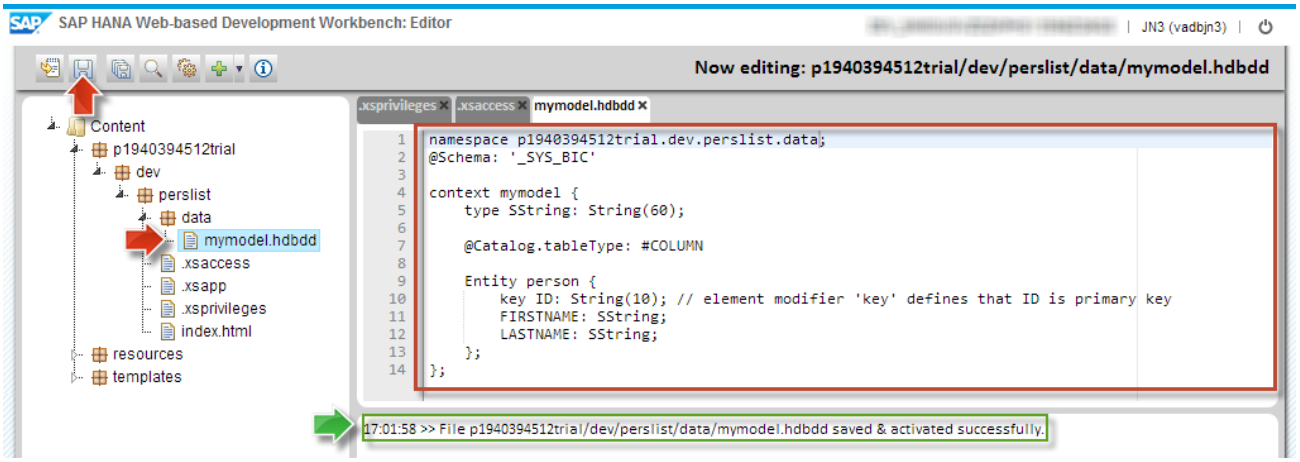
Hands-on Tasks

36. Open the **Editor** tab of SAP HANA Web-based Development Workbench.
37. In the Editor's repository content tree select package node **Content → p1940xxtrial → dev → perslist → data**
38. Select context menu item **Create File** and enter filename **`mymodel.hdbdd`.** The design-time object definition that you create using the CDS-compliant syntax must have the file extension **.hdbdd**
39. Enter Source Code 3 to define the entity **person**. Also replace the `highlighted string` corresponding to the name of your own trial account.

Source Code 3: Data definition file mymodel.hdbdd

```
namespace p1940xxtrial.dev.perslist.data;
@Schema: '_SYS_BIC'

context mymodel {
    type SString: String(60);

    @Catalog.tableType: #COLUMN

    Entity person {
        key ID: String(10); // element modifier 'key' defines that ID is primary key
        FIRSTNAME: SString;
        LASTNAME: SString;
    };
}
```

Screenshot 2: Activation of data definition file mymodel.hdbdd in SAP HANA repository



Result

The activation process implies to following functions:

- Syntax validation of data definition file **mymodel.hdbdd**
- Creation of table **<namespace>::<context name>.<entity name>**, in our case of table **p1940xxtrial.dev.perslist.data::mymodel.person,** in database schema **_SYS_BIC**. In the next step You open HANA database Catalog with the HANA Web IDE to display the *person* database table in _SYS_BIC schema (see Screenshot 3).

NOTE: In your trial HANA account on SAP HANA Cloud Platform the existing database schema **_SYS_BIC** is used to store your own database objects, so that you need not and even cannot create your own database schema. In a SAP HANA instance on a productive SAP HANA Cloud Platform landscape you would need to create an own database schema instead.

previous next

Core Data Services in a Nutshell

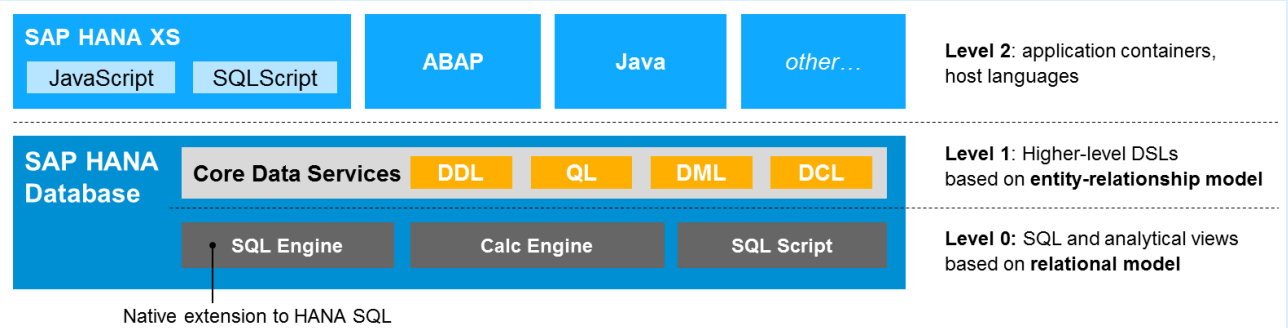
Core Data Services (CDS) enhance SQL to allow defining and consuming semantically rich data models natively in HANA applications, thereby improving productivity, consumability, performance and interoperability.

As most databases, HANA supports SQL as the standard means to define, read and manipulate data. On top of that, pretty much every consumer technology or toolset introduces higher-level models to add crucial semantics and ease consumption – e.g. OData EDM models, the Semantic Layer in the BI platform, JPA and enterprise objects in Java, or the business objects frameworks in ABAP. Also the River programming model and RDL follow this pattern.

Even though those higher-level models share many commonalities, the individual information cannot be shared across stacks. This leads to a fragmented environment and a high degree of redundancy and overhead for application developers and customers. To address that, we introduce a common set of domain-specific languages (DSL) and services for defining and consuming semantically rich data models as an integral part of HANA – called Core Data Services --, that can hence be leveraged in any consuming stack variant as depicted in the following illustration.

The Core Data Services comprise a family of domain-specific languages (highlighted in the illustration above) which serve as a common core model for all stacks on top:

- Data Definition Language (DDL) to define semantically rich domain data models which can be further enriched through Annotations.
- Database Query Language (DQL) to conveniently and efficiently read data based on data models as well asto define views within data models.
- Data Manipulation Language (DML) to write data
- Data Control Language (DCL) to control access to data
- Expression Language (EL) to specify calculated fields, default values, constraints, etc. within queries as well as for elements in data models.



Core Data Services focus on providing functional services independent of any programming language and language paradigms. They don't specify nor make assumptions on how to add application logic and behavior using general-purpose programming languages and services of application containers.
Further information:

- SAP HANA Developer Guide: Creating the Persistence Model in Core Data Services (CDS)
- SAP HANA Academy: SAP HANA SPS 6 - What's New: Development - Core Data Services, by Thomas Jung

3.2.3 Display person Table in SAP HANA Database Catalog

For a customer/developer account on a **productive** SAP HANA Cloud Platform you now would need to create a new database schema in the SAP HANA catalog. This database schema would enable you to create and activate application artifacts such as tables and database procedures. Without a defined SAP HANA database schema (in a SAP HANA instance on a productive SAP HANA Cloud Patform account), database objects cannot be generated in the SAP HANA catalog upon activation of specific design-time artifacts that are added to the SAP HANA repository.

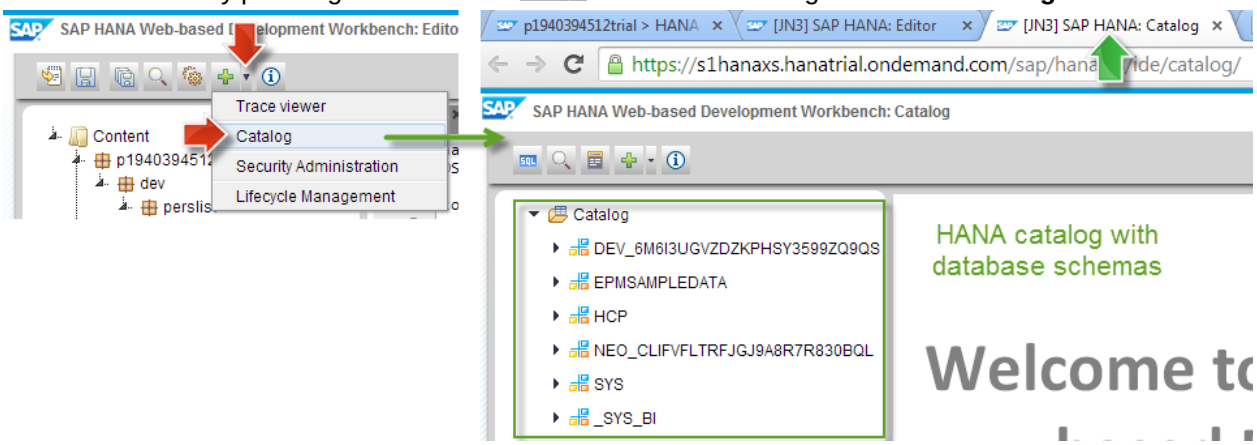
In the SAP HANA database instance (e.g. named **dev**) of your trial SAP HANA Cloud Platform account the existing SAP HANA schema **_SYS_BIC** is used for all your database objects so that you do not need to (and even cannot) create an own database schema. To see the **_SYS_BIC** schema containing your **'person'** database table (created via the **'mymodel.hdbdd'**) you have to execute a SQL command as follows.

Further Information: SAP HANA Cloud Platform Documentation > Specific Procedures and Views

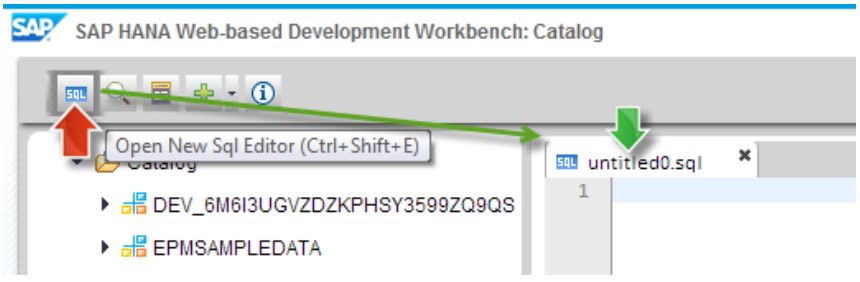
previous next

Hands-on Tasks

40. To see the database schemas, that are visible in your trial account, open the **SAP HANA catalog** in a new browser tab by pressing toolbar button *More* and selecting menu item **Catalog.**



41. Open a SQL console by clicking the toolbar icon **Open New SQL Editor** (blue SQL icon)



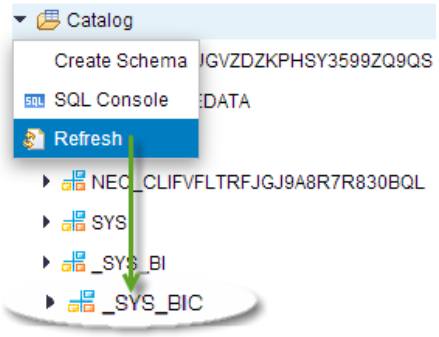
42. Copy the SQL code string from Source Code 4 into the opened SQL Editor:

Source Code 4: SQL call to view activated database objects in schema _SYS_BIC

```
CALL "HCP"."HCP_GRANT_SELECT_ON_ACTIVATED_OBJECTS"
```

43. Execute the SQL command by clicking the toolbar icon **Execute** (green icon with triangle).

44. Select the **Catalog** root node and choose context menu item **Refresh** to make the _**SYS_BI**C schema visible in your catalog.

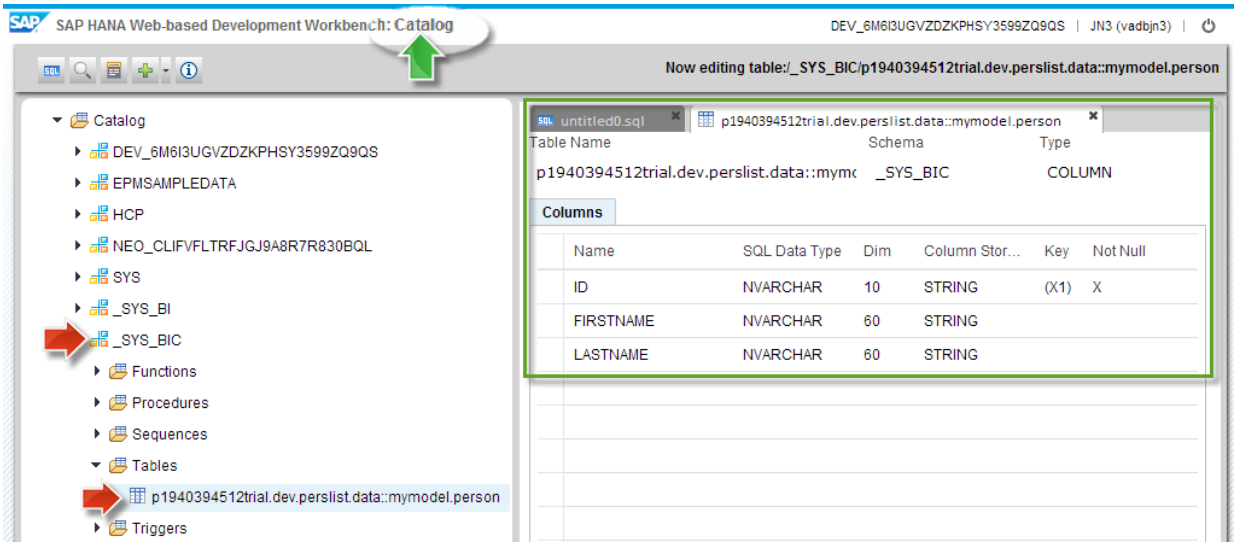


45. In the catalog tree expand the node **Content → _SYS_BIC → Tables** to display the newly created person table as defined in CDS document **mymodel.hdbdd**. The person table is displayed with its fully qualified name **`p1940394512trial.dev.perslist.data::mymodel.person`**.

Result

Your newly defined database table **'person'** is displayed in the **Tables** node of the catalog schema _SYS_BIC. The other nodes *Functions, Procedures, Sequences, Triggers* and *Views* are still empty and will be partly filled in advance of the tutorial. The *table metadata* (with defined columns, schema name _SYS_BIC, fully qualified table name) of your newly defined database table **'person'** gets displayed in a new catalog tab.

Screenshot 3: Automatic database table creation in SAP HANA catalog schema on activation of data definition file mymodel.hdbdd



previous next

3.2.4 Load Mock Data from a .csv File into Database Table

For sake of simplicity we use a comma-separated-values file to populate the Person table with initial data (*initial load*). Again this can easily be done by activating a **pers.csv** file together with a special ***HANA database table import file*** we add to the sub-package **data** in the SAP HANA repository. On activation the person table (**p1940xxtrial.dev.perslist.data::mymodel.person**) gets automatically filled with mock data that is read from the csv file.

3.2.4.1 Add Mock Data File *pers.csv* to new sub-package *loads*

previous next

Hands-on Tasks

46. In Editor tab select repository package **Content → p1940xxtrial → dev → perslist → data**.

47. Create new sub-package **`loads`** with context menu item ***Create Package***.
48. Select the newly created package ***loads*** and choose menu item ***Create File.***
49. Enter **`pers.csv`** as file name.
50. Enter a list of three comma-separated person entries (see Source Code 5):

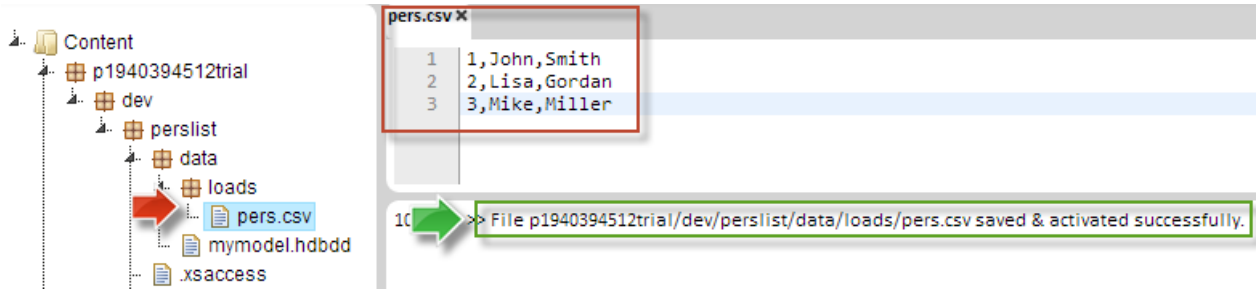
Source Code 5: pers.csv file to import mock data from a list of comma separated values

```
1,John,Smith
2,Lisa,Gordan
3,Mike,Miller
```

51. Save the **pers.csv** file.

Result

The **pers.csv** content for preloading later the persons table in the HANA catalog has been successfully saved and activated.



previous next

3.2.4.2 Add Table Import Definition File *pers.hdbti*

previous next

Hands-on Tasks

52. Select package **loads** and choose menu item ***Create File***.
53. Enter **`pers.hdbti`** file name to add a new table import definition.
54. Enter the import statement from Source Code 6 into the **pers.hdbti** file and replace the `highlighted strings` with your own account name:

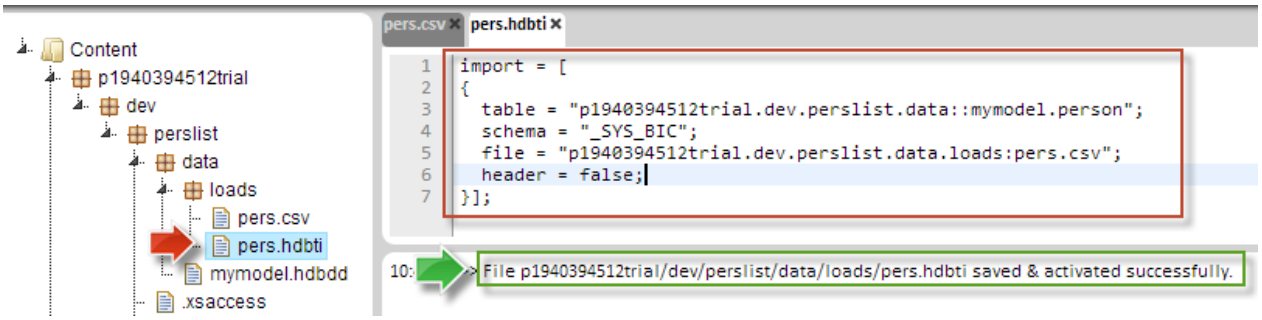
Source Code 6: pers.hdbti file to import table data from .csv file

```
import = [
{
  table = "p1940xxtrial.dev.perslist.data::mymodel.person";
  schema = "_SYS_BIC";
  file = "p1940xxtrial.dev.perslist.data.loads:pers.csv";
  header = false;
}];
```

55. Save the new **pers.hdbti** file.
56. Make sure that the editor returns a *success message* for activation of the new **pers.hdbti** file. Otherwise the automatic data transfer (initial load) from the **pers.csv** file to new records in the Persons database table will not be successfully processed.

