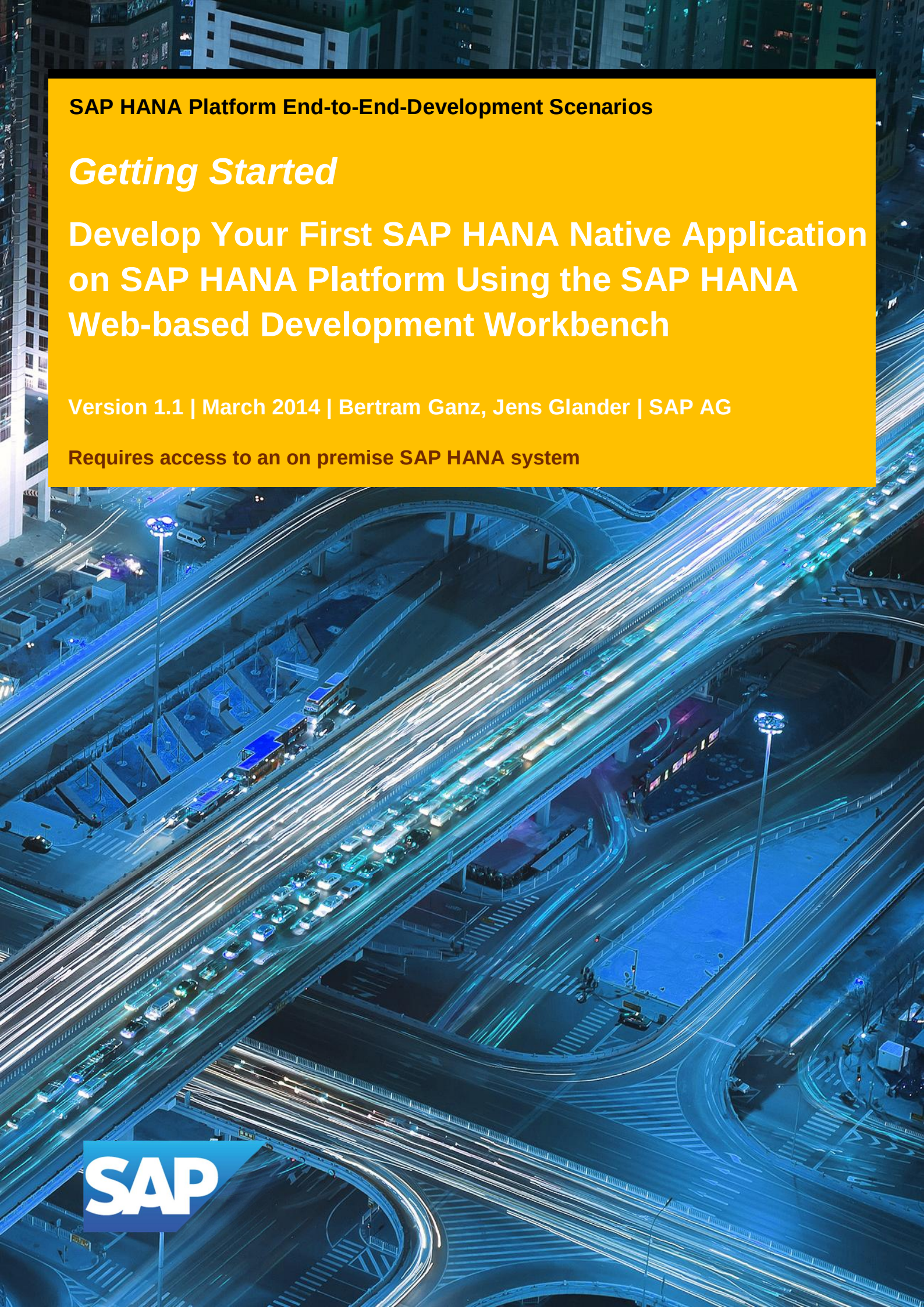


SAP HANA Platform End-to-End-Development Scenarios

Getting Started

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

Version 1.1 | March 2014 | Bertram Ganz, Jens Glander | SAP AG

Requires access to an on premise SAP HANA system

Table of Contents

Source Code

Figures

Screenshots

1 Introduction

1.1 What Do You Get Here?

This guide describes step by step how to create your first SAP HANA native application on an on premise SAP HANA Platform by using the zero-install SAP HANA Web-based Development Workbench. 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 on the SAP HANA Platform SPS07
- **Database**: SAP HANA 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 us to develop the whole application in a browser without the need to install any development tools on your client.
- **Frontend**: You will use the UI development toolkit for HTML5 (shortly *SAPUI5*), SAP's HTML5 library that helps you to build responsive business applications for all devices.

1.2 Intended Audience

All developers and programmers who may be new to the SAP HANA Platform and/or SAP HANA native development based on *SAP HANA extended application services* (also known as *SAP HANA XS*).

1.3 Applicable Releases

The tutorial was built on SAP HANA SPS 07 Database Revision 73 (version string 1.00.73.00.389160, see SAP note 1990011).

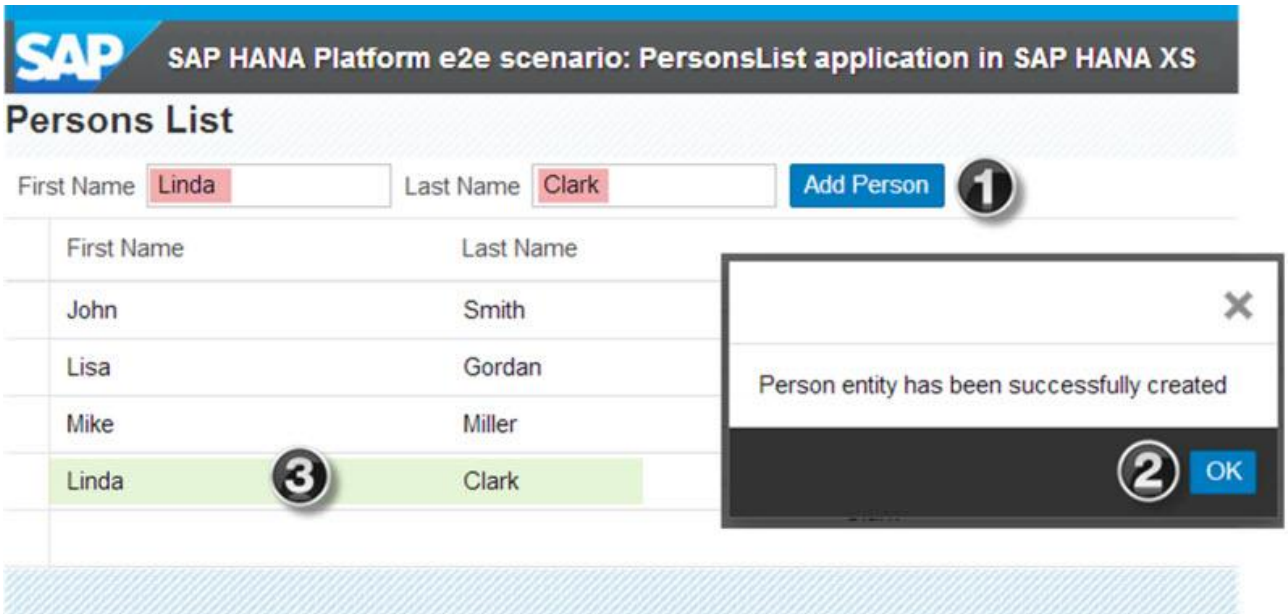
1.4 End-2-End Development Scenario Overview

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) Ask your SAP HANA administrator to assign HANA Web IDE specific roles to your database user

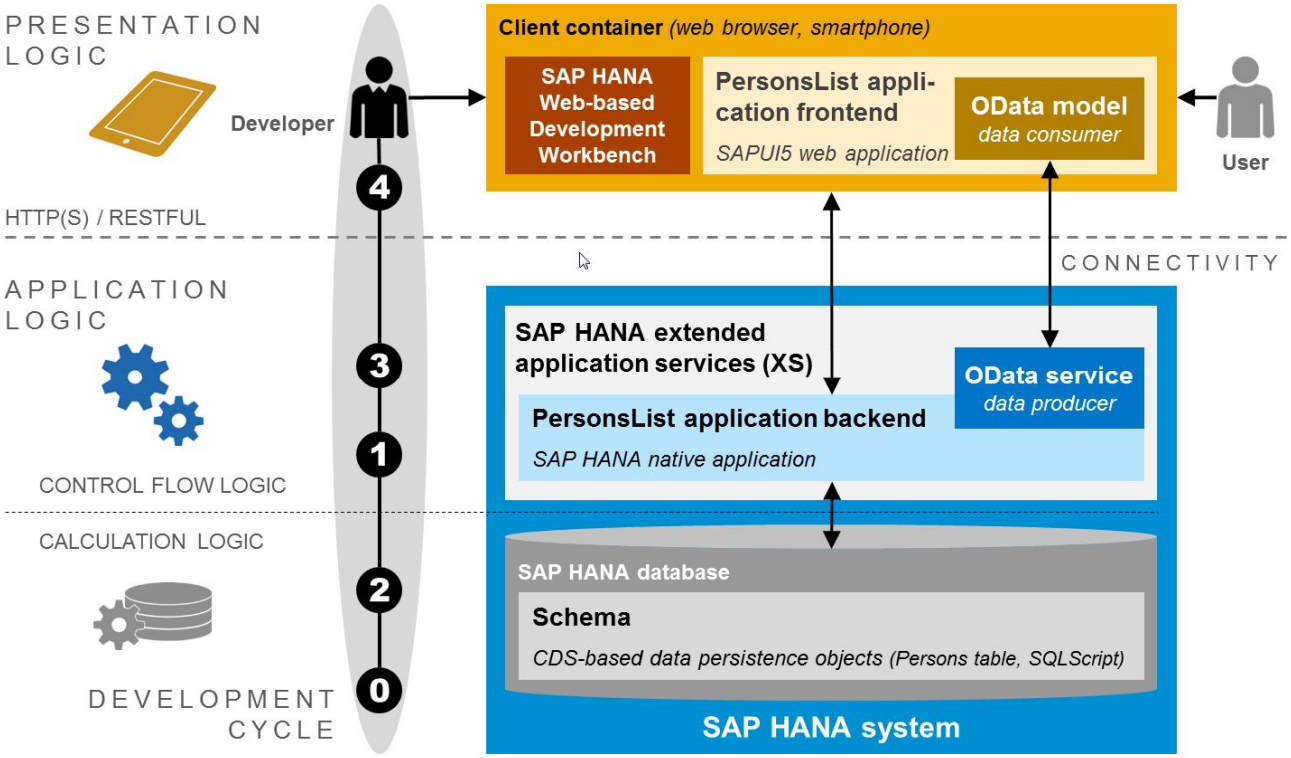
(1) Create a bare-bone SAP HANA XS application

(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 on premise 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: database user with developer roles**	• Know the developer roles that are required for a database user to use the SAP HANA Web-based Development Workbench.
Step 1	**Create new SAP HANA native application within SAP HANA Web-based Development Workbench**	• Start the SAP HANA Web-based Development Workbench in a Web browser • 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* • Create a database schema • Define data type *Person* within CDS document • Load initial mock data from .csv file into database table
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 read the following list of hints:

Hint 1: Which Web Browser Should you use?