Result

The **pers.hdbti** file which connects the preload content **pers.csv** file with the person table has been saved and by activating it the preload content will be inserted into the person table.

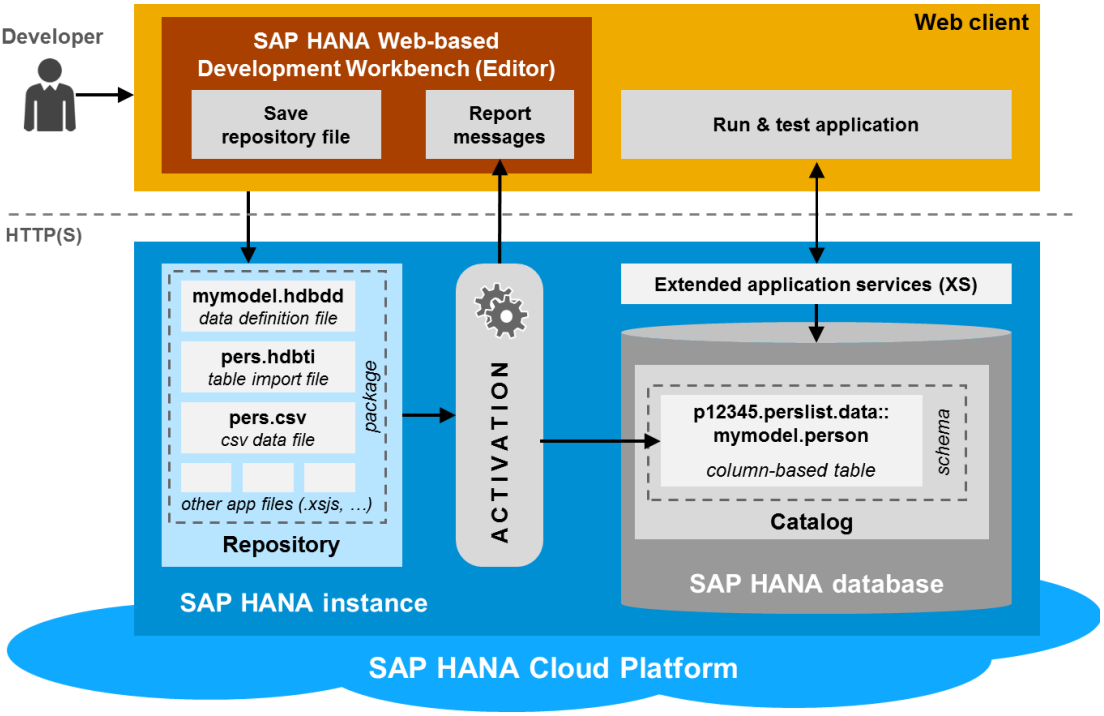


previous next

What happens on Activation of SAP HANA Repository Content?

Figure 6 illustrates the generic functions applied by SAP HANA when a developer activates content in the HANA Web IDE Editor. On activation of a repository file, the file suffix, for example, **.hdbdd**, is used to determine which runtime plug-in to call during the activation process. The plug-in reads the repository file selected for activation, in this case a CDS-compliant entity, parses the object descriptions in the file, and creates the appropriate runtime objects.

Figure 6: Save and activate XS application content – what happens on SAP HANA Platform side?



- **determine file type by extension**: we have three files for data definition, table import and csv data
- **validate content**: content and dependencies of three files needed for the initial load into the person table
- **create, update, delete runtime object in HANA catalog**: person table gets filled with mock data
- **report result messages** to the developer in HANA Web IDE

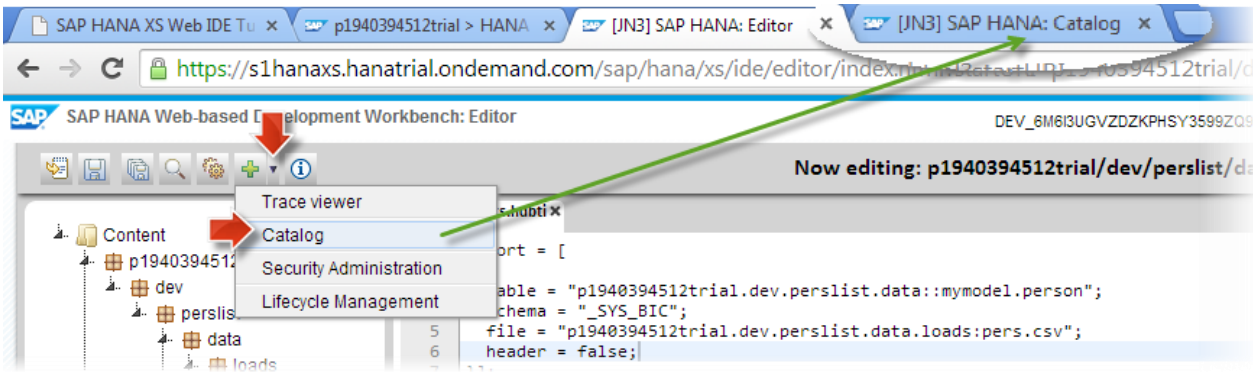
3.2.4.3 Test Initial Load of Person Table Data in SAP HANA Catalog

The previously activated table import definition file **pers.hdbti** implies the automatic initial load of csv-file data into the persons table (see Figure 6). To test whether the three initial *Person* records were successfully loaded into the Persons table we open the catalog tool of the HANA Web IDE and execute a SELECT statement:

previous next

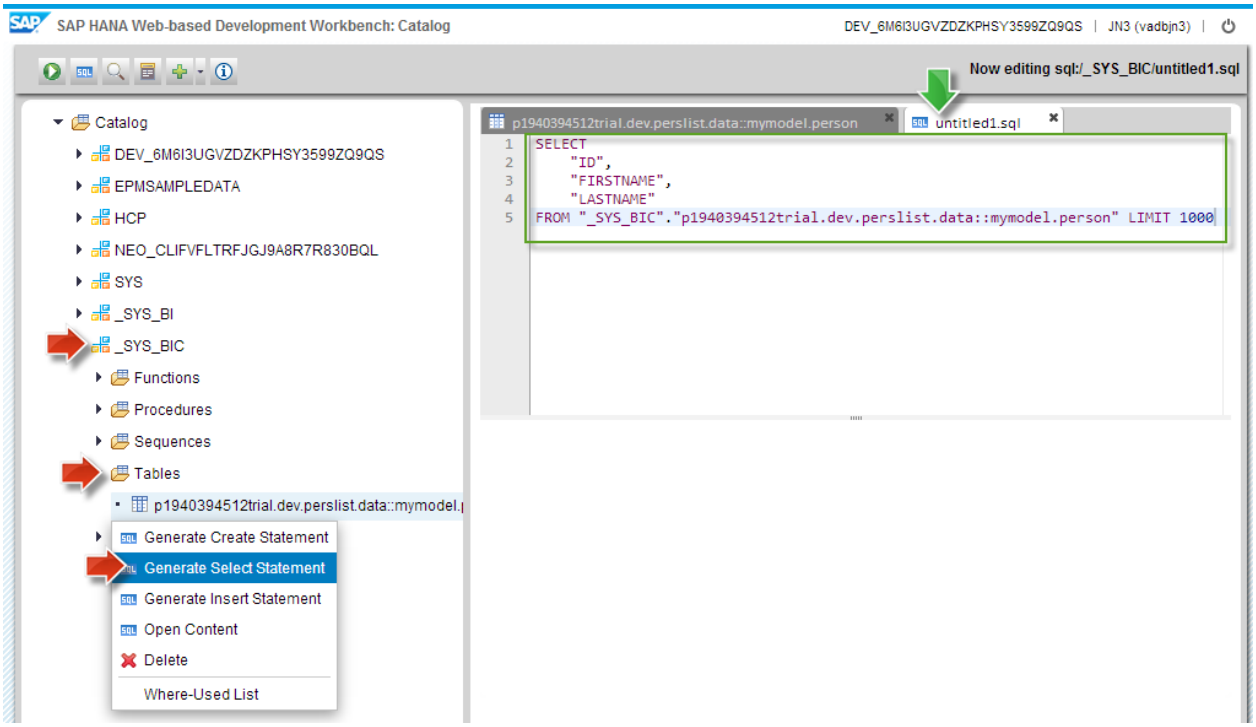
Hands-on Tasks

57. Switch to the **SAP HANA: Catalog** browser tab that you opened before in section 3.2.3. In case you have closed the tab open it again by pressing the editor toolbar button **More** and selecting menu item **Catalog.**



58. In the HANA catalog tree that gets displayed in a new browser tab select the *Person* table under node **Catalog → _SYS_BIC → Tables → p1940xxtrial.dev.perslist.data::mymodel.person**.

59. Select context menu item ***"Generate Select Statement".*** The HANA Web IDE adds a new editor tab with a SQL SELECT statement to read all records that are stored in the Persons table.

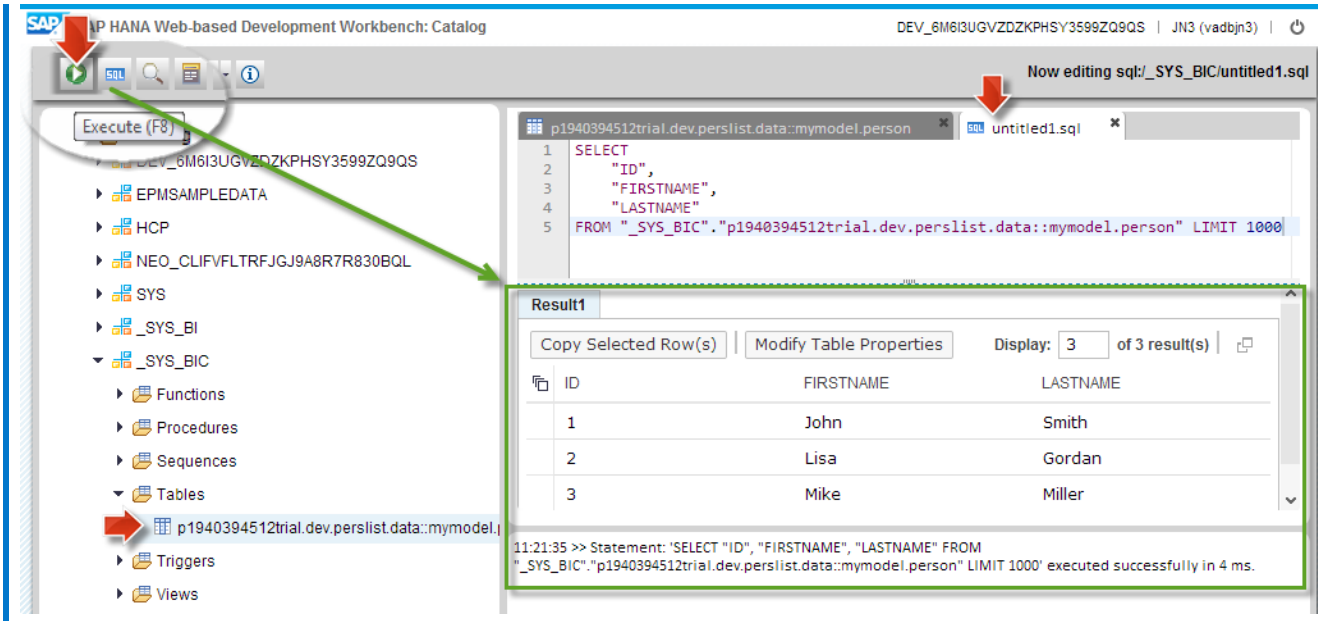


60. After successful initial load from the **pers.csv** file in the last step the SQL SELECT should return three records. To execute the generated SQL STATEMENT press keyboard shortcut **F8** or the execute button in the catalog toolbar.

NOTE: You can also apply the "**Open Content**" context menu function to generate the SQL statement and to view the result table in one single step (see section 3.5.4). For sake of better understanding we apply two separate functions at the first time.

Result

As a result the Person table content with three records (loaded before) gets displayed underneath the generated SQL select statement.



previous next

3.2.5 Add Database Sequence File to auto-generate Keys for new Records in Persons Table

previous next

Hands-on Tasks

61. In the Editor's repository content tree select package **Content → p1940xxtrial → dev → perslist → data**.

62. Select context menu item **Create File** and enter filename **`pers.hdbsequence`**.

63. Enter Source Code 7 to define a database sequence and replace the `highlighted string` with your own account name:

Source Code 7: Database sequence file pers.hdbsequence

```
schema= "_SYS_BIC";
start_with= 4;
maxvalue= 1000000000;
nomaxvalue=false;
minvalue= 4;
nominvalue=true;
cycles= false;
depends_on_table= "p1940xxtrial.dev.perslist.data::mymodel.person";
```

64. Save the new sequence file with Ctrl + S to activate it.

Result

The **pers.hdbsequence** file has been activated and will be used later to generate a serial list of unique numbers of person entity IDs.

previous next

Terminology: Sequences

A sequence is a database object that generates an automatically incremented list of unique numbers according to the rules defined in the sequence specification.

The sequence of numeric values is generated in an ascending or descending order at a defined increment interval, and the numbers generated by a sequence can be used by applications, for example, to identify the rows and columns of a table, to coordinate keys across multiple rows or tables.

Further information: SAP HANA Developer Guide: Create a Sequence

3.2.6 Define new Role in File user.hdbrole

previous next

Hands-on Tasks

65. In Editor tab select repository package **Content → p1940xxtrial → dev → perslist**
66. Create new sub-package **`roles`** with context menu item **Create Package.**
67. Select the newly created package **roles** and choose menu item **Create File.**
68. Enter **`user.hdbrole`** as file name and press Enter.
69. In the opened editor tab of file **user.role** enter Source Code 8 to and replace the `highlighted string` with your own account name:

Source Code 8: Role definition file user.hdbrole

```
role p1940xxtrial.dev.perslist.roles::user {
 1 sql object p1940xxtrial.dev.perslist.data::mymodel.person: SELECT, INSERT, UPDATE,
DELETE;
    application privilege: p1940xxtrial.dev.perslist::Execute; 2
}
```

NOTE: Know the code

1. Define a (database) *object privilege* for the underlying design-time object *person* (this *entity* was defined in CDS document **mymodel.hdbdd)** using the keyword `sql object` as follows: `sql object <package>:<object>.extension: PRIVILEGE;`. Privileges on the same object may be comma-separated or split up in several lines.
2. Define an *application privilege* with keyword `application privilege` in code line `application privilege: p1940xxtrial.dev.perslist::Execute;`. This makes the privilege "*Execute*" (defined in repository file **.xsprivileges**) to an *application privilege* at runtime. Remember, that application privileges define the authorization level required for access to an SAP HANA XS application, for example, to start the application or view particular functions and screens.

70. Save the new role definition file to activate it.

Result

The content of the file **user.hdbrole** has been successfully saved and activated. When a user accesses the SAP HANA database using a client interface (for example, ODBC, JDBC, or like in our case HTTP), his or her ability to perform database operations on database objects is determined by the privileges that he or she has been granted. In our case the defined "*Execute*" privilege is assigned to the role **user.hdbrole** as a privilege of type "*application privilege*".

Further Information: SAP HANA Developer Guide: User Roles, Object Privileges, Application Privileges

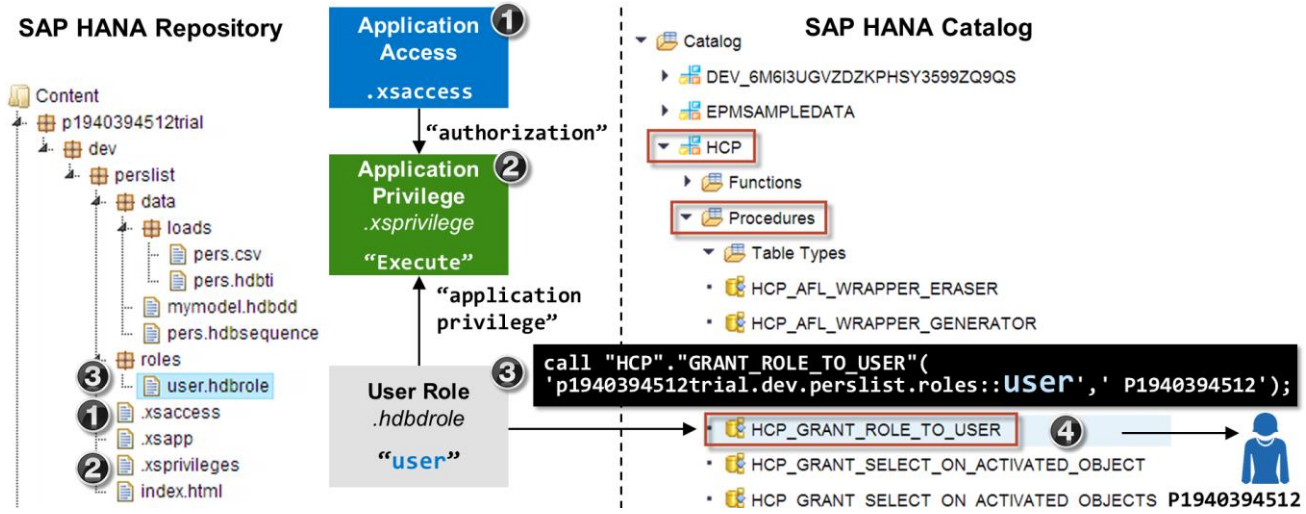
previous next

3.2.7 Grant Role user.hdbrole to Your User

In the next step the new role **user.hdbrole** gets granted to your account user **P1940XX** (technically to your HANA database user DEV_XX) by calling the HANA database procedure `HCP_GRANT_ROLE_TO_USER()` in catalog schema **HCP**. Figure 7 depicts the dependencies between application access file, application privilege, user role definition and role-to-user assignment.

See Figure 7 on next page …

Figure 7: How the definition of application access, application privilege, user role and role assignment fit together



- In the application access file **.xsaccess** (1) we authorized the application privilege named "*Execute*" to access the application content (of package **p1940xxtrial.dev.perslist** and its sub-packages).
- The application privilege "*Execute*" is defined in the **.xsprivilege** file (2).
- The assignment of the "*Execute*" application privilege to the end user who calls or executes the PersList application is based on two parts, a role definition and a role grant to an end user (developer). After adding the code line `application privilege: p1940xxtrial.dev.perslist::Execute;` to the role definition **user.hdbrole** (3) we call the HANA database procedure `HCP_GRANT_ROLE_TO_USER` in the HANA catalog schema „HCP", so that the **user.hdbrole** gets granted to the account user/developer (4).

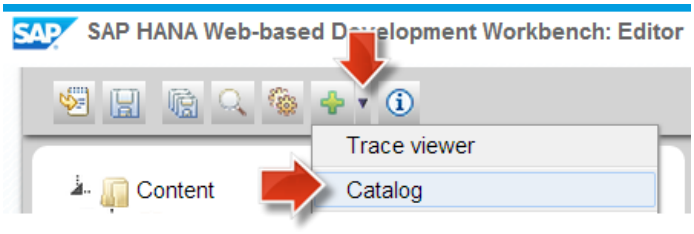
NOTE: The privilege **name** is not pre-defined by SAP HANA but can be freely chosen by the developer. The privilege **type** that is associated with a privilege at runtime is defined in the assignment of the *application* privilege to a user (with procedure `GRANT_APPLICATION_PRIVILEGE`) or to a user role (with procedure `HCP_GRANT_ROLE_TO_USER`). In the role definition file user **user.hdbrole** it is the privilege **type** name `application privilege` in code line `application privilege: p1940xxtrial.dev.perslist::Execute;` that makes the design-time privilege "*Execute*" to an *application privilege* at runtime.

Further Information: SAP HANA Developer Guide: Grant Privileges to Users, SAP HANA Cloud Platform Doc > Specific Procedures and Views: HCP_GRANT_ROLE_TO_USER()

previous next

Hands-on Tasks

71. Select the already opened Catalog browser tab. If not, press toolbar button **More** (green plus icon) in the HANA Web IDE Editor tab and select menu item **Catalog**.



72. The HANA catalog tree gets displayed in a new browser tab. Under tree node **Catalog > HCP** expand the three sub-nodes **Procedures** to display the generic HANA database procedure `HCP_GRANT_ROLE_TO_USER`.
73. Copy the SQL code string from Source Code 9 into the opened SQL editor. Replace the `highlighted string` corresponding to the name of your own trial account and user name.

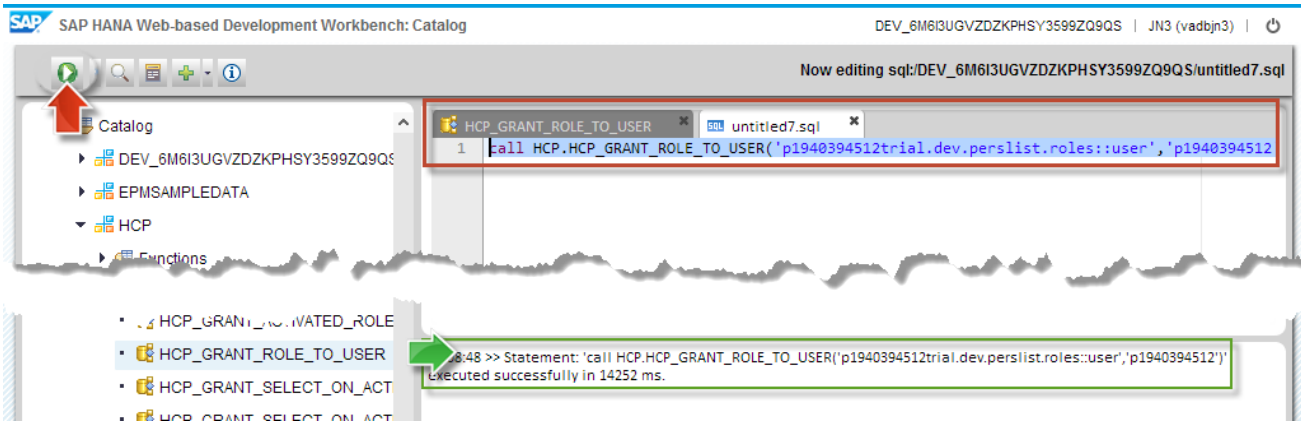
Source Code 9: Call SQL procedure HCP.HCP_GRANT_ROLE_TO_USER() to grant role user.hdbrole to your user

```
call HCP.HCP_GRANT_ROLE_TO_USER('p1940xxtrial.dev.perslist.roles::user','p1940xx')
```

74. Execute the SQL command by clicking the toolbar icon **Execute** (green icon with triangle).

Result

The content of the file **user.hdbrole** has been successfully saved and activated:



previous next

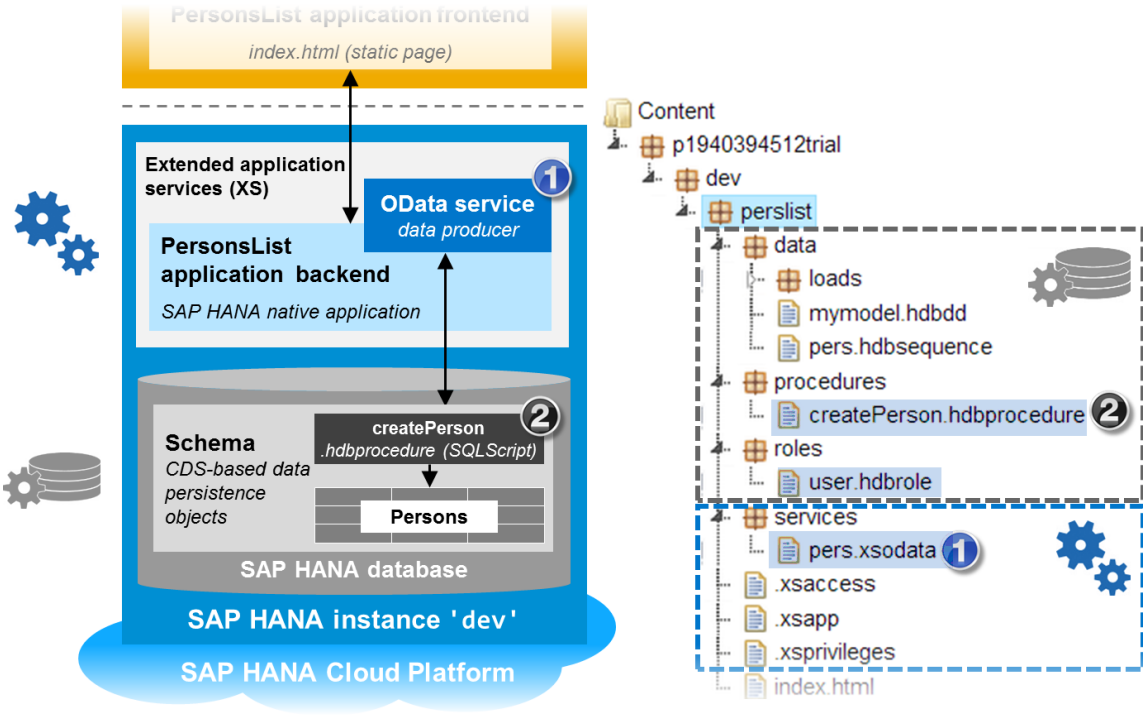
3.3 Step 3: Build the Application Backend with SAP HANA Extended Application Services

In the next step we implement the backend logic of the *PersonsList* application by using SAP HANA extended application services:

- **OData service**: exposes the Persons table in the SAP HANA database for read and write access by means of an OData service that can be consumed by the SAPUI5 application frontend.
- **HANA database procedure:** SQL Script implementation that is registered as modification exit for an OData CREATE operation (for entity *Person)*.
- **User role (modified):** enable users (with this role) to perform actions on database objects that are needed for write access to the Persons table.

Preview

Figure 8: Step 3 - application backend part with OData service and HANA database procedure for read/write logic



Design-Time Application Artifacts Created in this Step

File extension	Object	Description
.hdbprocedure	Procedure	A design-time definition of a database function for performing complex and data-intensive business logic that cannot be performed with standard SQL.
.xsodata	OData descriptor	A design-time object that defines an OData service that exposes SAP HANA data from a specified end point.

3.3.1 Expose the Person Database Entity by means of OData Service

SAP HANA extended application services provide a special tool for the creation of OData services (for web-based data access) without needing to perform server side coding. To create an OData service from an existing HANA table or view, we need only define a service definition file with suffix **.xsodata**.

previous next

Prerequisites

- Select privileges must be granted to the table to be exposed with the OData service.

Hands-on Tasks

3.3.1.1 Add new Sub-Package 'services' for OData Service Definition

To keep repository files together that are related with the application's OData services we add them to a new sub-package named **services:**

75. In editor tab select repository package **Content → p1940xxtrial → dev → perslist**.
76. Create new sub-package `services` with context menu item **Create Package**.

3.3.1.2 Create a Simple OData Service within OData Descriptor .xsodata

We want to define an OData service to expose the persons table by adding a corresponding **.xsodata** service descriptor file to the HANA repository. The syntax of the XSOData service is relative easy for this use case. We need only define …

- a namespace: our package path (**your package path**),
- the name of the SAP HANA table we will base the service from (**p1940xxtrial.dev.perslist.data::mymodel.person**)
- and the name of the OData entity (**Persons**).

77. Under new sub-package **services** add new file `pers.xsodata`.
78. Add the following code to the **pers.xsodata** file and replace the `highlighted string` with your own account name:

Source Code 10: OData service definition file pers.xsodata for service consumption of entity Person

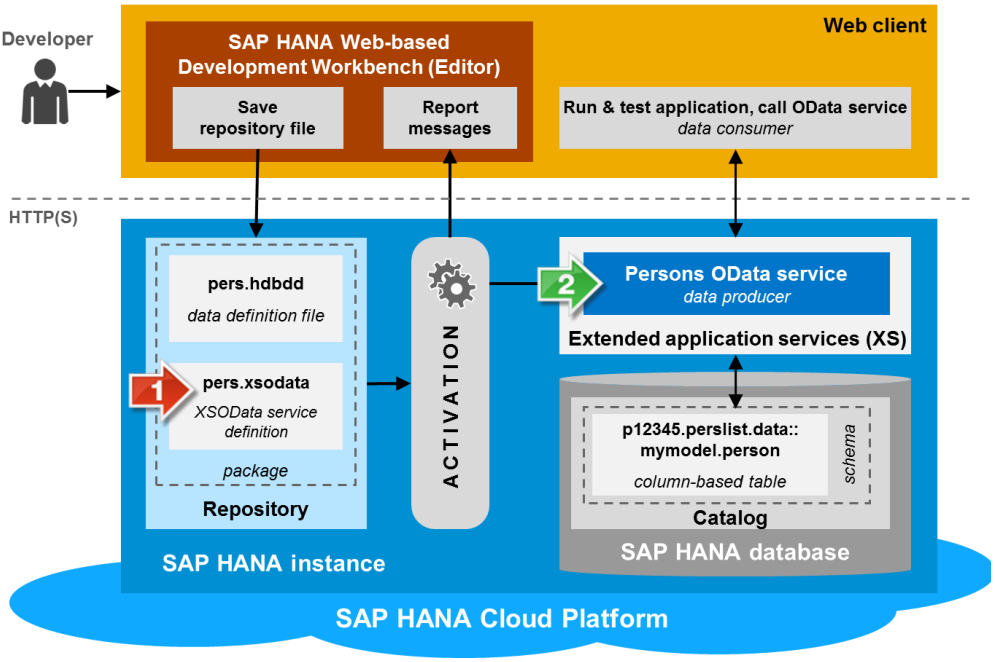
```
service {
  "p1940xxtrial.dev.perslist.data::mymodel.person" as "Persons";
}
```

79. Save the new XS-OData service descriptor.

Result

On activation of the **pers.xsodata** repository file (Figure 9, arrow 1), the file suffix, xsodata is used to determine which runtime plug-in to call during the activation process. The plug-in reads the repository file selected for activation, sees the object descriptions in the file, and creates the appropriate runtime objects, in our case the Persons OData service producer in the application backend ((Figure 9, arrow 2).

Figure 9: OData service creation based on new OData service definition file inside HANA repository



previous next

3.3.1.3 Consume and View the new XS OData Service inside Web Browser

In the next step we will consume the new *Persons* OData service by accessing the provided data in a web browser using standard HTTP. We will …

- view all resources that are exposed by the *Persons* OData service by requesting its root URI in a browser client.
- learn about the data model used by the *Persons* OData service by issuing a GET on the service's root URI with "`/$metadata`" appended to it.
- query the *Persons* table data in a browser client. To do so we issue a GET request on the corresponding OData service URI.

previous next

Prerequisites

- The repository file **pers.xsodata** that defines the *Persons* OData service in the HANA repository is successfully activated (visualized by icon  , whereas inactive/invalid repository files are marked with icon ).

Hands-on Tasks

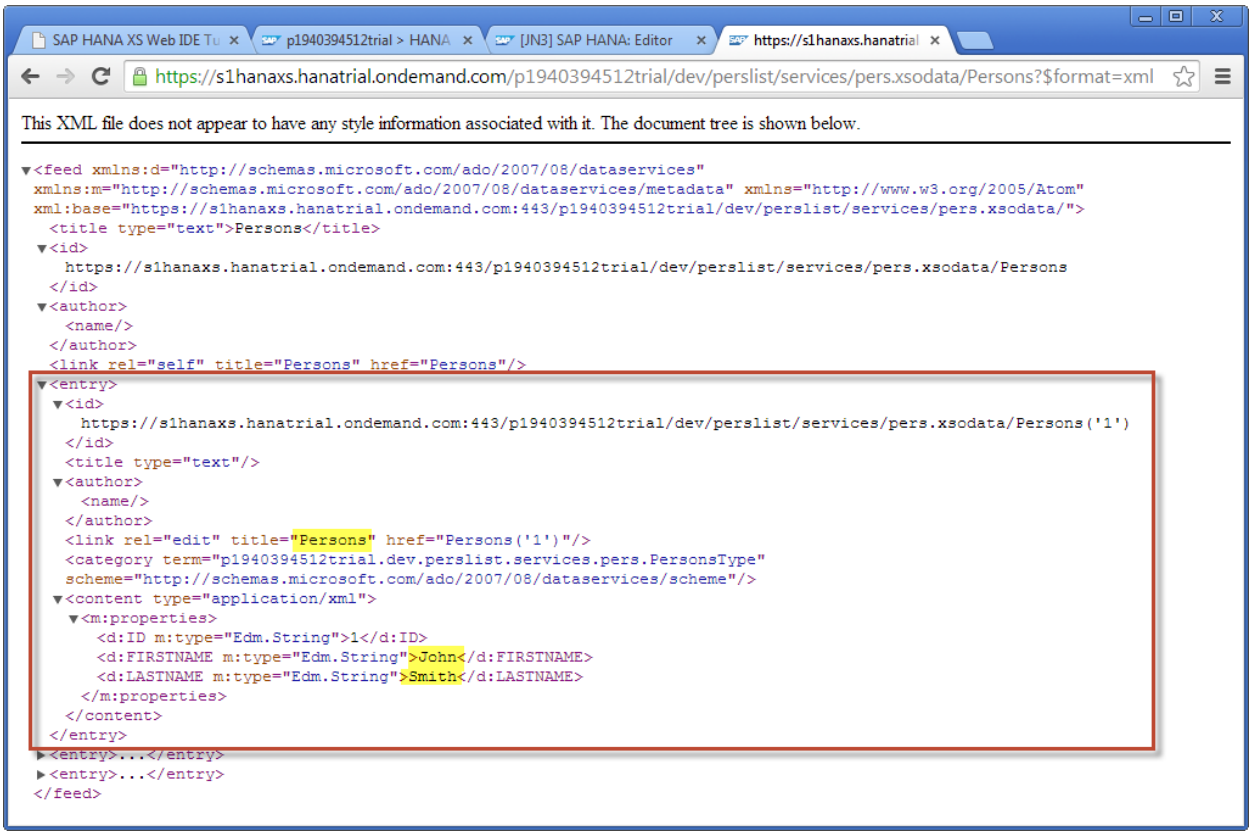
80. In the SAP HANA Web IDE Editor select the pers.xsodata file inside package **Content/p1940xxtrial/dev/perslist/services.**

81. In the toolbar click icon **Run on server**  (green icon with triangle) to request the person OData service in new browser tab. The addressed URI returns the list of resources exposed by the Persons OData service. In our simple example it is just a collection of Persons:

NOTE: When requesting the OData based URL in the JSON representation via URL parameter `format=json` `http://...perslist/services/pers.xsodata?$format=json` the response looks very simple. One "data" object (named "d") with a single name/value pair with the name equal to "*EntitySets*" and the value being an array of Collection names: `{ "d": { "EntitySets": ["Persons"] } }`.