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

Hint 2: First set-up Your Database User

Before you can start with the web-based HANA XS application development **you need to set-up your SAP HANA database user** as described in chapter 2: *Prerequisites of Web-based Development: Developer Roles*

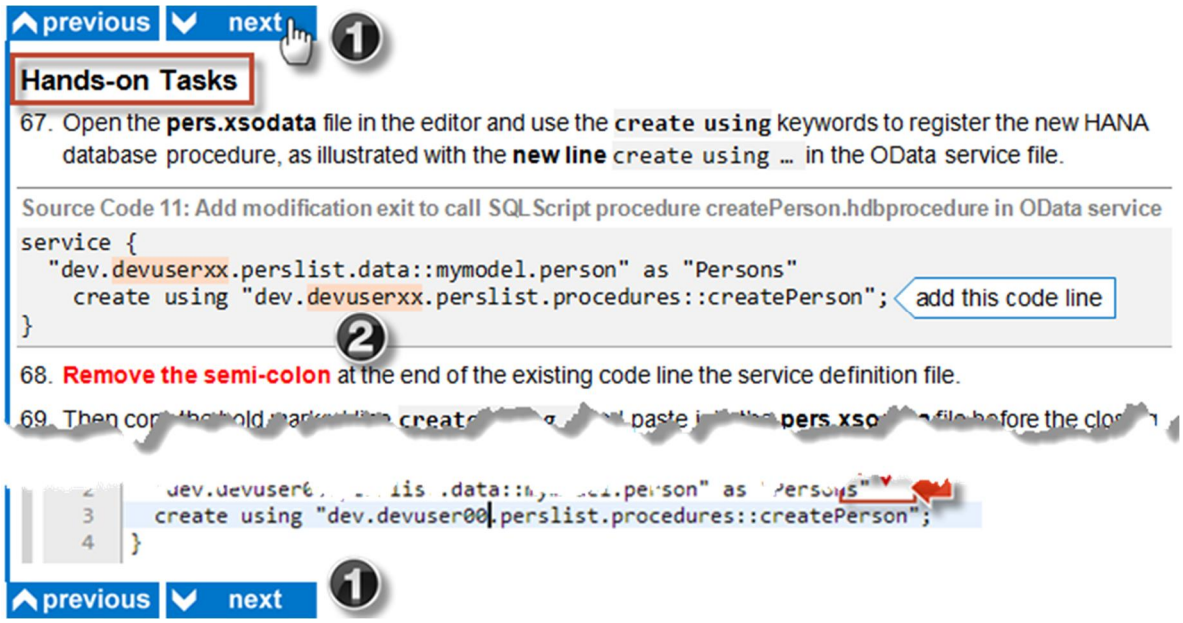
Hint 3: For Quick Success just Follow the Hands-On Tasks

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 sections.

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



NOTE: For a deeper understanding of SAP HANA XS application development on SAP HANA Platform we warmly 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-Specific Names

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** `devuserxx` with your own user-specific names (e.g. `devuser00` or `gordanl`, see Screenshot 1, bullet item 2).

2 Prerequisites of Web-based Development: Developer Roles

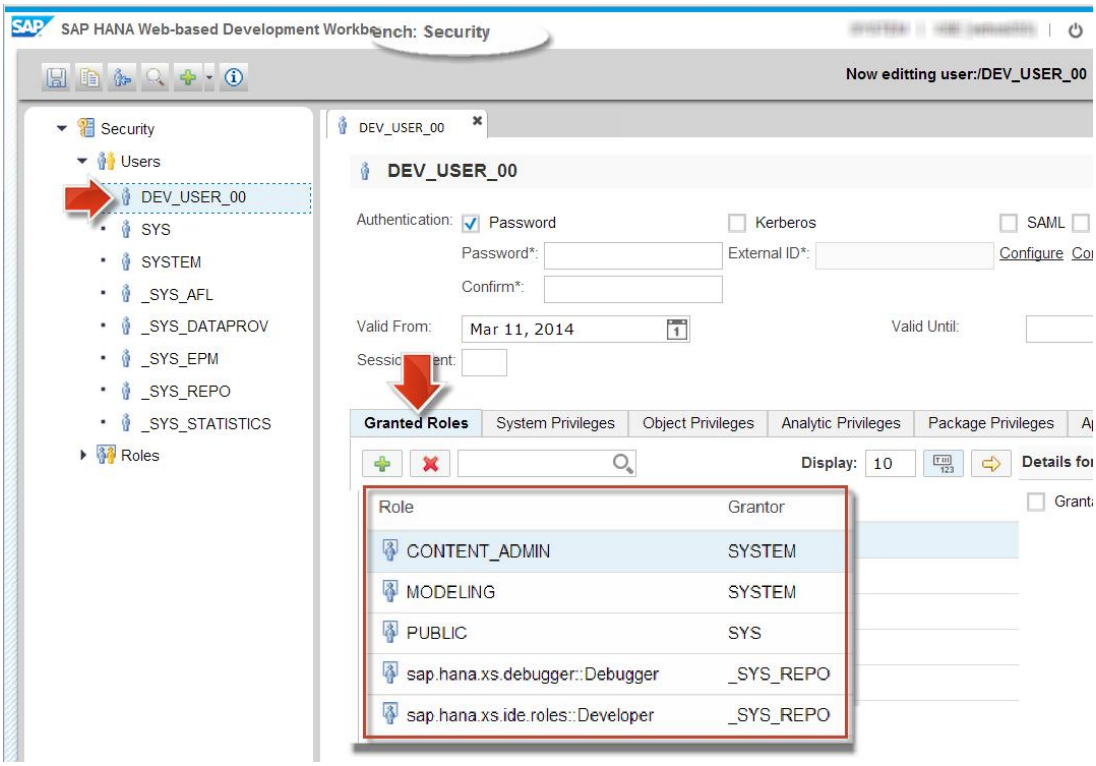
The SAP HANA Web-based Development Workbench provides application developers with tools that enable access to SAP HANA repository and catalog objects from a remote location using a Web browser without the need to install the SAP HANA studio locally.

2.1 Get a Database User with the Required Roles for the HANA Web IDE

Before you start using the SAP HANA Web-based Development Workbench, your **SAP HANA administrator** must set up a **user account** for you in the SAP HANA database and assign the following required developer roles to your SAP HANA database user:

Role	Description
`sap.hana.xs.ide.roles::Developer`	This role grants the privileges required to use all four tools included in the SAP HANA Web-based Development Workbench (editor, catalog, security and traces). The role `sap.hana.xs.ide.roles::Developer` contains the following four roles that grant access to the user interface of the HANA Web IDE. To see HANA content and to apply the associated tool functions a user requires additional privileges. • `sap.hana.xs.ide.roles::EditorDeveloper` to access to the HANA Web IDE editor (with proper privileges: inspect, create, change, delete and activate SAP HANA repository objects). • `sap.hana.xs.ide.roles::CatalogDeveloper` to access the HANA Web IDE catalog (with proper privileges: create, edit, execute and manage SQL catalog artifacts in SAP HANA database) • `sap.hana.xs.ide.roles::SecurityAdmin` to access the security section (with proper privileges: to create users, create roles, assign objects and manage security add) • `sap.hana.xs.ide.roles::TraceViewer` to access to the trace section (with proper privileges: view and download traces for SAP HANA XS applications and set trace levels e.g. info, error, debug).
`sap.hana.xs.debugger::Debugger`	The debugger role enables the developer to debug XS server-side JavaScript source code. This role is needed to run section 1. Extension: Writing Server-Side JavaScript Code at the end of the tutorial.

Screenshot 2: Database user DEV_USER_00 with granted roles sap.hana.xs.ide.roles::Developer and sap.hana.xs.debugger:Debugger that are needed to access the HANA Web IDE frontend



NOTE: The user `DEV_USER_00` we used in this tutorial to build the PersonsList XS application got also assigned the roles `PUBLIC`, `MODELING` and `CONTENT_ADMIN`.

Further information: SAP HANA Developer Guide > 3.9.3.1 Tutorial: Creating a User and Assigning Roles

2.2 Start SAP HANA Web-based Development Workbench and Change Initial Password

For development of our SAP HANA native PersonsList application you can launch the SAP HANA Web-based Development Workbench and login with your SAP HANA database user.