82. Add to the browser URL the following string `/Persons?$format=xml` and press the Refresh button (or keyboard shortcut **F5**).
    An example of the response returned for the Persons query is shown below. In the below screenshot you see the first Persons record **John Smith** from all three records that were initially loaded from csv-

file into the Persons table on the HANA database.



Result

The **Person** database entity was successfully exposed as OData service by means of SAP HANA extended applications services. It can be consumed on application frontend side to list persons in a HTML5 table UI that is bound to a SAPUI5 OData model as the corresponding OData service consumer.

previous next

3.3.2 Implement Application Logic to Write new Records into Persons Table

SAP HANA XS enables you to execute custom code at defined points of an OData write request. If you provide a custom exit for an OData write request, the code has to be provided in form of a SQLScript procedure with signatures that follow specific conventions. SAP HANA XS supports two types of write exits for OData write requests:

- **Validation exits** for validation of input data and data consistency checks.
- **Modification exits** to create, update or delete an entry in an OData entry set.

3.3.2.1 Add CDS User-Defined Datatypes to be used in SQLScript Procedure

previous next

Hands-on Tasks

83. In the Editor browser tab (of SAP HANA Web-based Development Workbench) select the inner editor tab for the **mymodel.hdbdd** file. In case you closed it before click on the repository content node **Content → p1940xxtrial → dev → perslist → data → mymodel.hdbdd** to open the editor tab.

84. Add the `highlighted code` with new context "*procedures*" and new CDS user-defined datatypes ***pers*** and ***errors*** to the CDS data definition file **mymodel.hdbdd**.

Copy Source Code 11 from next page …

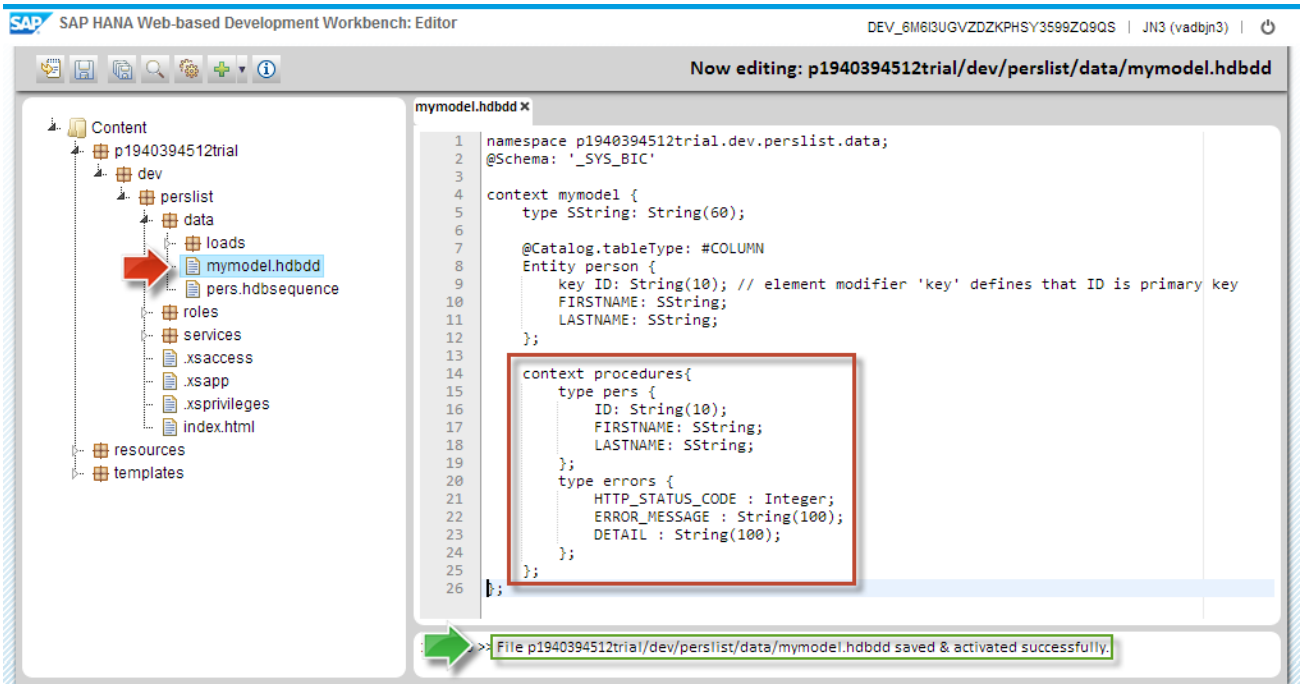
Source Code 11: New context 'procedures' with CDS-user-defined datatypes pers and errors in CDS Data Definition File mymodel.hdbdd

```
namespace p1940xxtrial.dev.perslist.data;
@Schema: '_SYS_BIC'

context mymodel {
    type SString: String(60);

    @Catalog.tableType: #COLUMN

    Entity person {
        key ID: String(10); // element modifier 'key' defines that ID is primary key
        FIRSTNAME: SString;
        LASTNAME: SString;
    };

    context procedures{
        type pers {
            ID: String(10);
            FIRSTNAME: SString;
            LASTNAME: SString;
        };
        type errors {
            HTTP_STATUS_CODE : Integer;
            ERROR_MESSAGE : String(100);
            DETAIL : String(100);
        };
    };
};
```

85. Save the newly edited data definition file **mymodel.hdbdd** and take care that it gets activated successfully.

Result

The new datatypes **pers** and **errors** can be used as parameter types in SQLScript procedure **createPerson.hdbprocedure** that gets implemented in the next hands-on task.



3.3.2.2 Implement Modification Exit in SQLScript Procedure to add new Records in Persons Table

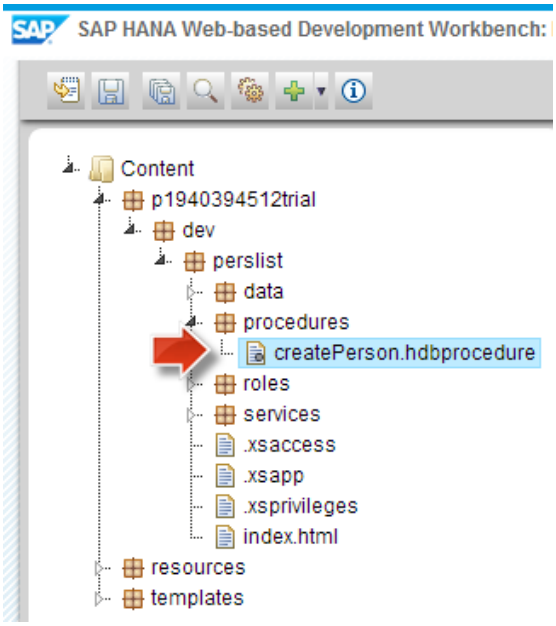
Create a new SQLScript procedure that runs as modification exit before the create event in the OData service.

Hands-on Tasks

86. In editor tab select repository package **Content → p1940xxtrial → dev → perslist**

87. Create new sub-package **`procedures`** with context menu item ***Create Package***

88. Under new sub-package procedures add new file **`createPerson.hdbprocedure`**



89. Copy Source Code 12 with the procedure logic to insert new records into the Persons table:

Source Code 12: SQLScript procedure createPerson.hdbprocedure to insert table records (via modification exit for OData service)

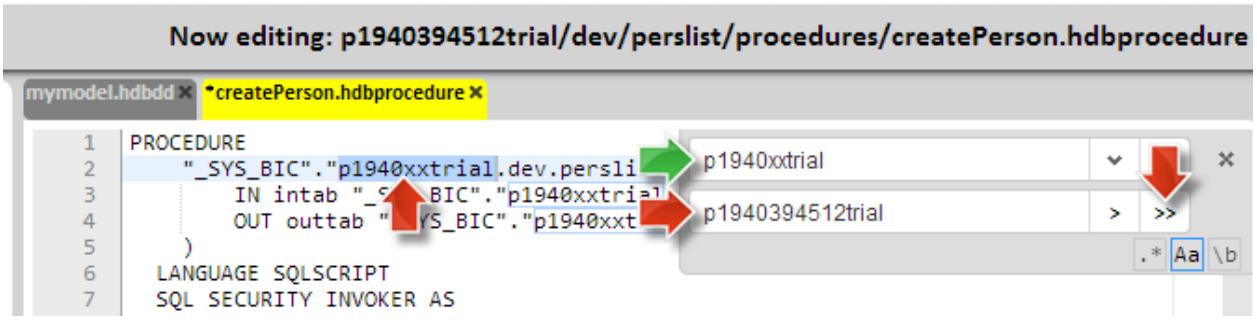
```
PROCEDURE
    "_SYS_BIC"."p1940xxtrial.dev.perslist.procedures::createPerson" ( 1
        IN intab "_SYS_BIC"."p1940xxtrial.dev.perslist.data::mymodel.procedures.pers",
        OUT outtab "_SYS_BIC"."p1940xxtrial.dev.perslist.data::mymodel.procedures.errors" 2
    )
  LANGUAGE SQLSCRIPT
  SQL SECURITY INVOKER AS
  --DEFAULT SCHEMA <schema>
  --READS SQL DATA AS
begin

declare lv_pers_no string;
declare lv_firstname string;
declare lv_lastname string;

select ID, FIRSTNAME, LASTNAME into lv_pers_no, lv_firstname, lv_lastname from :intab;

if :lv_lastname = '' then
  outtab = select 500 as http_status_code,
              'Invalid last name ' || lv_firstname as error_message,
                    'No Way! Last name field must not be empty' as detail from dummy;
else
   insert into "p1940xxtrial.dev.perslist.data::mymodel.person" 3
         values ("p1940xxtrial.dev.perslist.data::pers".NEXTVAL, lv_firstname, lv_lastname); 4
end if;
```

```
end;
```

NOTE: Know the code

1. Fully qualified name of SQLScript procedure `createPerson` starts with schema name `"_SYS_BIC"` followed by repository package name `"p1940xxtrial.dev.perslist.procedures"`.
2. Reference to fully qualified types `pers` and `error` defined in CDS file **mymodel.hdbd** (with `"::mymodel"` = context name, `".procedures"` = inner context name, see Source Code 11).
3. Reference to entity `"p1940xxtrial.dev.perslist.data::mymodel.person"` defined in CDS file **mymodel.hdbd**.
4. Reference to database sequence `"p1940xxtrial.dev.perslist.data::pers"` defined in file **pers.hdbsequence**, see Source Code 7.

90. To replace the corresponding user string **`p1940xxtrial`** (that is part of the namespaces and the schema to reference HANA objects in the procedure) make use of search and replace function in the HANA Web IDE as follows:



In the editor tab **createPerson.hdbprocedure** proceed as follows

- Mark string **`p1940xxtrial`**.
- Press keyboard shortcut **Ctrl+H** so that the search & replace dialog opens in the upper right corner of the editor.
- The first input field contains the selected string **`p1940xxtrial`**. Enter your trial account name e.g. **`p1940394512trial`** into the second input field.
- Press **Aa** button to assure case sensitive search (all matching strings get highlighted with blue rectangles)
- Press **>>** button to replace all occurrences of the search string **`p1940xxtrial`**.
- Close the popup dialog.

The table and error type objects you created in the previous steps are used as types in the procedure created here. The procedure also performs verification on the data and inserts a new row with error information into the output table (Persons).

91. Save the new SQLScript procedure to activate it.

Result

All strings **`p1940xxtrial`** are replaced with correct user names and the HANA database procedure **createPerson.hdbprocedure** is activated in the repository.

previous next

3.3.2.3 Register a Modification Exit for OData Write Requests

Register the new SQLScript procedure **createPerson.hdbprocedure** in the OData service **pers.xsodata** to be executed at the CREATE event.

previous next

Hands-on Tasks

92. Open the **pers.xsodata** file in the editor and use the **`create using`** keywords to register the new HANA database procedure, as illustrated with the **new line** `create using …` in the OData service file.

Source Code 13: Add modification exit to call SQLScript procedure createPerson.hdbprocedure in OData service

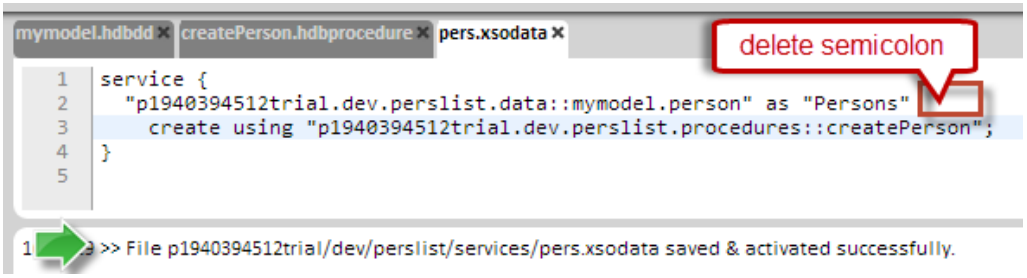
```
service {
  "p1940xxtrial.dev.perslist.data::mymodel.person" as "Persons"
    create using "p1940xxtrial.dev.perslist.procedures::createPerson";
}
```

add this code line

93. **Remove the semi-colon** at the end of the existing code line the service definition file.
94. Then copy the bold marked line **`create using …`** and paste into the **pers.xsodata** file before the closing curly bracket.
95. In the new codeline replace **`p1940xxtrial`** with your user name.
96. Save **pers.xsodata** so that it is activated with the additional code line.

Result

By this the OData service can invoke the HANA database procedure **createPerson.hdbprocedure** during the CREATE event.



previous next

3.3.3 Modify Role Definition in File user.hdbrole and Synchronize Roles

previous next

Hands-on Tasks

97. Open the **user.hdbrole** file in repository package **p1940xxtrial/dev/perslist/roles**.
98. In the opened editor tab of file **user.role** modify Source Code 8 by adding two object privileges to execute the newly created SQLScript procedure **createPerson.hdbprocedure** (1) and to select entries from the database sequence table defined in repository file **pers.hdbsequence** (2). Replace the `highlighted string` with your own account name:

Source Code 14: Role definition file user.hdbrole

```
role p1940xxtrial.dev.perslist.roles::user {
    sql object p1940xxtrial.dev.perslist.data::mymodel.person: SELECT, INSERT, UPDATE,
DELETE;

    sql object p1940xxtrial.dev.perslist.procedures::createPerson: EXECUTE; (1)
    sql object p1940xxtrial.dev.perslist.data:pers.hdbsequence: SELECT; (2)

    application privilege: p1940xxtrial.dev.perslist::Execute;
```

```
}
```

NOTE:

Object privileges on the design-time (repository) objects views, procedures and others are defined using the keyword `sql object` as follows: `sql object <package>:<object>.extension: PRIVILEGE;`.

Object privileges on the catalog objects are defined using the keyword `catalog SQL object` as follows: `catalog sql object "SCHEMA"."CATALOG_OBJECT": PRIVILEGE;` For security reasons, system privileges, object privileges on schema level and package privileges are not allowed to be granted to activated **hdbroles** when using a Trial SAP HANA instance.

99. Save **pers.xsodata** and check whether the user.hdbrole file gets activated successfully.

100. Switch to the **SAP HANA: Catalog** browser tab that you opened before in section 3.2.3. In case you have closed the tab open it again by pressing the editor toolbar button **More** and selecting menu item **Catalog.**

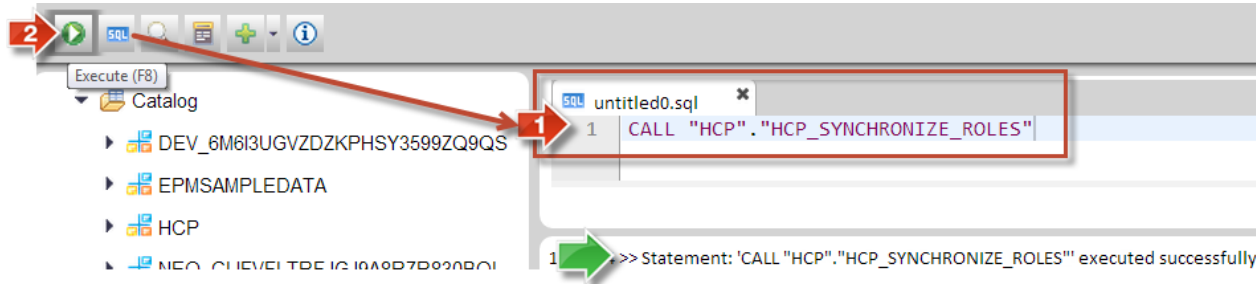
101. Open a SQL console by clicking the toolbar icon **Open New SQL Editor** (blue SQL icon).

102. Copy the SQL code string from Source Code 4 into the opened SQL Editor:

Source Code 15: SQL procedure call HCP_SYNCHRONIZE_ROLES() to synchronize the developer's role with the activated hdbroles from the developer's package

```
CALL "HCP"."HCP_SYNCHRONIZE_ROLES"
```

103. Execute the SQL command by clicking the toolbar icon **Execute** (green icon with triangle).



Result

The content of file **user.hdbrole** has been successfully saved and activated. By calling the SQL procedure `HCP_SYNCHRONIZE_ROLES()` provided in schema HCP the developer's role (i.e. your user role) privileges are updated with the privileges from the activated **user.hdbrole** file. All users with assigned role **user.hdbrole** are now privileged to write new person entries back to the person table by means of the SQLScript procedure (modification exit) **createPerson.hdbprocedure** and the database sequence **pers.hdbsequence**.

Further Information:

- SAP HANA Developer Guide: Object Privileges
- SAP HANA Cloud Platform Doc > Specific Procedures and Views: HCP_SYNCHRONIZE_ROLES()

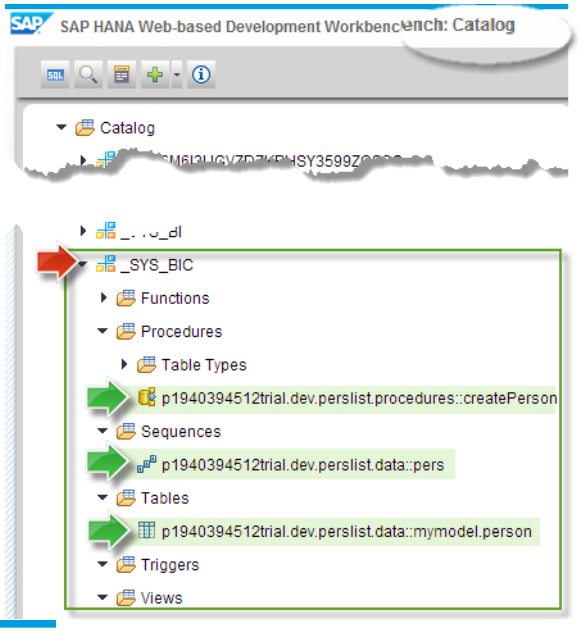
previous next

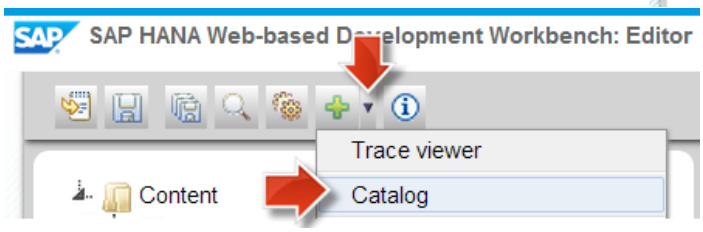
3.3.4 View the Application Backend in SAP HANA Catalog

previous next

Hands-on Tasks

104. Select the already opened Catalog browser tab. If not, press toolbar button **More** and select menu item **Catalog.**



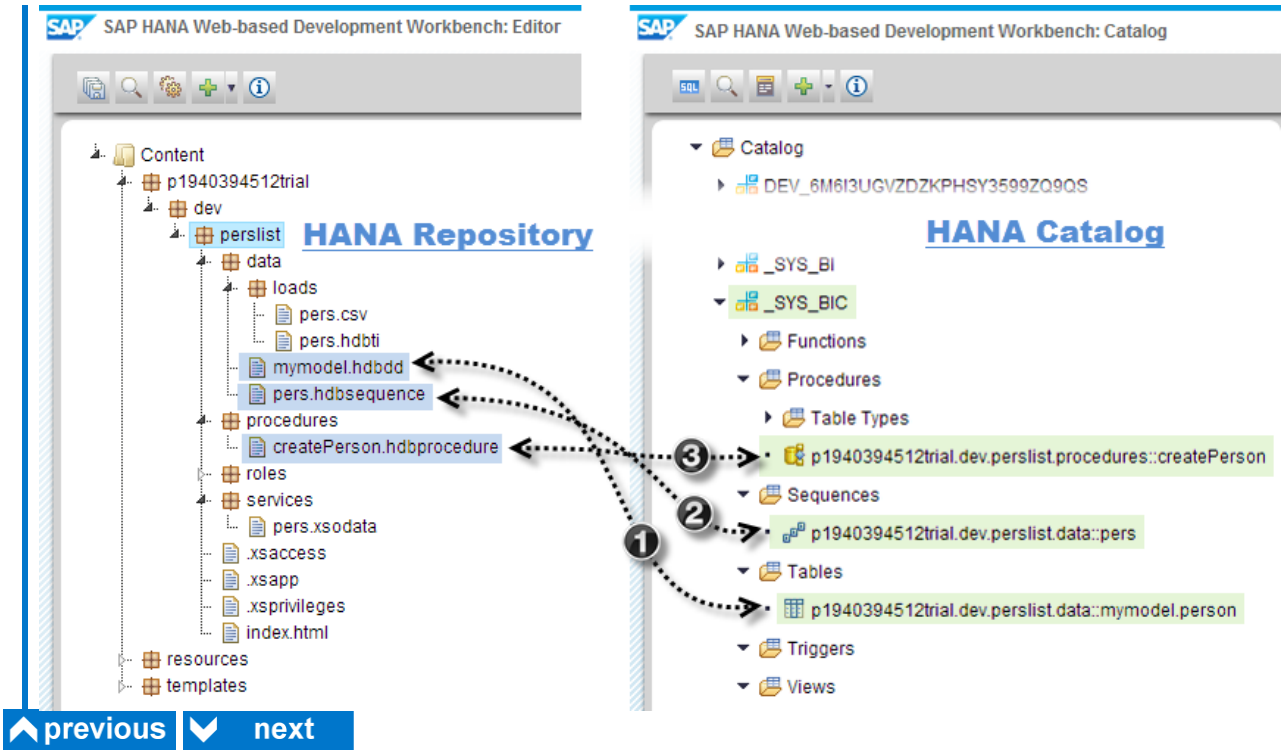


105. The HANA catalog tree gets displayed in a new browser tab. Under tree node **Catalog > _SYS_BIC** *(*= database schema where the runtime artifacts of your *PersonsList* application reside) for expand the three sub-nodes **Procedures**, **Sequences** and **Tables.**

Result

In database schema **`_SYS_BIC`** you see the SAP HANA runtime artifacts that were automatically added to the SAP HANA catalog based on the corresponding design-time artifacts you added before in the SAP HANA repository:

	Repository content	Catalog content
1	CDS-based data definition file `p1940xxtrial/dev/perslist/data/`**`mymodel.hdbdd`**	`_SYS_BIC/Tables/`**`p1940xxtrial.dev.perslist.data::mymodel.person`**
2	Sequence definition file `p1940xxtrial/dev/perslist/data/`**`pers.hdbsequence`**	`_SYS_BIC/Sequences/`**`p1940xxtrial.dev.perslist.data::pers`**
3	SAP HANA database procedure file `p1940xxtrial/dev/perslist/procedures/`**`createPerson.hdbprocedure`**	`_SYS_BIC/Tables/`**`p1940xxtrial.dev.perslist.procedures::createPerson`**



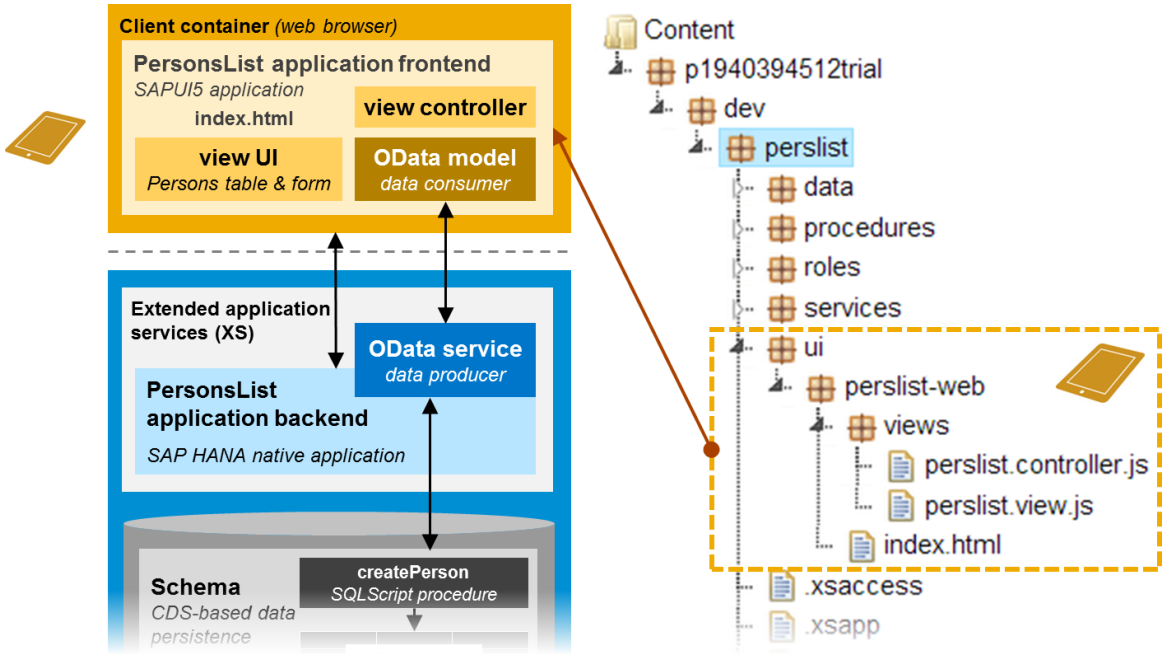
previous next

3.4 Step 4: Build the Application Frontend with SAPUI5

After having fully implemented the application backend you build the application frontend on top by using the UI development toolkit for HTML5 (aka SAPUI5).

Preview

Figure 10: Step 4 - application frontend part with SAPUI5 view UI, controller and OData consumption



Design-Time Application Artifacts Created in this Step

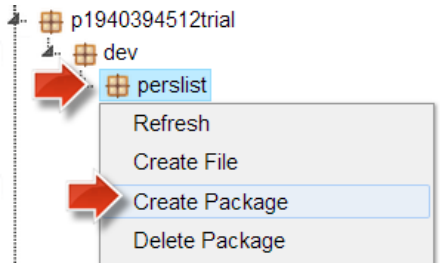
File extension	Object	Description
index.html	SAPUI5 application entry page	HTML file containing bootstrap script (to load SAPUI5 libraries and theme at the client), application script (to load the application's view layout and controller logic) and html body (to embed views).
.view.js	SAPUI5 JavaScript view (JSView)	A SAPUI5 view flavor that allows you to use JavaScript (JS) code to construct a view.
.controller.js	SAPUI5 JavaScript controller	A SAPUI5 application unit containing the active part of the application. It is the logical interface between a model and a view, corresponding to the model view controller (MVC) concept. A controller reacts to view events and user interaction by modifying view and model.

3.4.1 Create Package Structure for SAPUI5 Application Content

previous next

Hands-on Tasks

106. In editor tab select repository package **Content → p1940xxtrial → dev → perslist**

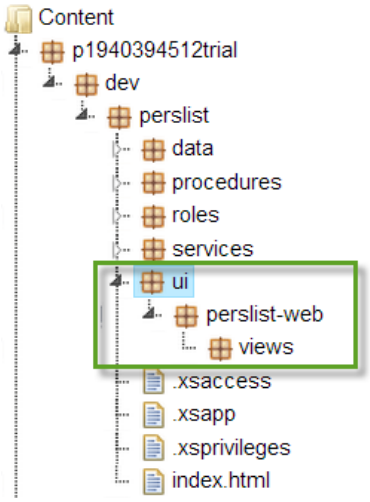


107. Create new sub-packages **ui**, **perslist-web** and **views** by choosing context menu item **Create Package**

108. Enter `ui.perslist-web.views` and press RETURN.

Result

The SAPUI5 application sources that are implemented in the next step can be added to the newly created subfolders **/ui/perslist-web** (for *index.hmtl*) and **ui/perslist-web/views** (for *view.js* and *controller.js*).



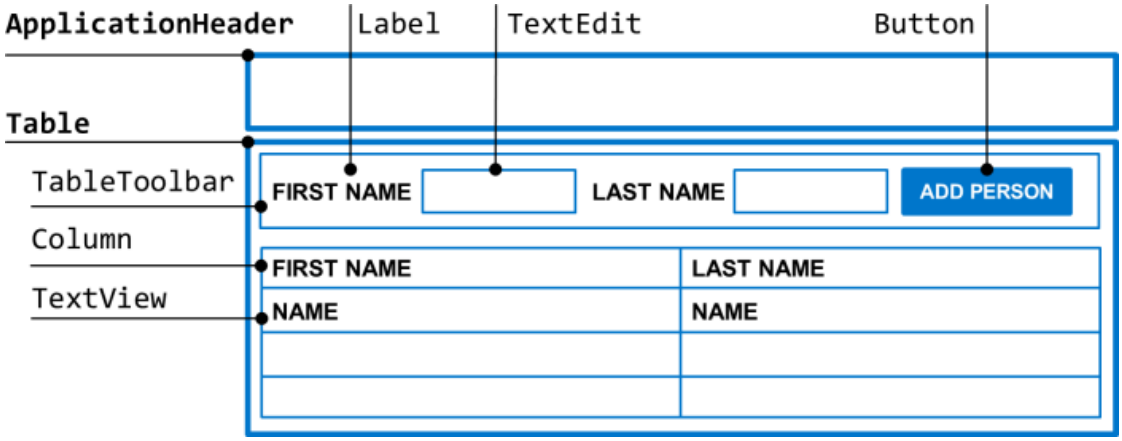
previous next

3.4.2 Design and Implement Application UI in SAPUI5 JavaScriptView

Prototype a User Interface

Before we implement the new JavaScriptView in SAPUI5 let's first have a look at the simple user interface with its control tree.

Figure 11: User interface sketch diagram with Persons entry form and table control



User Interaction Should Look Like This

1. User enters a person's first name and last name in the TextEdit controls with the TableToolbar.
2. User then clicks the Add Person Button (the persons data is submitted to the controller which adds it to the OData model) in the TableToolbar.
3. Entered person name is taken from the model and displayed in a table control.

Further information: SAPUI5 Demo Kit - Control Gallery

previous next

Hands-on Tasks

109. Select package **p1940xxtrial/dev/perslist/ui/perslist-web/views** and choose context menu item **Create File**

110. Enter new file name **`perslist.view.js`** (UI5 JavaScriptView for the view layout implementation)

111. Copy Source Code 16 into the **perslist.view.js** editor:

Source Code 16: SAPUI5 application UI implemented in JavaScriptView file perslist.view.js

```
sap.ui.jsview("views.perslist", {

  oFirstNameField : null,
  oLastNameField : null,

  getControllerName : function() {
    return "views.perslist";
  },

  createContent : function(oController) { // Create an instance of the table control
    var oTable = new sap.ui.table.Table({ title : "Persons List", visibleRowCount : 6,
      firstVisibleRow : 3, selectionMode : sap.ui.table.SelectionMode.Single, });

    // add TableToolbar
    var oTableToolbar = new sap.ui.commons.Toolbar();

    // add first name field
    var oFirstNameLabel = new sap.ui.commons.Label({ text : 'First Name' });
    this.oFirstNameField = new sap.ui.commons.TextField({ id : 'firstNameFieldId', value : '',
      width : '10em', });
    oFirstNameLabel.setLabelFor(this.oFirstNameField);
    oTableToolbar.addItem(oFirstNameLabel);
    oTableToolbar.addItem(this.oFirstNameField);

    // add last name field
    var oLastNameLabel = new sap.ui.commons.Label({ text : 'Last Name' });
    this.oLastNameField = new sap.ui.commons.TextField({ id : 'lastNameFieldId', value : '',
      width : '10em', });
    oLastNameLabel.setLabelFor(this.oLastNameField);
    oTableToolbar.addItem(oLastNameLabel);
    oTableToolbar.addItem(this.oLastNameField);

    // add button
    var oAddPersonButton = new sap.ui.commons.Button({ id : 'addPersonButtonId',
      text : "Add Person", press : function() {
        oController.addNewPerson();
      } });
    oTableToolbar.addItem(oAddPersonButton);
    oTable.setToolbar(oTableToolbar);

    // define the columns and the control templates to be used
    oTable.addColumn(new sap.ui.table.Column({
      label : new sap.ui.commons.Label({ text : "First Name" }),
      template : new sap.ui.commons.TextView().bindProperty("text", "FIRSTNAME"),
      sortProperty : "FIRSTNAME", filterProperty : "FIRSTNAME", width : "100px" }));
    oTable.addColumn(new sap.ui.table.Column({
      label : new sap.ui.commons.Label({ text : "Last Name" }),
      template : new sap.ui.commons.TextView().bindProperty("text", "LASTNAME"),
      sortProperty : "LASTNAME", filterProperty : "LASTNAME", width : "200px" }));

    // bind table rows to /Persons based on the model defined in the init method of the
    // controller (aggregation binding)
    oTable.bindRows("/Persons");

    return oTable;
  },

  getFirstNameField : function() {
    return this.oFirstNameField;
  },

  getLastNameField : function() {
    return this.oLastNameField;
  },
```