Start Hands-on

Prerequisites

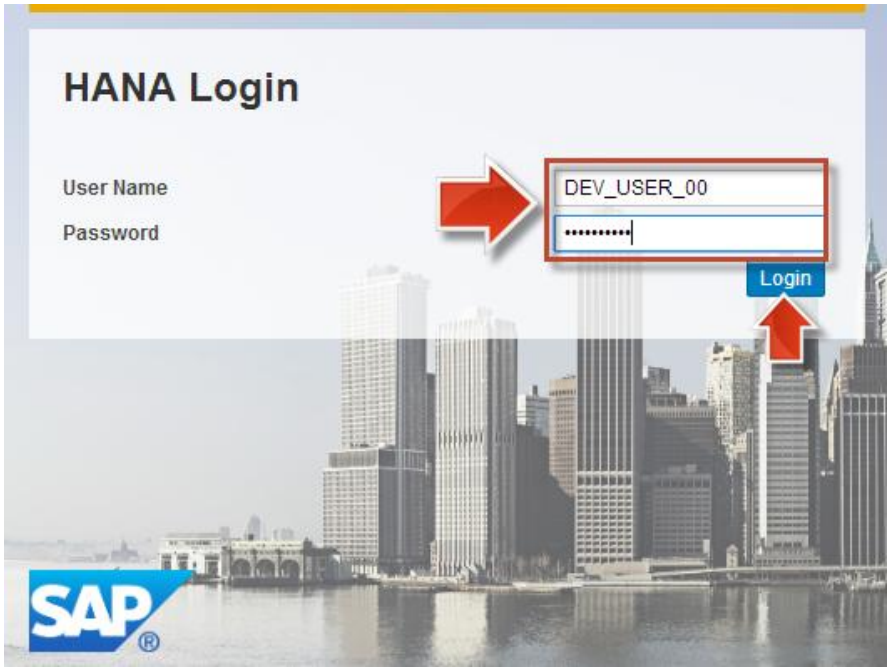
- **You have a database user:** You need the database user to work with SAP HANA Web-based Development Workbench. In the same way you would also need the database user if you would develop with SAP HANA Studio.
- Your database user is assigned the relevant roles for the SAP HANA Web-based Development Workbench (`sap.hana.xs.ide.roles::Developer`).

Hands-on Tasks

1. Open the SAP HANA Web-based Development Workbench Editor in your Web browser. It is available on the SAP HANA XS Web server at the following URL:
   `https://<Web server host>:80<SAP HANA instance>/sap/hana/xs/ide/`

   NOTE: SAP HANA Web-based Development Workbench supports Microsoft Internet Explorer (10+), Mozilla Firefox, and Google Chrome Web browsers.

2. On the HANA Login page enter your database user name and initial password.

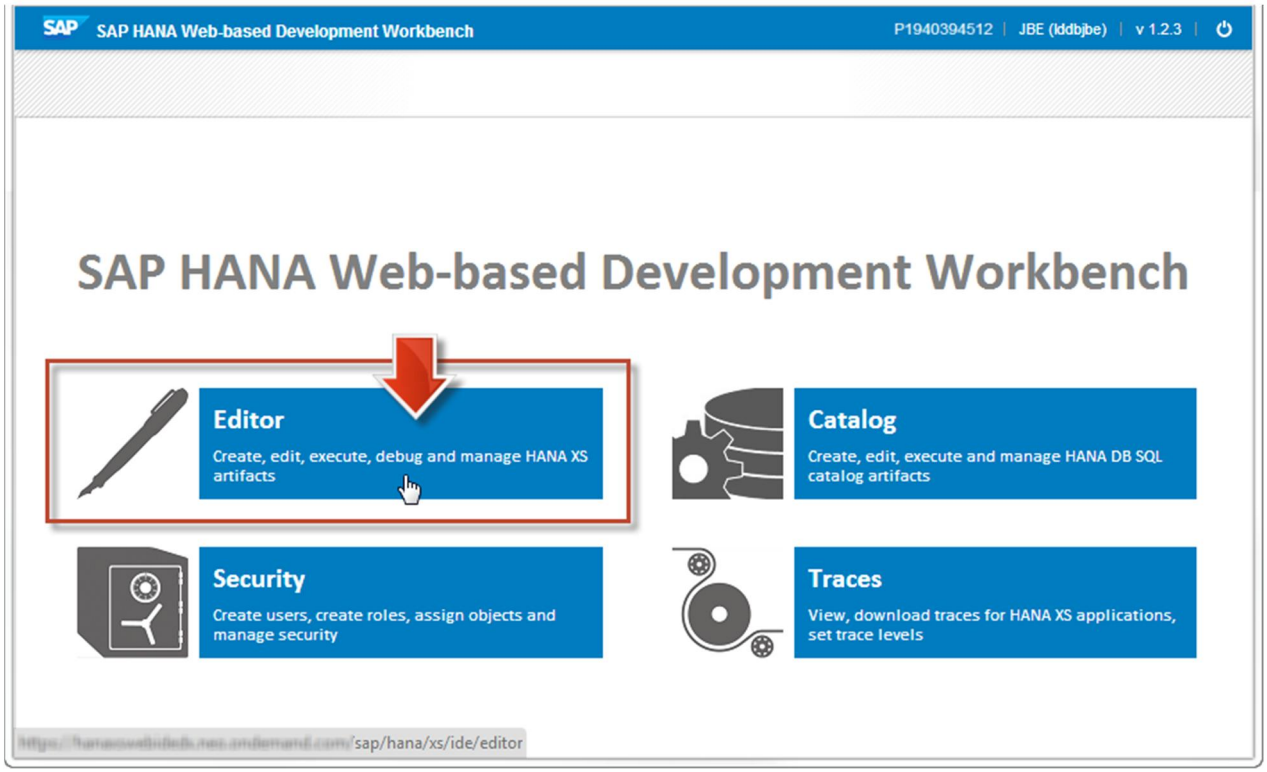


3. When you launch the SAP HANA Web-based Development Workbench for the very first time with your initial password you get a prompt to change your initial password.

   Enter the corresponding password fields and press **Change Password** button.

Result

After the HANA Login dialog you enter the landing page of SAP HANA Web-based Development Workbench.



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
- SCN blog What´s New? SAP HANA SPS 07 Web-based Development Workbench, Tom 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, Tom Jung, SAP AG, Dec 18, 2013

2.2.1 Remember Your Database User Data

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

Data	Comment	Name used in this tutorial	Example
Database user	Database user of the SAP HANA database on an on premise SAP HANA system. Needed to log on to the HANA Web IDE	`DEV_USER_XX` *as the example user used through the entire tutorial*	`DEV_USER_00` `GORDANL`
Database password		`********`	
XS application package name	Name of the SAP HANA repository package were your XS application design time artifacts are stored	In source code: `devuserxx`	`devuser00` `gordanl`
SAP HANA database schema name	Schema name in SAP HANA catalog were your database objects are stored	`devuserxx_schema`	`devuser00_schema` `gordanl_schema`

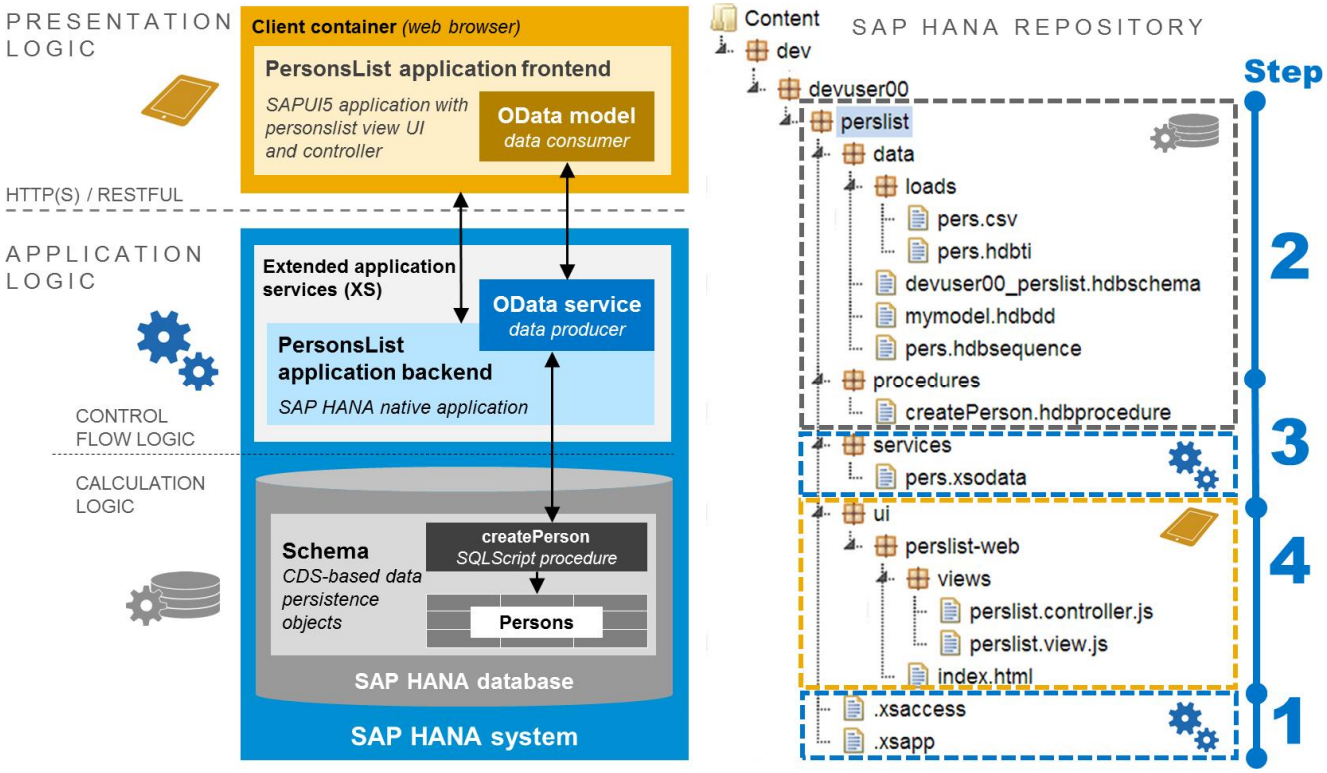
3 Build a Simple SAP HANA Native PersonsList Application

Preview

In Figure 2 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).