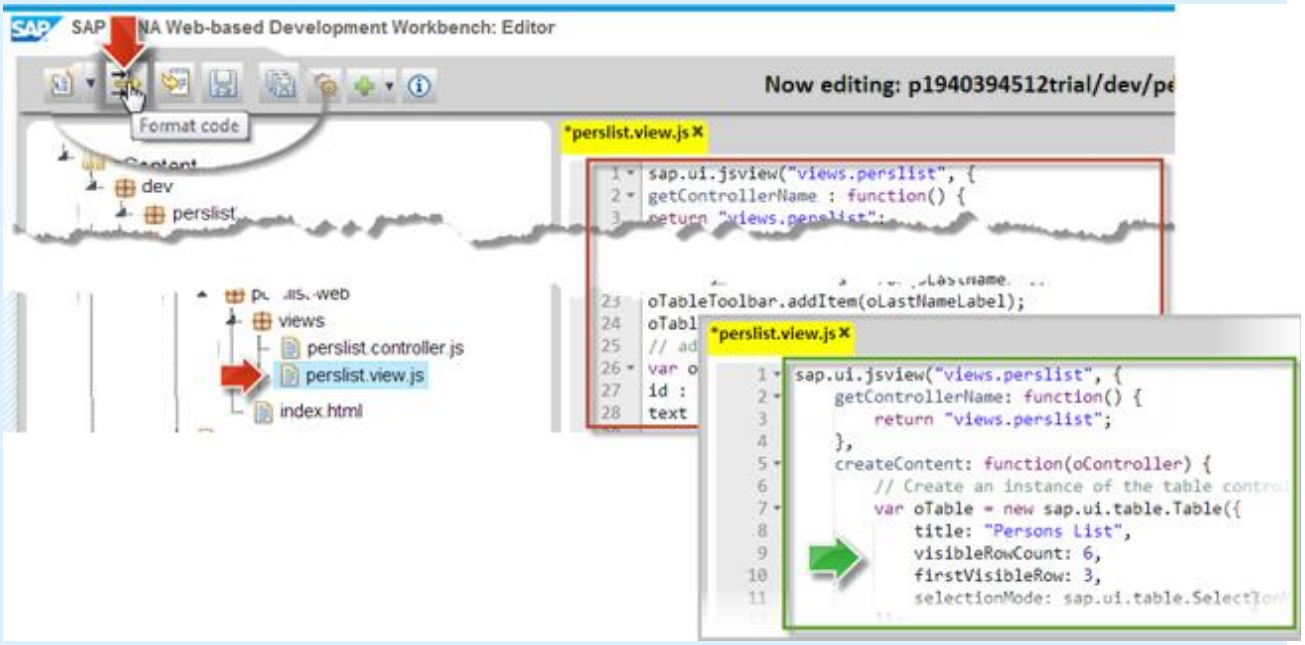
```
});
```

NOTE: Know the code

1. **Property binding**: add new *Column* control to the table and bind the *TextView* control property `"text"` to the OData model property `"FIRSTNAME"`.
2. **Aggregation binding:** aggregation binding is used to bind a collection of table rows with data from the OData model to the table. The absolute binding path `"/Persons"` points to the entity set with name `"Persons"` defined in our XSOData service. The binding paths in an OData model reflect the structure of the OData service as exposed through the `$metadata` of the service.

NOTE: How to Format Code within HANA Web IDE

After you copied the file into the created **perslist.view.js** editor of the HANA Web IDE, content is not formatted because copied .pdf-content in general cannot do this. To format source code inside the HANA Web IDE press button **"Format code"**  in the toolbar:



INFO: The "Format code" feature of the HANA Web IDE so far only works for **`.js`** , **`.xsjs`** and **`.xsjslib`** (where the last two file types are not explained in the tutorial).

112. Save the **perslist.view.js** file with **Ctrl+S** or toolbar button **Save** to activate it.

Result

The SAPUI5 **JSView perslist.view.js** defining the UI controls of the PersonsList application has been created.

Further information:

- SAPUI5 Demo Kit - Control Gallery - Application Header, Table Example (apply "Show Source Code" function and SAPUI5 Demo Kit - Control Gallery – Introduction to Data Binding

previous next

3.4.3 Create SAPUI5 View Controller with SAPUI5 OData Model

previous next

Hands-on Tasks

113. Select package **p1940xxtrial/dev/perslist/ui/perslist-web/views** and choose context menu item **Create File**.

114. Enter new file name **`perslist.controller.js`** (view controller logic implemented in JavaScript).

115. Copy Source Code 17 into the **perslist.controller.js** editor. Replace the `highlighted string` with your own account name:

Source Code 17: SAPUI5 view controller JS file to call OData service exposed by pers.xsodata file

```
sap.ui.controller("views.perslist", {

  onInit : function() {
    var sOrigin = window.location.protocol + "//" + window.location.hostname
       + (window.location.port ? ":" + window.location.port : "");
    var persListOdataServiceUrl = sOrigin
       + "/p1940xxtrial/dev/perslist/services/pers.xsodata"; (1)
    var odataModel = new sap.ui.model.odata.ODataModel(persListOdataServiceUrl); (2)
    this.getView().setModel(odataModel); (3)
  },

  addNewPerson : function() {
    var firstName = this.getView().getFirstNameField().getValue();
    var lastName = this.getView().getLastNameField().getValue();

    var persons = {};
    persons.ID = "1";
    persons.FIRSTNAME = firstName;
    persons.LASTNAME = lastName;
    this.getView().getModel()
       .create("/Persons", persons, null, this.successMsg, this.errorMsg); (4)
  },

  successMsg : function() {
    sap.ui.commons.MessageBox.alert("Person entity has been successfully created");
  },

  errorMsg : function() {
    sap.ui.commons.MessageBox.alert("Error occured when creating person entity");
  },

  onAfterRendering : function() {
    this.getView().getFirstNameField().focus(); } (5)
});
```

116. Replace string **`p1940xxtrial`** with your own user name.

117. Save the **perslist.controller.js** file with **Ctrl+S** or toolbar button **Save**.

NOTE: Know the code

(1) Generate absolute URL of OData service `pers.xsodata` added in section 3.3.1.

(2) **OData model instance creation**: to use data binding in a SAPUI5 applications we instantiate the OData model first. The constructor takes the URL of the model data or the data itself as the first parameter.

(3) **OData model instance assignment** to view controller `"views.perslist"` for data binding in the view's UI elements.

(4) OData model write access to create a new Person records.

(5) After rendering of the view layout the initial focus is set to the first input field so that the user can start typing.

Result

The SAPUI5 controller perslist.controller.js has been created and enables the control logic between the client side SAPUI5 OData model and the HANA backend according to some UI interaction (button click).

Further information:

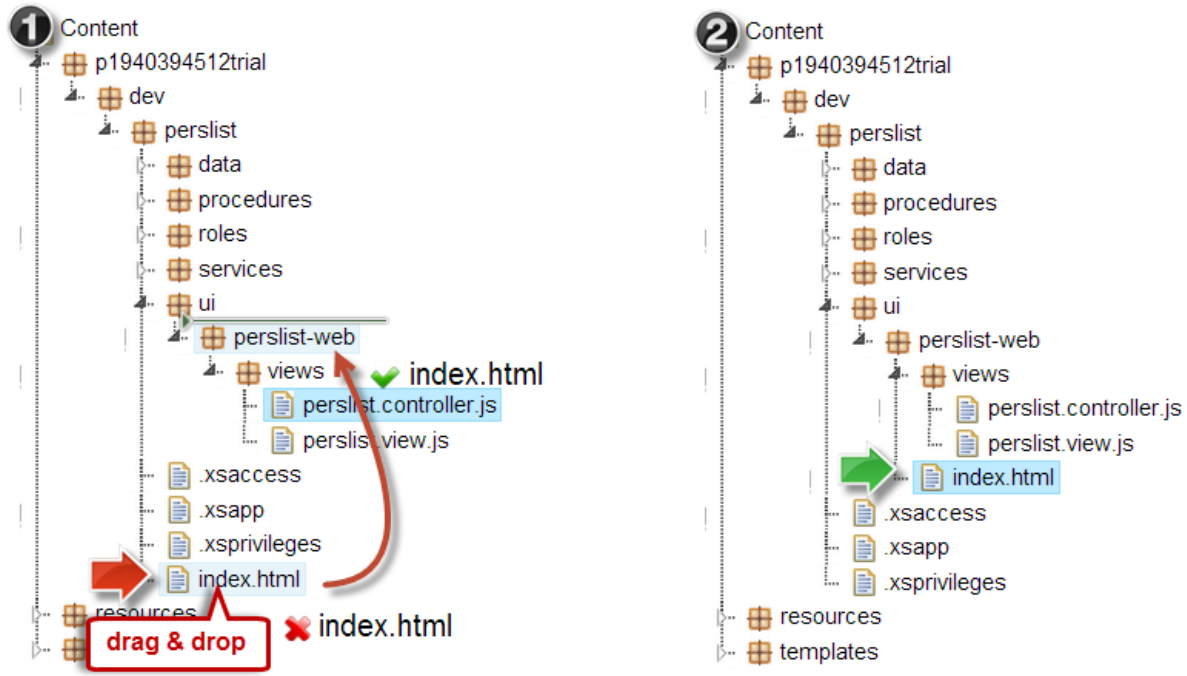
- SAPUI5 SAPUI5 Demo Kit - Developer Guide - The Model-View-Controller Approach
- SAPUI5 Demo Kit - Developer Guide - OData Model

3.4.4 Implement index.html file with the SAPUI5 Application Bootstrap and Content

Hands-on Tasks

118. We reuse the **index.html** file of our initially created blank HANA XS application to define the entry point for our SAPUI5 PersonsList application UI.

119. Select the **index.html** file under package **p1940xxtrial/dev/perslist** and move it to the package location **p1940xxtrial/dev/perslist/ui/perslist-web** using drag & drop:



120. After the drag & drop operation the **index.html** editor is opened. In the editor tab remove the entire content and replace it with Source Code 18:

Copy Source Code 18 from next page …

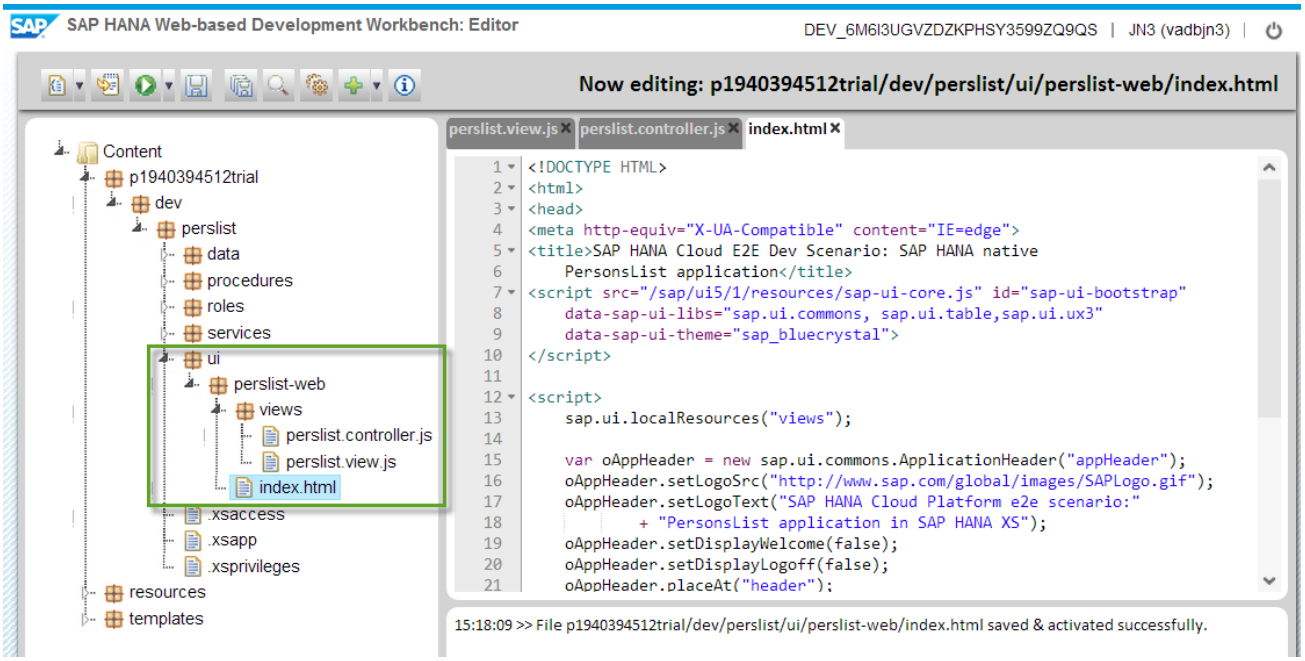
Source Code 18: index.html with SAPUI5 application bootstrap and html content

```
<!DOCTYPE HTML>
<html>
<head>
<meta http-equiv="X-UA-Compatible" content="IE=edge">
<title>SAP HANA Cloud E2E Dev Scenario: SAP HANA native
  PersonsList application</title>
<script src="/sap/ui5/1/resources/sap-ui-core.js" id="sap-ui-bootstrap"
  data-sap-ui-libs="sap.ui.commons, sap.ui.table,sap.ui.ux3"
  data-sap-ui-theme="sap_bluecrystal">
</script>

<script>
  sap.ui.localResources("views");

  var oAppHeader = new sap.ui.commons.ApplicationHeader("appHeader");
  oAppHeader.setLogoSrc("http://www.sap.com/global/images/SAPLogo.gif");
  oAppHeader.setLogoText("SAP HANA Cloud Platform e2e scenario:"
     + "PersonsList application in SAP HANA XS");
  oAppHeader.setDisplayWelcome(false);
  oAppHeader.setDisplayLogoff(false);
  oAppHeader.placeAt("header");

  var view = sap.ui.view({ id : "perslist-id", viewName : "views.perslist",
    type : sap.ui.core.mvc.ViewType.JS });
  view.placeAt("content");
</script>

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

121. Save **index.html** to activate it.

Result

The SAP HANA native *PersonsList* application is now fully implemented and can be launched and tested in a new browser tab.



Further information:

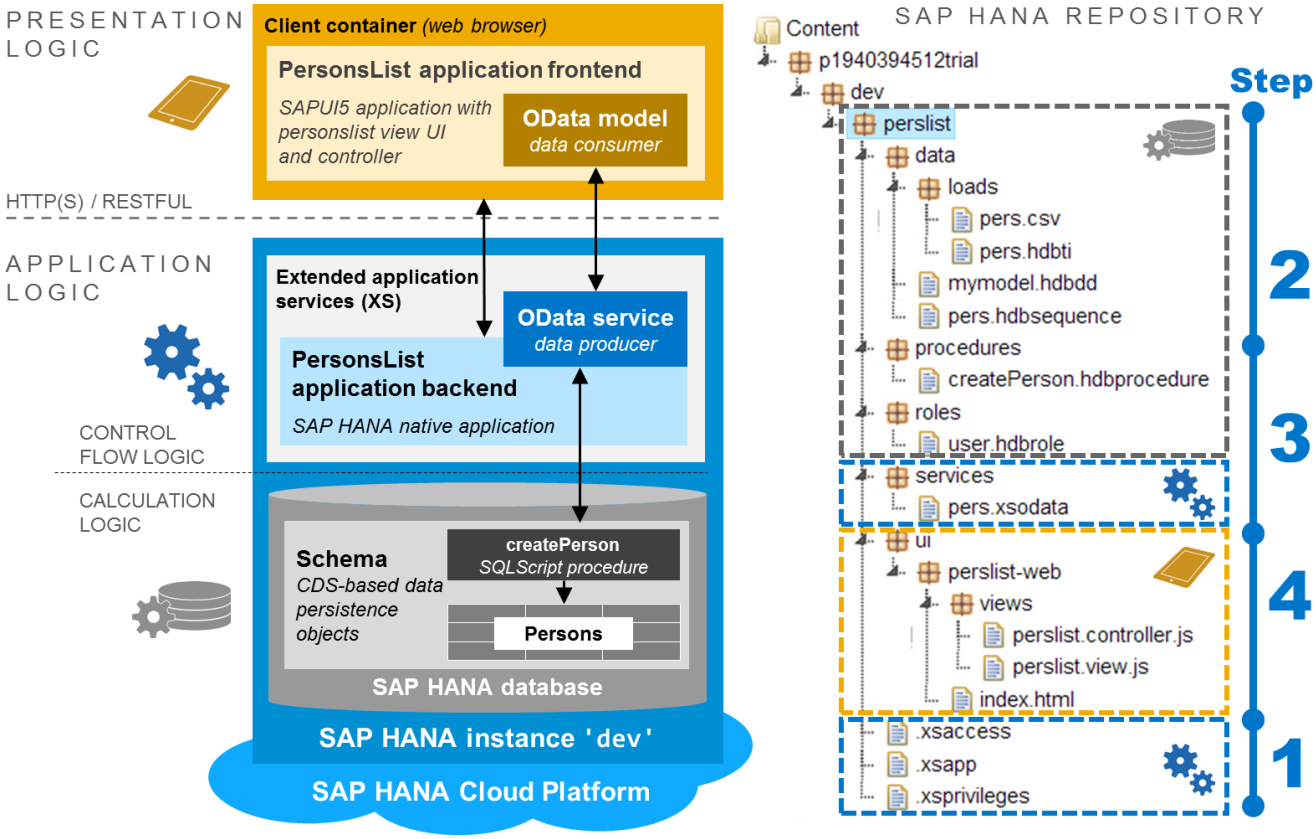
- SAPUI5 Demo Kit - Developer Guide - SAPUI5 HelloWorld
- SAPUI5 Demo Kit - Developer Guide - Bootstrapping

previous next

3.5 Run and Test the Final PersonsList Application

3.5.1 Recap: Anatomy of the Whole PersonsList Application

Figure 12: Anatomy of the whole PersonsList application implemented with SAP HANA extended application services



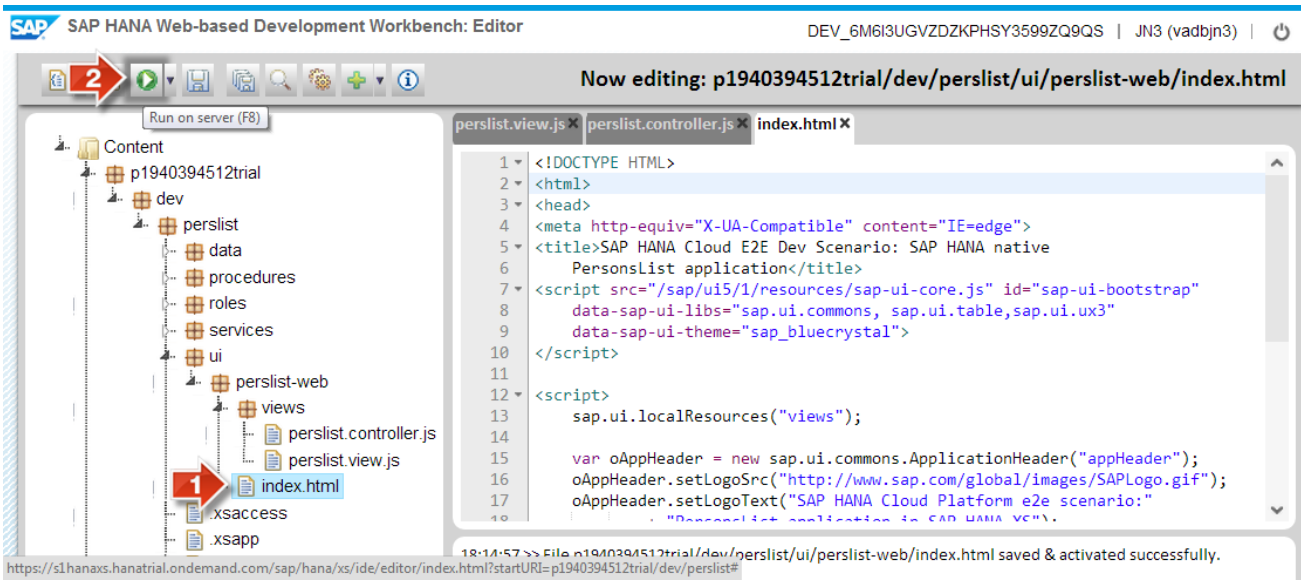
3.5.2 Run PersonsList Application on SAP HANA XS Server

previous next

Hands-on Tasks

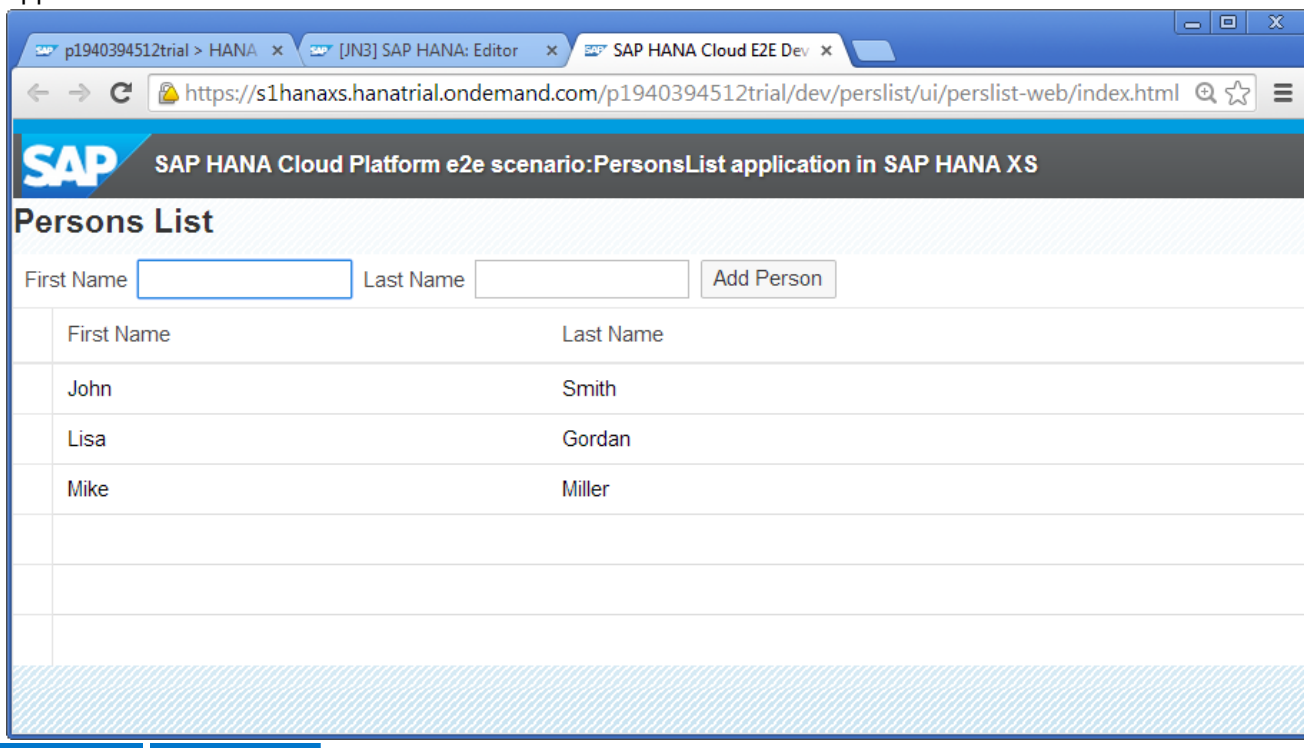
122. In editor tab select repository package **Content → p1940xxtrial → dev → perslist → ui → perslist-web → index.html**.

123. To run the application frontend in a new browser tab press toolbar button **"Run on server (F8)"**.



Result

The *PersonsList* application frontend is launched in a new browser tab. The "First Name" input field is already focused for user input and the "Person List" table displays three records that were fetched from the application backend via XS-OData service.



previous next

3.5.3 Test PersonsList Application in Web Browser

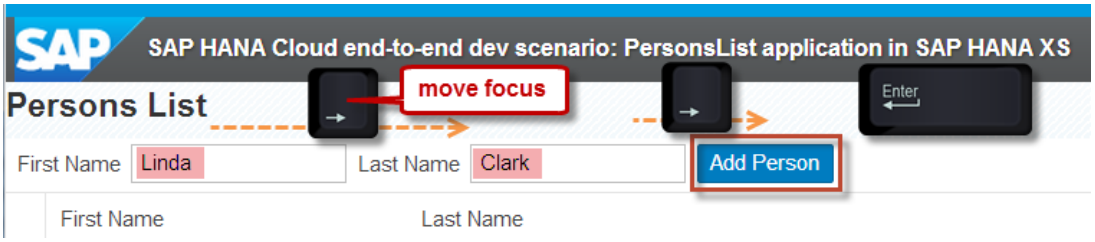
To test the XSOData write access we enter first and last name of a new Person named Linda Clark into the form fields and then press the *Add Person* button.

previous next

Hands-on Tasks

124. Enter first name **`Linda`** into the first input field that is already focused on application startup.

125. Press the *right arrow* keyboard button to move focus to the next input field '*Last Name*'.



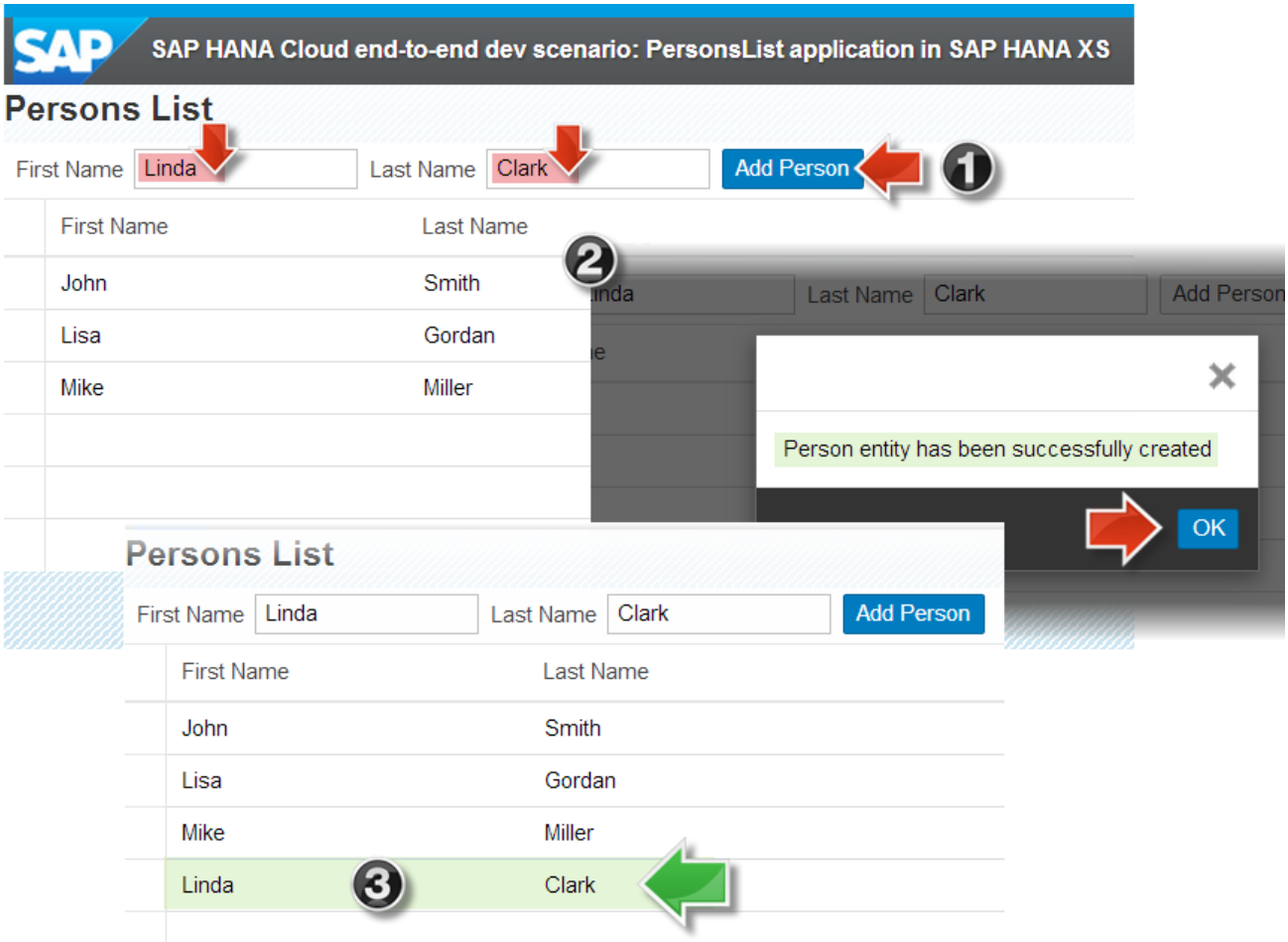
NOTE: Inside a SAPUI5 form (or container) you move focus with **keyboard arrow buttons** (right arrow keyboard button for next form field or left arrow keyboard button for previous form field) but not with the TAB button. The TAB keyboard button is used to move focus between container controls (like from the form in the table toolbar to the embedding table).

126. Enter **`Clark`** in the second input field "Last Name" and press the *right arrow* keyboard button to select the Add **"Person Button"**.

127. Press form button **"Add Person**" to send the new person record to the application backend.

128. In the popup dialog press ***Ok*** button to confirmation the success message about the creation of a new Person entity.

Result



After pressing the "*Add Person*" button (1) the entered person form data is sent to the backend via OData service. The success message "Person entity has been successfully created" is displayed in a popup dialog. After popup dialog confirmation, the new record "Linda Clark" occurs as new line inside the Persons List table (3).

previous next

3.5.4 Check OData Write Access in SAP HANA Database

Finally we move to the SAP HANA catalog on backend side and look at the new entry for "Linda Clark" in the Persons database table.

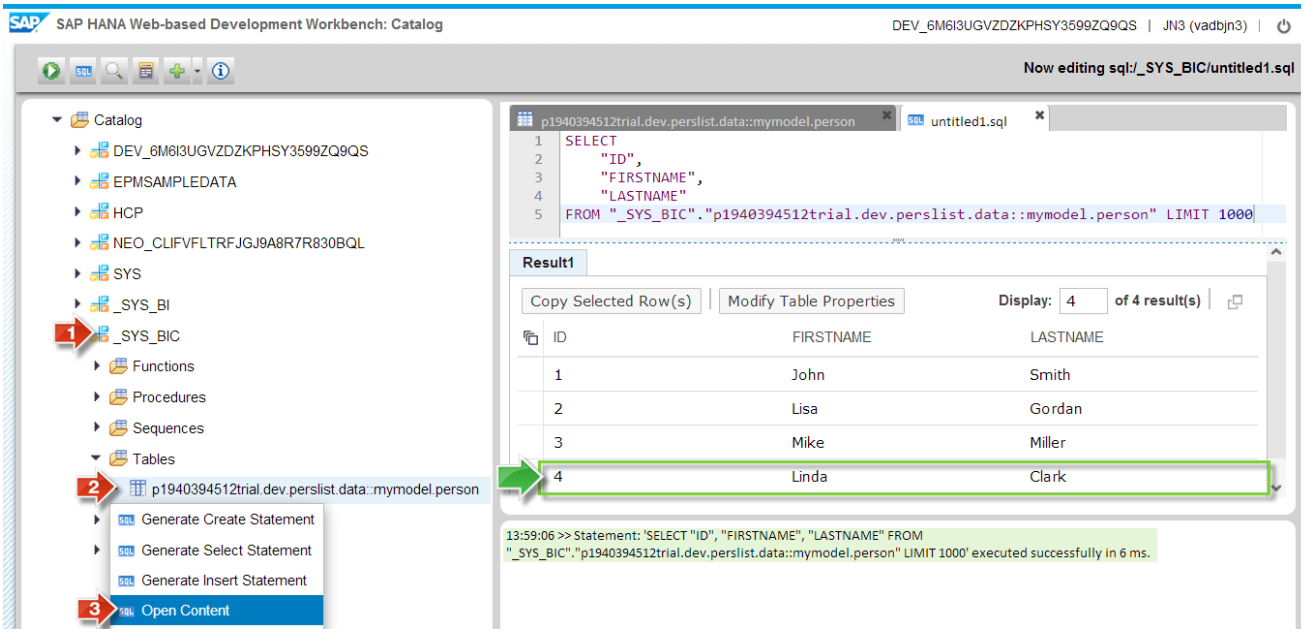
previous next

Hands-on Tasks

129. Open SAP HANA catalog.
130. Select node **Catalog → _SYS_BIC → Tables → p1940xxtrial.dev.perslist.data::mymodel.person**.
131. Select context menu item **Open Content**.

Result

An auto-generated SQL SELECT statement to read all records from table **"_SYS_BIC"."p1940xxtrial.dev.perslist.data::mymodel.person"** gets executed. The new record for person “Linda Clark” gets displayed as last entry in the result table.



previous next

3.5.5 Publish PersonsList application to other SAP HANA Trial Account Users

Finally we make the PersonsList application accessible for all SAP HANA Trial account users by granting the user role defined in section 3.2.6. to the public role.

previous next

Hands-on Tasks

132. From SAP HANA Web IDE editor open the Catalog via the green plus icon in the toolbar.
133. Open a SQL console by clicking the toolbar icon *'Open New SQL Editor'* (blue SQL icon).
134. Copy the SQL code string from Source Code 9 into the opened SQL editor. Replace the `highlighted string` corresponding to the name of your own trial account and user name.

Source Code 19: Call SQL procedure HCP.HCP_GRANT_TO_PUBLIC_ROLE() to grant user role to public role

```
call HCP.HCP_GRANT_TO_PUBLIC_ROLE('p1940xxtrial.dev.perslist.roles::user')
```

135. Execute the SQL command by clicking the toolbar icon **Execute** (green icon with triangle).
136. To run and test the PersonsList application select **index.html** and click **Run on server** from the editor toolbar.

Result

The PersonsList application opens and you can send the URL to anyone else who has a SAP HANA trial account user (or in general a SAP Community Network (SCN) user) to authenticate and run your application.

previous next

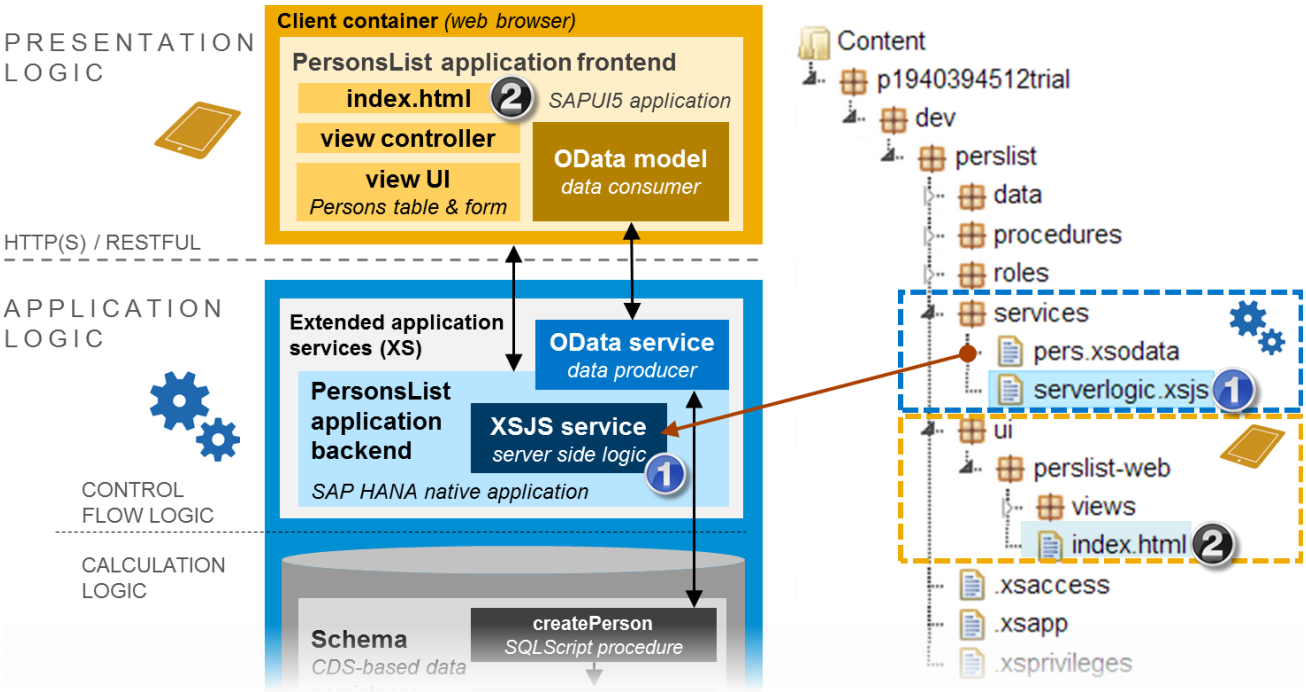
3.6 1. Extension: Writing Server-Side JavaScript Code

Beside the data access via OData, we used in chapter 3.3.1 to access the person table data, SAP HANA XS offers a second consumption model, i.e. ***server-side application programming in JavaScript*** to extract data from SAP HANA.

You will now extend your HANA XS PersonsList application with a small example of such a server-side JavaScript (see below Start Hands-on: 1. Extension) like depicted in Figure 13. The added **serverlogic.xsjs** file contains the function **getUserName()** to retrieve the session user (1). This **serverlogic.xsjs** is called from a script section in the SAPUI5 application's **index.html** to display the result, i.e. the logged on user name, as welcome message in the application header (2).

Preview

Figure 13: 1. Extension – Calling a server side XSJS service from SAPUI5 client by using JavaScript and jQuery



Further information:

For more details on SAP HANA XS server-side application programming in JavaScript read

- SAP HANA Developer Guide: Writing Server-Side JavaScript Code
- SAP HANA Cloud Platform > Debugging with SAP HANA Web-based Development Workbench

Design-Time Application Artifacts Created in this Step

File extension	Object	Description
.xsjs	Server-Side JavaScript Code	A file containing JavaScript code that can run in SAP HANA Extended Application Services and that can be accessed via URL.

Start Hands-on (1. Extension)

Prerequisites

- You have already successfully executed the PersonsList HANAXS application. The PersonsList application is running and works as described in chapter 3.5.

3.6.1 Create a Simple Server-Side JavaScript Service within Descriptor .xsjs

Hands-on Tasks

Now we define a server-side JavaScript service to expose the name of the user which is logged on the PersonsList application. For that you will add a corresponding **.xsjs** service descriptor file to the HANA repository.

137. Under the package **services** add a new file **serverlogic.xsjs**.

138. Add the following code to the **serverlogic.xsjs** file:

Source Code 20: Server-side JavaScript service definition file serverlogic.xsjs to retrieve the session user name

```
function getUsername(){
   var username =  $.session.getUsername();
   return username;
}
var result = getUsername();
$.response.setBody(result);
```

139. Save the new **XSJS service descriptor** to activate it.

3.6.2 Add XSJS Service to Your PersonsList Web Application

Now you will use now the XSJS service that was previously implemented in the PersonsList application to display the name of the logged on user in the application header.

140. In the Editor browser tab (of SAP HANA Web-based Development Workbench) select the file **Content → p1940xxtrial → dev → perslist → ui → perslist-web → index.html** to open the corresponding editor tab.

141. Add the **highlighted code** to the **index.html**.

Source Code 21: index.html with the included xsjs service to display the logged on user in the application header

```
<!DOCTYPE HTML>
<html>
<head>
<meta http-equiv="X-UA-Compatible" content="IE=edge">
<title>SAP HANA Cloud Platform E2E dev scenario: SAP HANA native
  PersonsList application</title>
<script src="/sap/ui5/1/resources/sap-ui-core.js" id="sap-ui-bootstrap"
  data-sap-ui-libs="sap.ui.commons, sap.ui.table,sap.ui.ux3"
  data-sap-ui-theme="sap_bluecrystal">
</script>

<script>
  sap.ui.localResources("views");

  var oAppHeader = new sap.ui.commons.ApplicationHeader("appHeader");
  oAppHeader.setLogoSrc("http://www.sap.com/global/images/SAPLogo.gif");
  oAppHeader.setLogoText("SAP HANA Cloud Platform e2e scenario:"
     + "PersonsList application in SAP HANA XS");
  oAppHeader.setDisplayWelcome(false);
  oAppHeader.setDisplayLogoff(false);
  oAppHeader.placeAt("header");

  var view = sap.ui.view({ id : "perslist-id", viewName : "views.perslist",
    type : sap.ui.core.mvc.ViewType.JS });
  view.placeAt("content");
</script>
```

```
<script>
    jQuery( document ).ready(function() {
            jQuery.get("../../services/serverlogic.xsjs",function(result){
            var oAppHeader = sap.ui.getCore().byId("appHeader");
            oAppHeader.setDisplayWelcome(true);
            oAppHeader.setUserName(result);
        });
        });
</script>

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

NOTE: Know the code – How to call a server side XSJS service from a SAPUI5 application frontend?

SAPUI5 heavily uses jQuery, and the default flavor of SAPUI5 (the sap-ui-core.js bootstrap script) even *includes* jQuery**.** We can therefore directly call the jQuery-API in our SAPUI5 **index.html** file to get the logged in user name as a result of our server side XSJS service:

- `jQuery( document ).ready(function() { … })`: run code as soon as the document is ready to be manipulated. Read more details here …
- `jQuery.get( xsjsServiceUrl, function(result){ … })`: call server side XSJS service via relative URL `"../../services/serverlogic.xsjs"` and pass the result to a client side callback function (in our case an anonymous function). Read more details here …
- `oAppHeader.setUserName(result);`: in the anonymous callback function we set the 'user name' property of the *ApplicationHeader* UI5 control to the result value that was returned from the XSJS service.

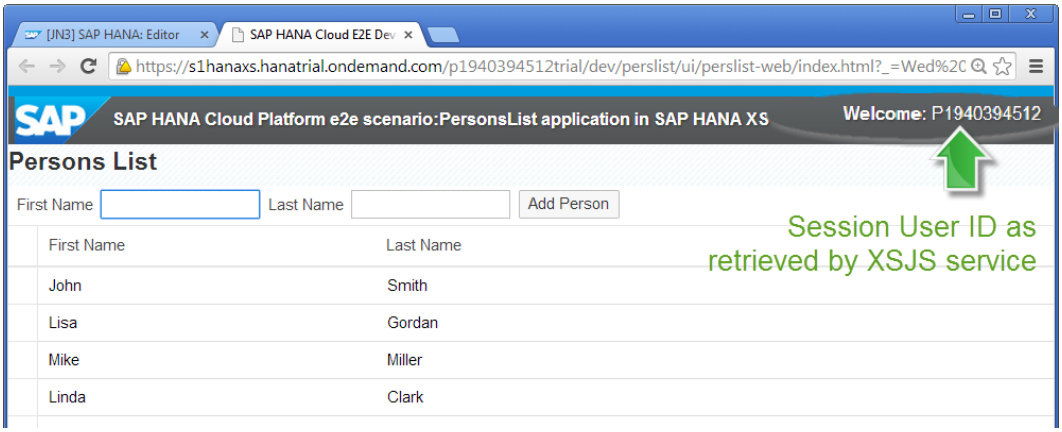
The `$` sign is an alias for the `jQuery` object that is commonly used for jQuery source code like `$( document ).ready()` or `$get()`.

142. Save **index.html** to activate it.

143. Run **index.html** as described in section *3.1.2 Run Blank SAP HANA XS Application in Web Browser.*

Result

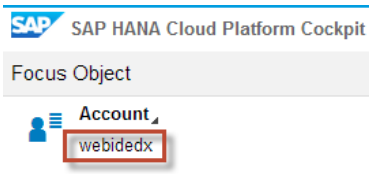
The *PersonsList* application frontend is launched in a new browser tab. The user ID of the logged on user is displayed in the application header.



End Hands-on (1. Extension)

4 Glossary

NOTE: To ease reading we use short terms according to the following table.

Term	Short term	Description
Account		A hosted environment provided to a customer organization, representing a named collection of configurations, authorizations, platform resources and applications.
Account member	member	Indicates a user's assignment to an account. As an account member, a user automatically has the permissions required to use the SAP HANA Cloud Platform functionality within the scope of the respective account. A user can be assigned to more than one account.
Account user		User (with name and password) for login to a SAP HANA Cloud Platform account (via SAP HANA Cloud Platform cockpit) Example: `p1940394512` Placeholder in tutorial: `p1940xx`
Customer account	productive account	Allows customers to build applications and host them in a production environment for their own purposes. A customer account can be purchased as part of a predefined or tailored package. Example: `webidedx` (customer account name obtained from SAP) 
Database user	DB user	User (with name and password) for login to a SAP HANA database. Needed for login to SAP HANA Web-based Development Workbench. *NOTE: Gets auto-created for an account user on SAP HANA Cloud Platform trial landscape with single SAP HANA database. Not needed in this tutorial as it is wrapped by the account user for sake of simplicity.* Example on trial SAP HANA instance: `DEV_1A2B`
Developer account	trial account	Offers access to the SAP HANA Cloud Platform trial landscape for evaluation purposes. A developer account on https://account.hanatrial.ondemand.com/ is free of charge and valid for an unlimited period. It allows restricted use of the platform resources. Example: `p1940394512trial`
Landscape host	host	SAP HANA Cloud Platform host. Europe (customer account): `hana.ondemand.com`, US (customer account): `us1.hana.ondemand.com`, developer account `hanatrial.ondemand.com`
Partner account	productive account	Allows partners to build applications and sell them to their customers. A partner account is available through a partner program, which provides a package of predefined resources and the opportunity to certify, advertise, and ultimately sell products.
Productive SAP HANA instance *see also Trial SAP HANA Instance*	Productive HANA instance	SAP HANA database instance on SAP HANA Cloud Platform that is designed for developing with SAP HANA in a production environment. It is reserved for your exclusive use, allowing you to work with SAP HANA as with an on-premise system.

Term	Short term	Description
SAP Community Network	SCN	SAP's professional social network for SAP customers, partners, employees and experts, which offers insight and content about SAP solutions and services in a collaborative environment: http://scn.sap.com . To use SAP HANA Cloud Platform, you have to be registered on SCN.
SAP HANA catalog	HANA catalog	SAP HANA native applications persist their content in the HANA repository and, depending on the content type, compile artifacts into the runtime catalog.
SAP HANA Cloud Platform	HANA Cloud Platform	SAP HANA Cloud Platform is the Platform-as-a-Service (PaaS) offering from SAP, which enables SAP partners and customers to deploy and use Java applications in a cloud environment.
SAP HANA Cloud Platform cockpit	cockpit	A central point of entry to key information about your accounts and applications, and for managing all activities associated with your account.
SAP HANA Cloud Platform identity service	SAP ID service	The service is responsible for identity management and authorization on SAP HANA Cloud Platform, which you can use in your on-demand applications to ensure proper identity management mechanism.
SAP HANA extended application services	HANA XS	A small-footprint application server, web server, and basis (with configurable OData support, server-side JS execution and full access to SQL and SQLScript) for application development integrated into SAP HANA.
SAP HANA native application	HANA native application	Applications that use SAP HANA extended application services integrated into SAP HANA.
SAP HANA repository	HANA repository	Integral part of the SAP HANA system and development infrastructure that is used for central storage, versioning and lifecycle management of software artifacts.
SAP HANA Web-based Development Workbench	HANA Web IDE	Tools that enable access to and development of repository and catalog objects from a remote location, for example, using a Web browser.
Trial SAP HANA instance *see also Productive SAP HANA Instance*	trial HANA instance	SAP HANA Cloud Platform provides SAP HANA instances that allow you to develop with SAP HANA in a trial environment. A trial SAP HANA instance provides you with a shared database instance, allowing you to work with SAP HANA in a managed environment. Due to restrictions in place to ensure user and data isolation, developers have limited access rights
UI development toolkit for HTML5	SAPUI5	SAP's enterprise-ready HTML5 rendering library for client-side UI rendering and programming

Read more:

- SAP HANA Cloud Platform Documentation > Glossary
- SAP HANA Cloud Platform Documentation > SAP HANA Development

5 Related Content

NOTE: Resources marked with symbol ► are directly related with topics that are covered in this tutorial: *SAP HANA native application development on SAP HANA Cloud Platform using the SAP HANA Web-based Development Workbench*.

5.1 SAP HANA Cloud Platform

- SAP Hana Cloud Platform – The In-Memory Platform-as-a-Service: product landing page
- SAP HANA Cloud Platform documentation: help landing page
- ► Introduction to SAP HANA Cloud Platform and Next Steps in SAP HANA Cloud Platform, openSAP: open online courses delivered by SAP (Rui Nogueira), free of charge. See all course videos here …
- ► SCN blog & interactive online tutorial Try out Web-based HANA XS Development on SAP HANA Cloud Platform by Jens Glander, SAP AG. **NOTE: The application described in the online tutorial is the same that is described in this PDF tutorial.**
- ► SAP Online Help on SAP HANA Cloud Platform > Using a Trial SAP HANA Instance
- ► SAP Online Help on SAP HANA Cloud Platform > SAP HANA Development
- ► SAP Online Help on SAP HANA Cloud Platform > Developing with SAP HANA Web-based Tools
- SAP Online Help on SAP HANA Cloud Platform > Debugging with SAP HANA Web-based Development Workbench
- SAP HANA Cloud Developer Center: SAP Community Network site where you can find information, news, discussions, blogs, and more about SAP HANA Cloud Platform.
- SCN Cross-Technology space: find tutorials on how to implement cross-technology scenarios using a combination of SAP technology products for mobile, cloud and data.
- SCN blog on "Videos of openSAP course "Introduction to SAP HANA Cloud Platform"" by Rui Nogueira, SAP AG, Jan 2014
- SCN document on "SAP HANA Cloud Platform - Content Overview" by Matthias Steiner, SAP AG: content overview about various SCN blog posts/documents about SAP HANA Cloud Platform

5.2 SAP HANA Extended Application Services

- SAP HANA Developer Center in SAP Community Network
- SAP HANA web site: Find out about the latest news updates, upcoming events, exclusive VIP access, and learn more about SAP HANA through the expertise of SAP employees, customers, and partners.
- SAP HANA Academy: engage with SAP HANA through hours of free videos and projects
- SAP HANA Platform help documentation
- openSAP online course (free): access full course material on "*Introduction to Software Development on SAP HANA*"
- ► SAP Help: SAP HANA Developer Guide > Developing Applications in Web-based Environments
- ► http://www.saphana.com/docs/DOC-4372: in this video, Thomas Jung, SAP AG, illustrates the Web-based Development Workbench released with SAP HANA SPS7, 12:33 min, Dec 2013
- ► SCN blog "8 Easy Steps to Develop an XS application on the SAP HANA Cloud Platform" by Stoyan Manchev, Oct 2013
- ► SCN blog "HANA XS development with the SPS07 Web IDE (focus on debugging)" by Kai-Christoph Mueller, Nov 2013
- ► SCN tutorial PDF, **on-premise version of the tutorial you read now**: https://scn.sap.com/docs/DOC-53591 Develop Your First SAP HANA Native Application on SAP HANA Platform Using the SAP HANA Web-based Development Workbench, Bertram Ganz, Jens Glander, SAP AG, March 2014

5.3 UI Development Toolkit for HTML5 (aka SAPUI5)

- UI Development Toolkit for HTML5 Developer Center, SCN space for SAPUI5
- SAPUI5 SDK Demo Kit with developer guide documentation, control/API reference and Test Suite
- Get to Know the UI Development Toolkit for HTML5 (aka SAPUI5), SCN doc, Bertram Ganz, SAP AG, October 2012
- Building SAP Fiori-like UIs with SAPUI5 in 10 Exercises: SCN tutorial on how to build SAP-Fiori like UIs with SAPUI5 in 10 exercises, SAP AG, January 2014