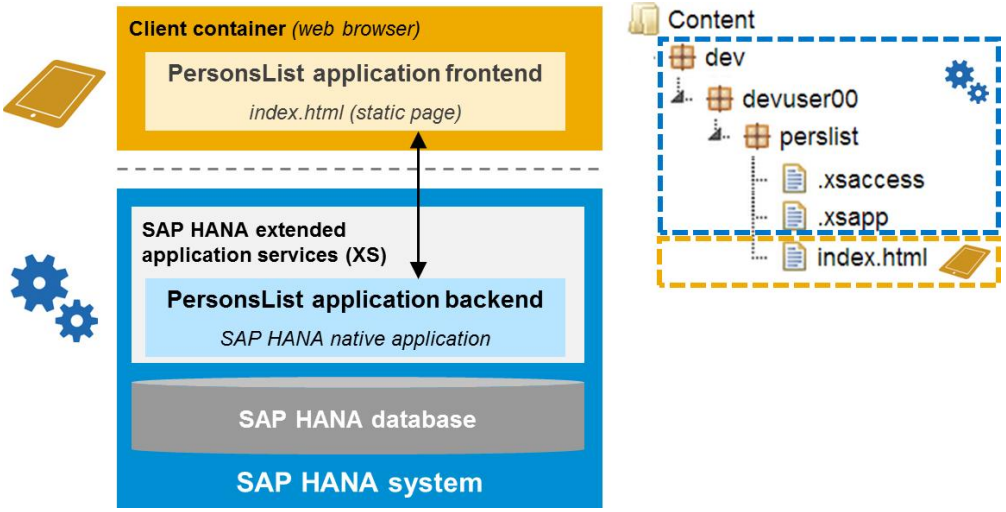
Figure 2: Architecture, artifacts and development steps of final SAP HANA native PersonsList application



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

Preview

Figure 3: 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 an application's availability within SAP HANA extended application services. The package that contains the application descriptor becomes the root path of the resources exposed by the application.
.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.
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 (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) that altogether define the behavior and appearance of the application.

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.

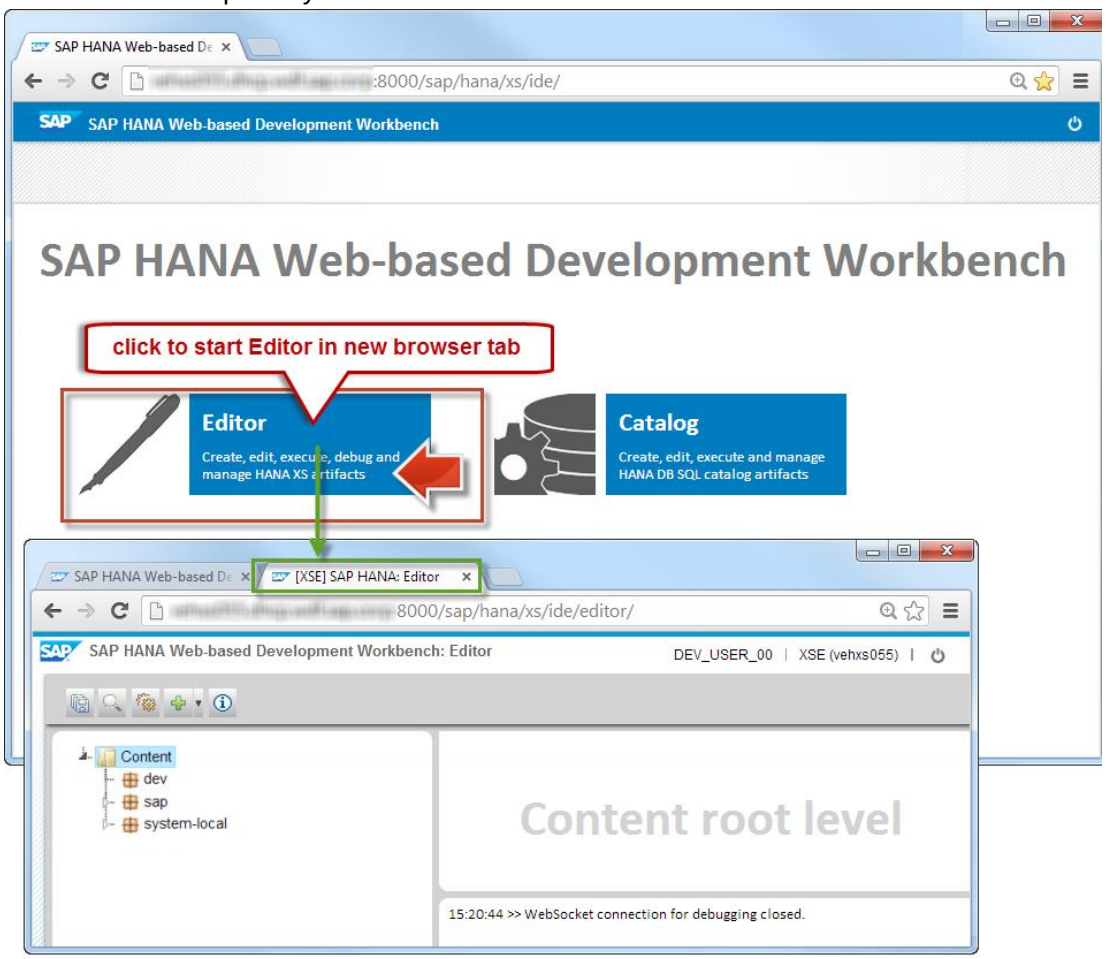
previous next

Prerequisites

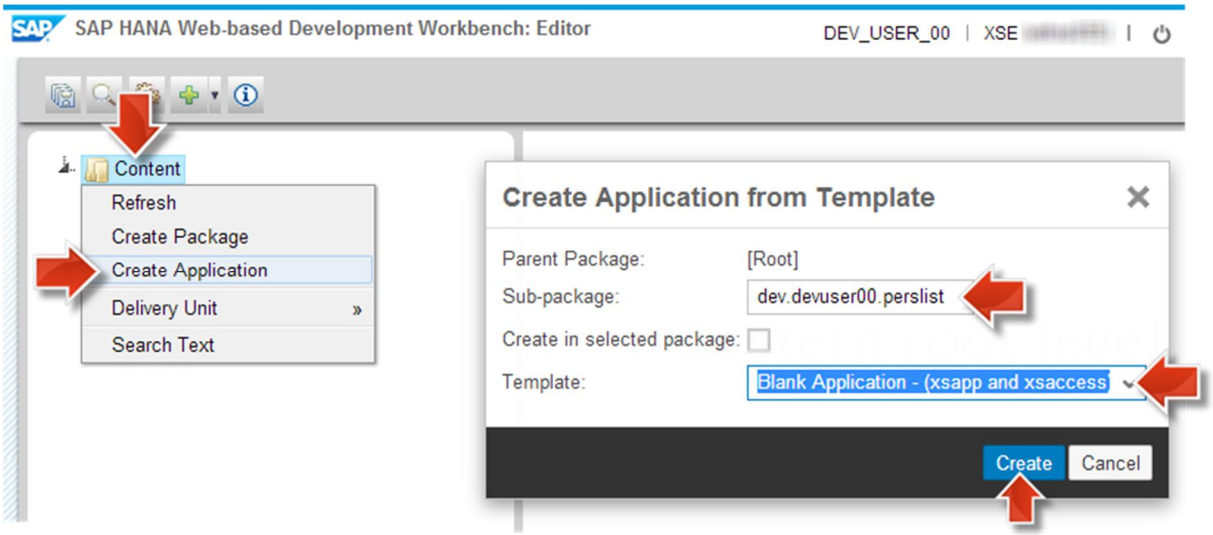
- You have launched the *SAP HANA Web-based Development Workbench*. To do so, you need a database user with the role **sap.hana.xs.ide.roles::Developer** that grants the privileges required to use all the tools included in the SAP HANA Web-based Development Workbench.

Hands-on Tasks

1. On the landing page of the SAP HANA Web-based Development Workbench click on the “Editor” tile to start the HANA Repository Editor in a new browser tab.



2. In the Editor’s content tree select root node **Content** inside the SAP HANA Repository.
3. Open context menu and select item **Create Application**



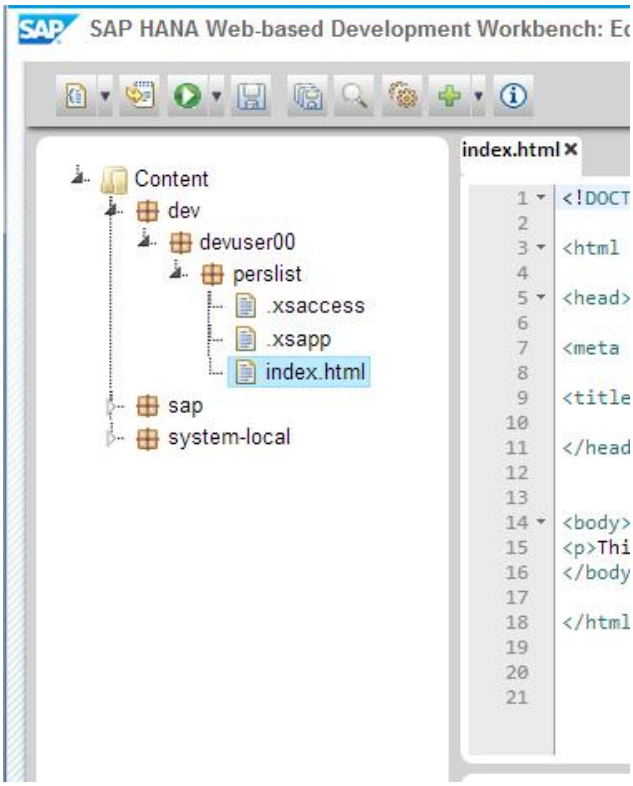
4. In the *“Create Application from Template”* popup dialog enter package name `dev.devuserxx.perslist` e.g. `dev.devuser00.perslist`
5. In *Template* dropdown list select item **Blank Application – (xsapp and xsaccess)**
6. Press **Create.**
7. The system creates the files **index.html**, **.xsaccess** (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 **<user's root package>.perslist** in SAP HANA Repository that comprises the initial application (see below) those that will be incrementally added later.
- **.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 **dev/devuserxx/perslist** with the editor's template creation function we can run it in the web browser.

previous next

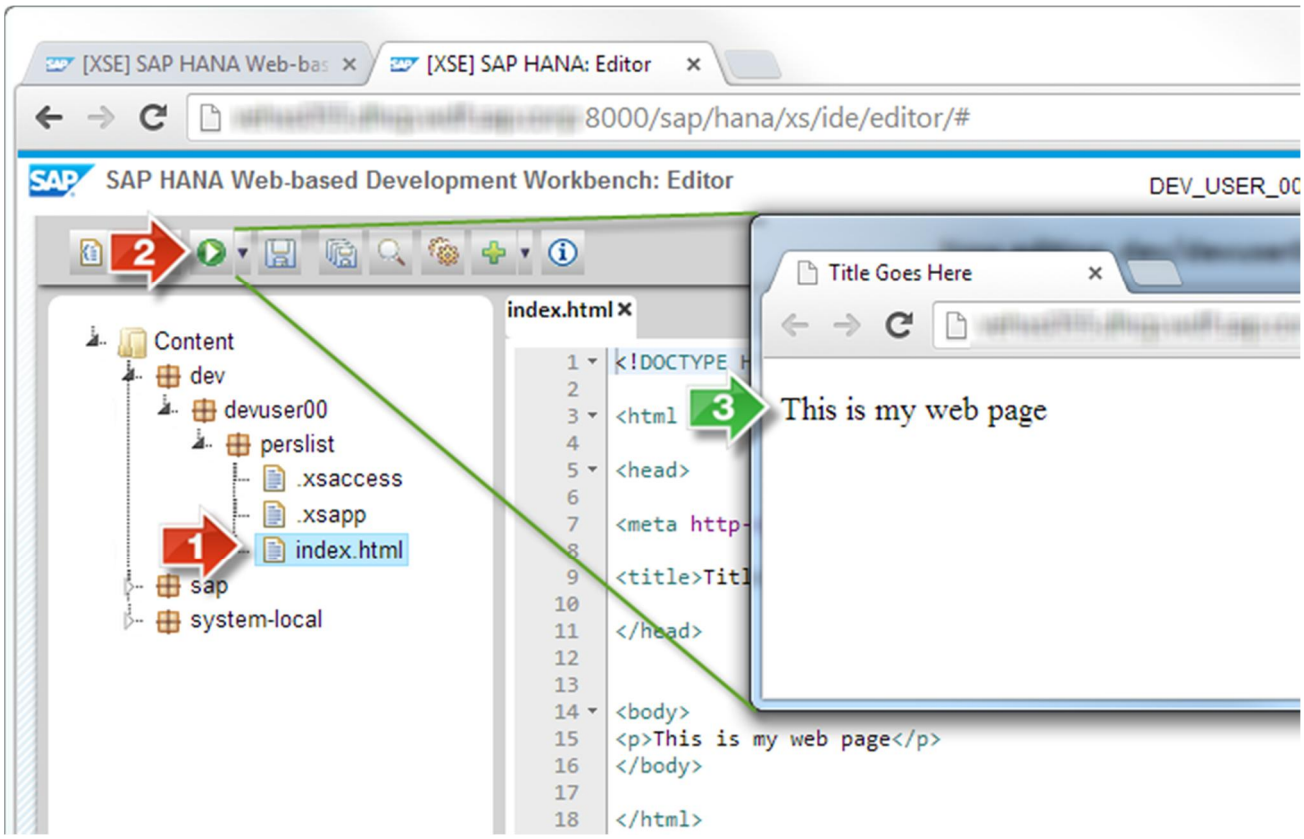
Hands-on Tasks

8. To test the blank application in the web browser, open your application package in the editor's Content tree: **Content → dev → devuserxx → 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.

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

Result



previous next

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. If the package **sap.test** contains the file .**xsapp**, the application will be available under the URL `http://<host>:<port>/sap/test/`.

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.

3.1.2.2 Application Access File .xsaccess

The **application access file** with name **.xsaccess** (suffix only) defines data (content) exposure to clients, defines the authentication method and configures application privileges required to access content of package and its sub-packages, 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.

Source Code 1: .xsaccess file to define application access permissions

```
{"exposed" : true ,"authentication" : [{"method" : "Basic"}]}
```

3.1.2.3 Application Privilege File .xsprivileges *[not covered in this tutorial]*

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).

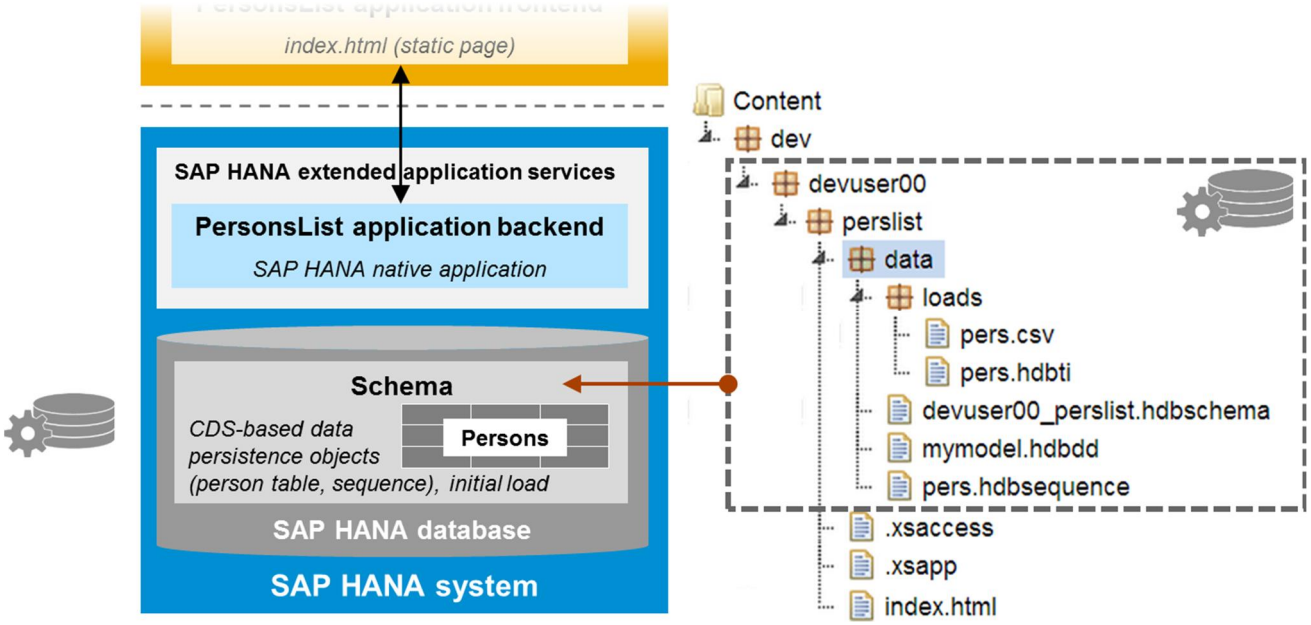
Multiple **.xsprivileges** files are allowed, but only at different levels in the package hierarchy.

Further information: SAP HANA Developer Guide > 3.5.8 Creating an SAP HANA XS Application Privileges File

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

Preview

Figure 4: 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
`.hdbschema`	Schema	A design-time definition of a database schema, which organizes database objects into groups.
`.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).
`.hdbti`	Table import definition	A table-import configuration that specifies which .csv file is imported into which table in the SAP HANA system
`.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.

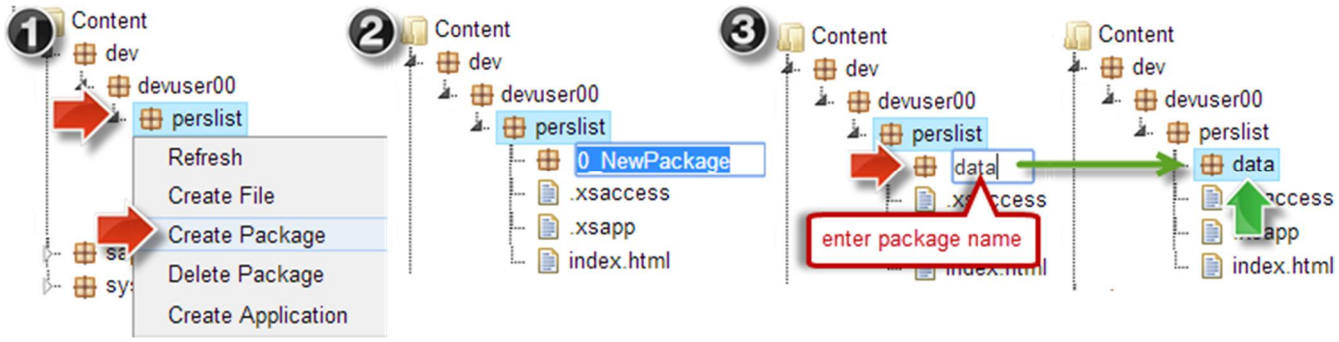
3.2.1 Create Sub-Package *data*

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 `dev/devuserxx/perslist`. Files related with the persistence model are stored in sub-package **data** that can easily be added within three steps:

previous next

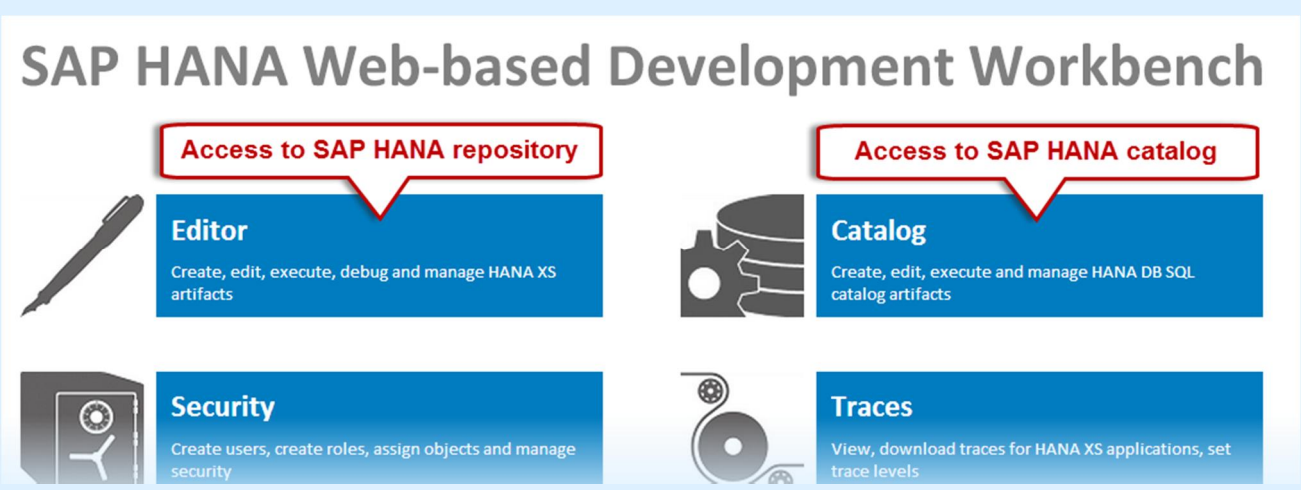
Hands-on Tasks

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



Result

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 Create New Database Schema

Next, we create a new database schema in the SAP HANA catalog by adding a new schema definition file `.hdbschema` to the HANA repository. This database schema enables you to create and activate application artifacts such as tables, views and database procedures. Without a defined schema, 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.

3.2.2.1 Add File devuserxx_perslist.hdbschema to define new Database Schema

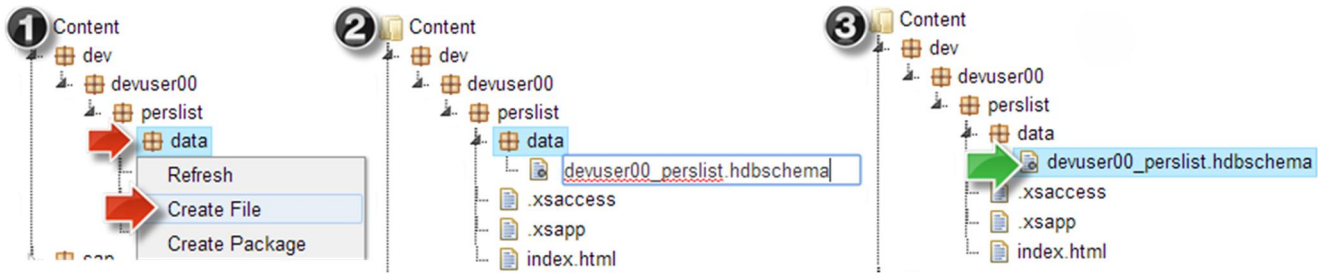
previous next

Hands-on Tasks

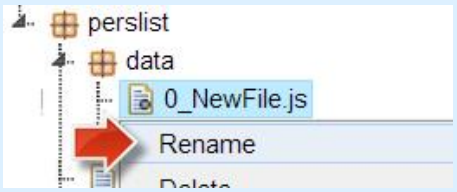
13. In the Editor's Content tree select package node ***Content → dev → devuserxx → perslist → data***

14. Select context menu item ***Create File.*** The newly created file is named ***0_NewFile.js*** by default.

15. Enter `devuserxx_perslist.hdbschema` as name of the new database schema.



NOTE: In case the tree node item for a new file loses focus then apply rename function in context menu



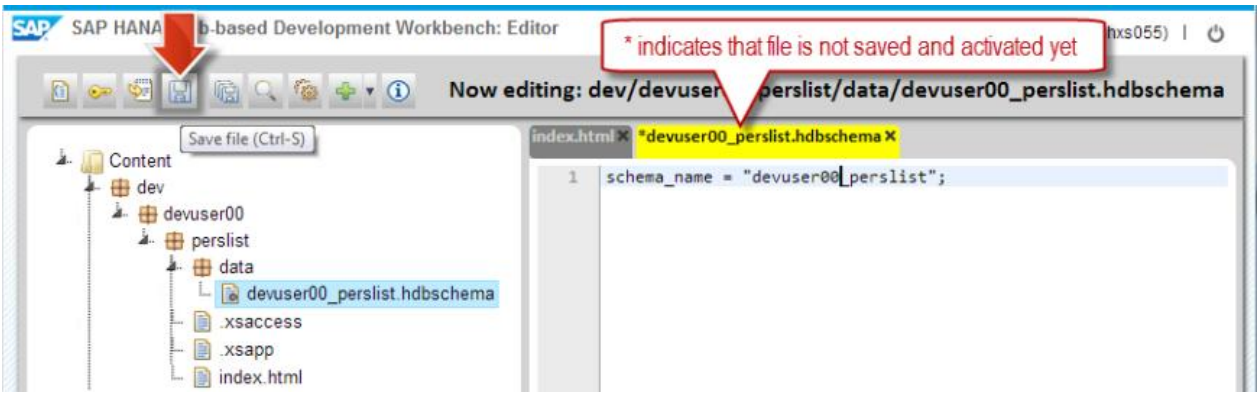
16. In the editor tab for file *devuserxx_perslist.hdbschema* enter attribute-value pair `schema_name = "devuserxx_perslist";` to define the name of your new database schema:

Source Code 2: devuserxx_perslist.hdbschema file

```
schema_name = "devuserxx_perslist";
```

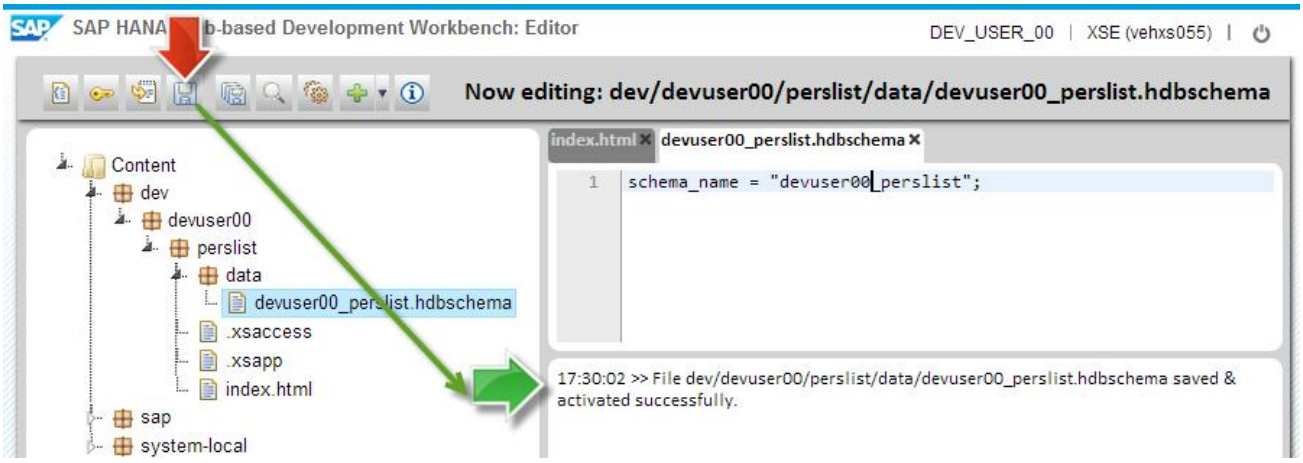
NOTE: You can also press toolbar button  *Insert template* to add an attribute-value pair for the schema name.

17. The yellow marked editor tab with asterisk * before the file name indicates, that your locally edited file is not yet saved and activated in the SAP HANA repository. Either press toolbar button **Save file** or apply keyboard shortcut **Ctrl + S** to save and activate the new **.hdbschema** file in a single step.



Result

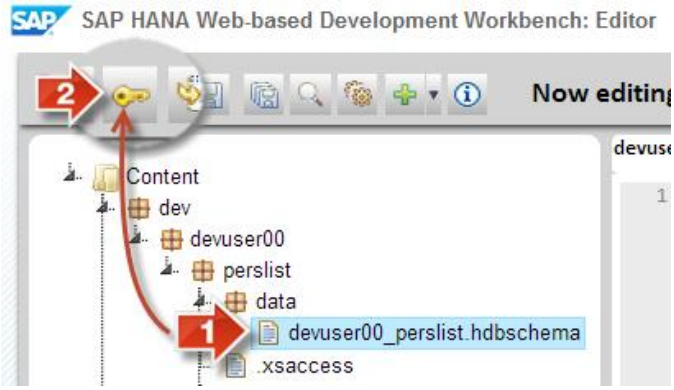
The message area reports the successful save and activation of the new database schema.



previous next

3.2.2.2 Assign Schema Execution Authorization to Your User as Schema Owner

You will see the newly activated schema in the SAP HANA catalog; however you don't actually have access to this schema yet. The activation of the schema was done by the repository and consequently the user SYS_REPO owns the schema. This is true of all database objects created in this new repository approach.



Next you will assign an execution authorization for database schema `devuserxx_perslist` to your own user. Only then your user has authorization to see the objects, which we are creating in this tutorial, inside the SAP HANA catalog.

previous next

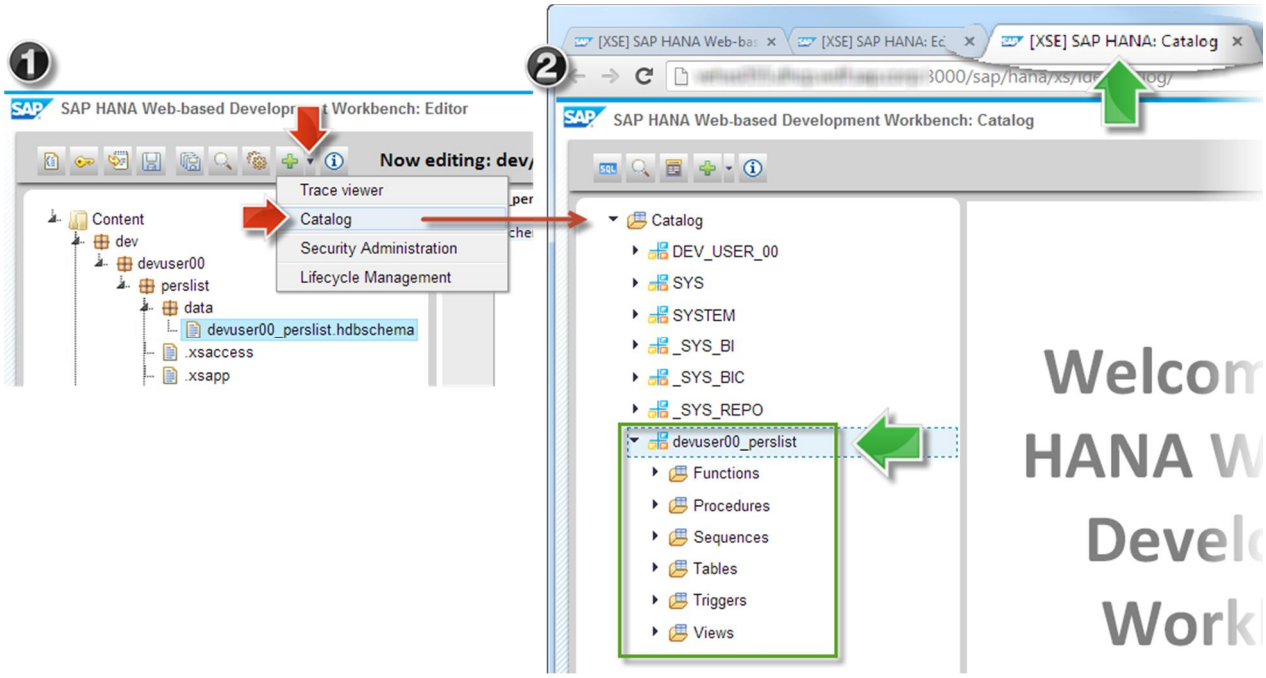
Hands-on Tasks

18. In the catalog tree select node item **Content → dev → devuserxx → perslist → data → devuserxx_perslist**.

19. In the toolbar press button  ***Assign Execution Authorization.*** As a result, the success message *Schema privileges assigned successfully* gets displayed in the editor's message view:

NOTE: Optionally access rights for a catalog schema can be defined by a adding a new repository role via **.hdbrole** file and granting it to our user.

20. To see the new database schema open the SAP HANA catalog in a new browser tab by pressing toolbar button  *More* and selecting menu item ***Catalog*.**



21. In the new browser tab select and expand catalog schema node **devuserxx_perslist**

Result

Your newley created empty database schema `devuserxx_perslist` with empty sub-folders *Functions, Procedures, Sequences, Tables, Triggers, and Views* get displayed in the **Catalog** tree.

previous next

3.2.3 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

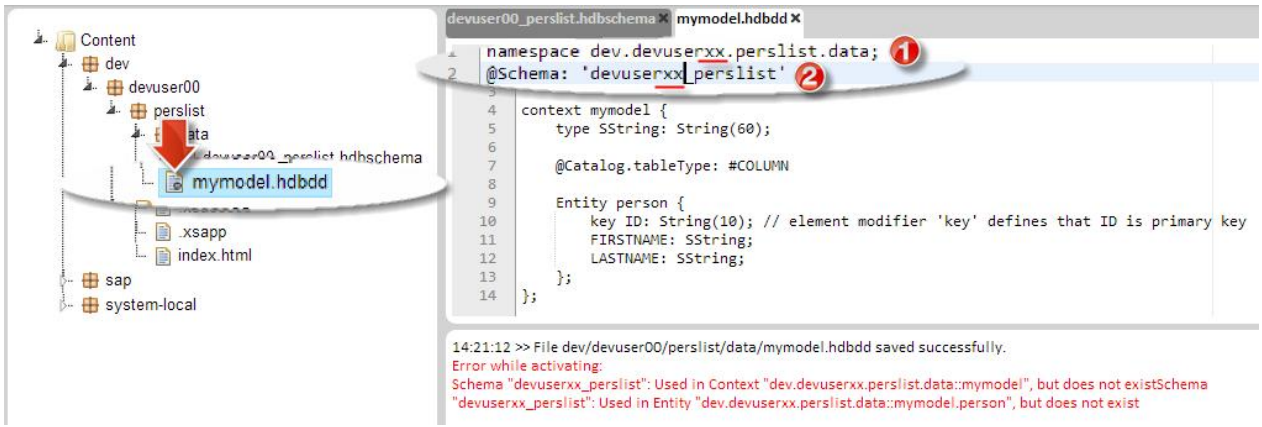
22. Open the **Editor** tab of SAP HANA Web-based Development Workbench.
23. In the Editor's repository content tree select package node **Content → dev → devuserxx → perslist → data**
24. Select context menu item ***Create File*** and enter filename `mymodel.hdbdd`**.** The design-time object definition that you create using the CDS-compliant syntax must have the file extension **.hdbdd**
25. Enter Source Code 3 to define the entity **person**:

Source Code 3: Data definition file mymodel.hdbdd with invalid schema name and namespace

```
namespace dev.devuserxx.perslist.data;
@Schema: 'devuserxx_perslist'

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;
    };
};
```

NOTE: When running this tutorial on SAP HANA SPS 07 Database Revision 73 (version string 1.00.73.00.389160) using Google Chrome browser you encounter a problem: Source Code 3 you copied and pasted from the PDF file to the HANA Web IDE editor tab *mymodel.hdbdd* cannot be activated due to a Chrome-specific formatting issue. The easiest workaround is to edit, save and activate the CDS file in Mozilla Firefox browser where no error occurs.

26. Save the **mymodel.hdbdd** file with toolbar button **Save file** or with keyboard shortcut **Ctrl+S**



Activation fails for two reasons:

- **invalid namespace (1)**: the name space declared in a data definition file must match the repository package in which the hdbdd-file is located. The copied namespace `dev.devuserxx.perslist.data` contains the string `devuserxx` to be replaced with your **own package name,** e.g. `devuser00`.
- **Invalid schema name (2):** the schema name also contains the string `devuserxx` which needs to be replaced by the correct schema name `'devuserxx_perslist'` defined in your **.hdbdd** file.

Due to the failed activation of **mymodel.hdbdd** it is displayed with another icon: (decorated with black circle). Successfully activated repository files are displayed with icon .

27. To resolve this activation error replace the `highlighted strings` corresponding to the name of your own database user.

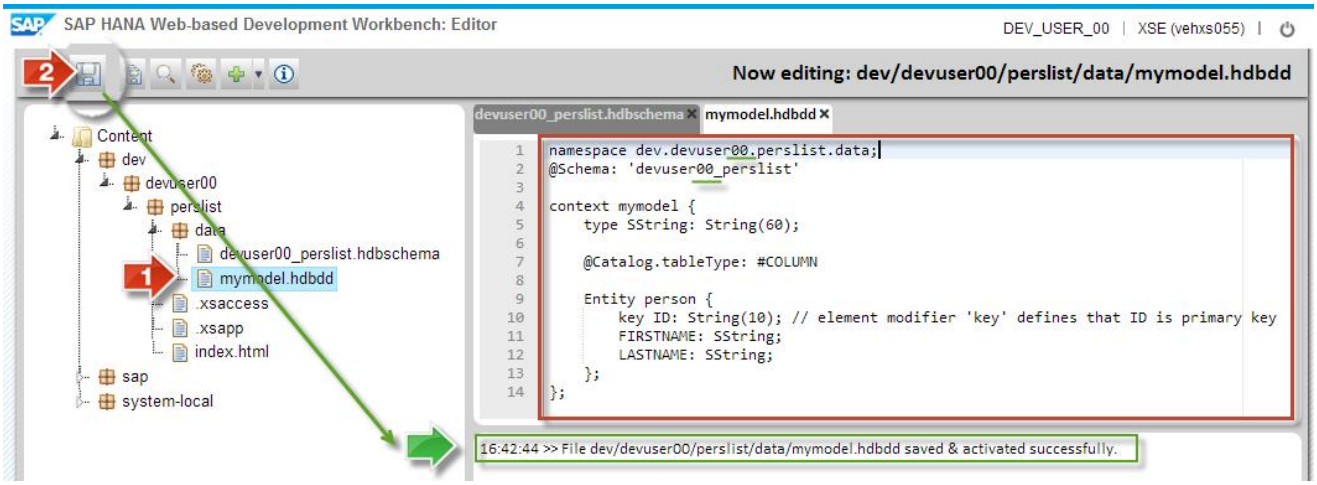
copy Source Code 4 *from next page …*

Source Code 4: Data definition file mymodel.hdbdd with correct namespace and schema

```
namespace dev.devuserxx.perslist.data;
@Schema: 'devuserxx_perslist'

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;
    };
}
```

28. Save the **mymodel.hdbdd** file with toolbar button **Save file** or with keyboard shortcut **Ctrl+S.**

Screenshot 3: Activation of data definition file mymodel.hdbdd in SAP HANA repository



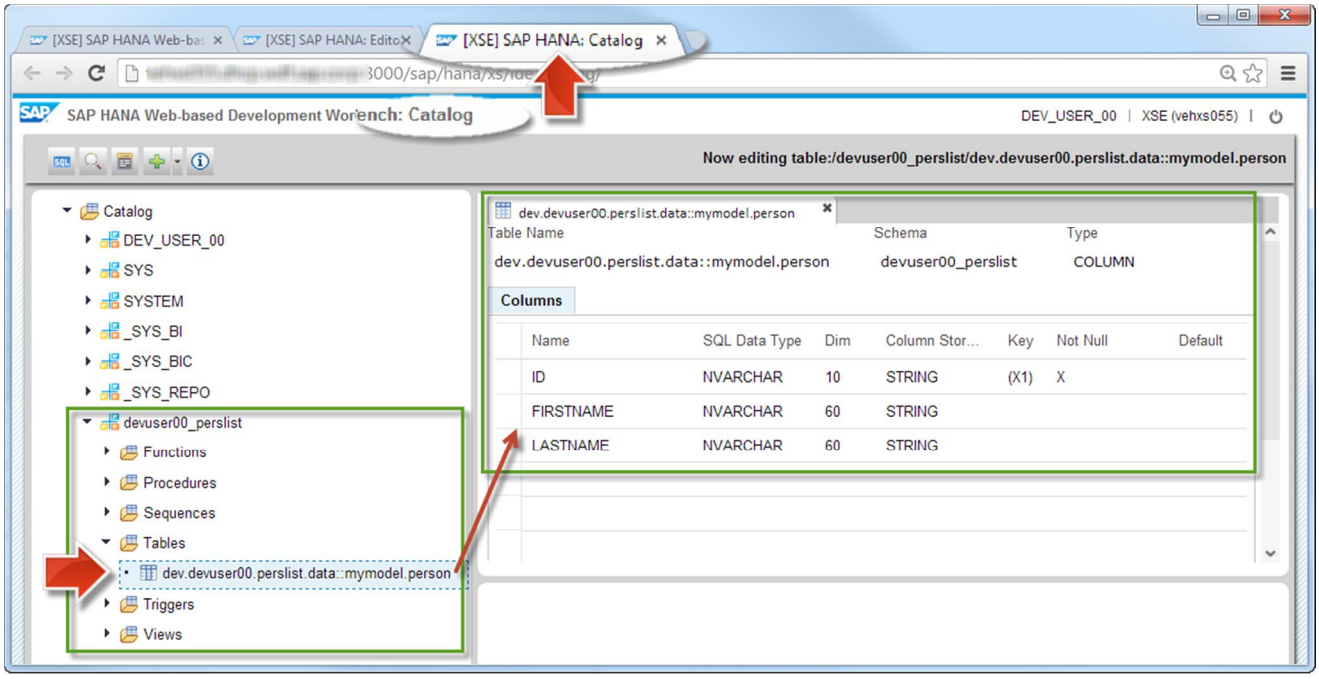
Result

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 **dev.devuserxx.perslist.data::mymodel.person,** in database schema devuserxx_perslist. You can open the catalog of the HANA Web IDE to display both objects.

see Screenshot 4 on next page …

Screenshot 4: Automatic database table creation in SAP HANA catalog schema on activation of data definition file mymodel.hdbdd



NOTE: Activation would also fail when the name of the **.hdbdd** file is not the same as the name of the context. In the example, you define the context "**mymodel**" in the CDS document **mymodel.hdbdd**.

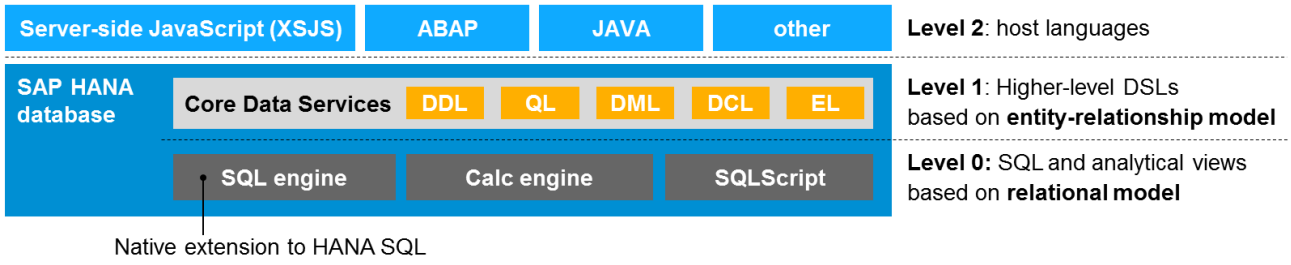
previous next

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.
- *Query Language (QL)* to conveniently and efficiently read data based on data models as well as to 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: 4.7 Data Persistence Objects in Core Data Services (CDS)
- SAP HANA Academy: SAP HANA SPS 6 - What's New: Development - Core Data Services, by Tom Jung

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

previous next

Hands-on Tasks

29. In the Editor's repository content tree select package **Content → dev → devuserxx → perslist → data**
30. Select context menu item ***Create File*** and enter filename `pers.hdbsequence`**.**
31. Enter Source Code 5 to define a database sequence and replace the `highlighted strings` with your own schema/package names.

Source Code 5: Database sequence file pers.hdbsequence

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

32. Save the new sequence file.

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: 4.4 Creating Sequences

3.2.5 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 (see next section 3.2.5.2). On activation the person table (*dev.devuserxx.perslist.data::mymodel.person*) gets automatically filled with mock data that is read from the csv file.

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

previous next

Hands-on Tasks

33. In Editor tab select repository package **Content → dev → devuserxx → perslist → data**
34. Create new sub-package `loads` with context menu item ***Create Package***
35. Select the newly created package ***loads*** and choose menu item ***Create File.***
36. Enter `pers.csv` as file name
37. Enter a list of three comma-separated person entries (see Source Code 6):

Source Code 6: pers.csv file to import mock data from a list of comma separated values

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

38. Save the **pers.csv** file.

Result

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

previous next

3.2.5.2 Add Table Import Definition File *pers.hdbti*

previous next

Hands-on Tasks

39. Select package ***loads*** and choose menu item ***Create File.***
40. Enter `pers.hdbti` file name to add a new table import definition
41. Enter the import statement from Source Code 7into the **pers.hdbti** file and replace the `highlighted strings` with your own database user name:

Source Code 7: pers.hdbti file to import table data from .csv file

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

42. Save the new **pers.hdbti** file.
43. 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

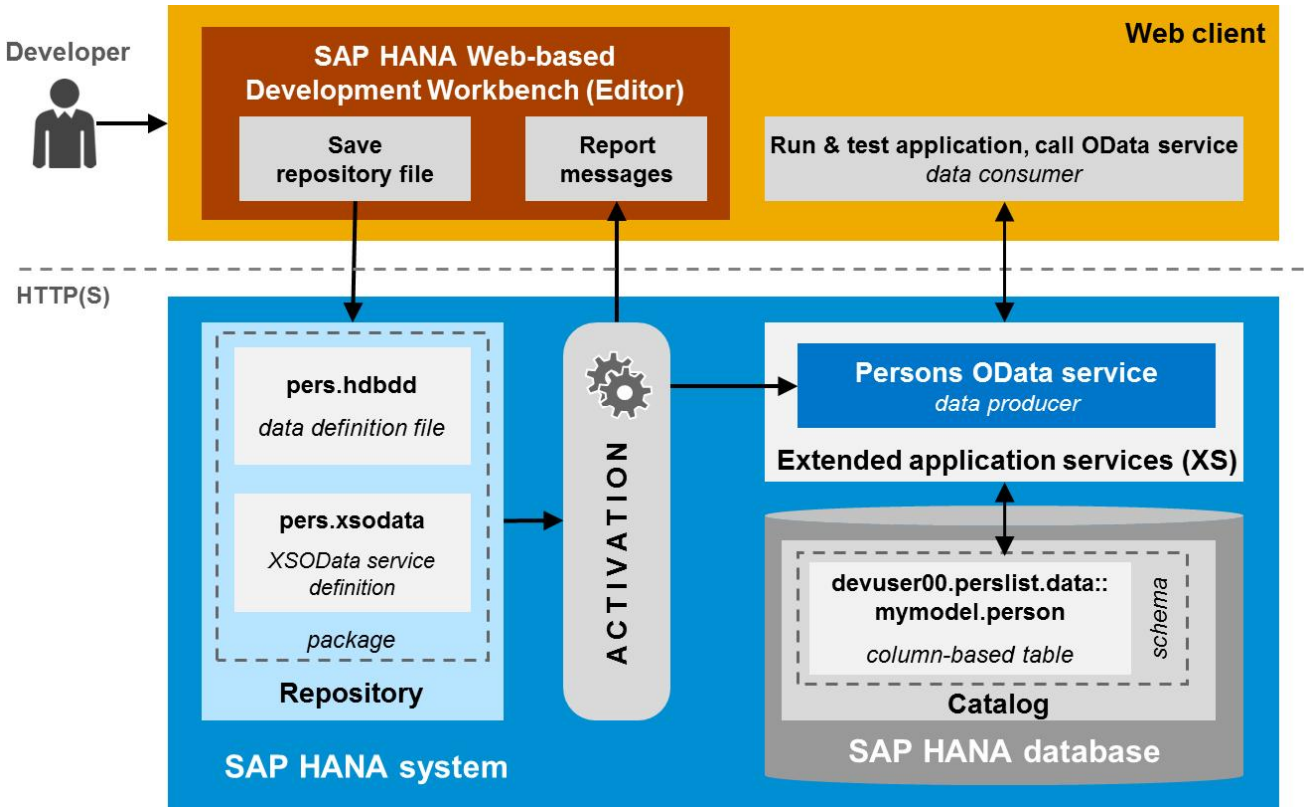
The **pers.hdbti** file which connects the preload content **pers.csv** file with the person table has been saved and by activating it the preload content will be inserted into the person table.

previous next

What happens on Activation of SAP HANA Repository Content?

Figure 5 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 5: 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

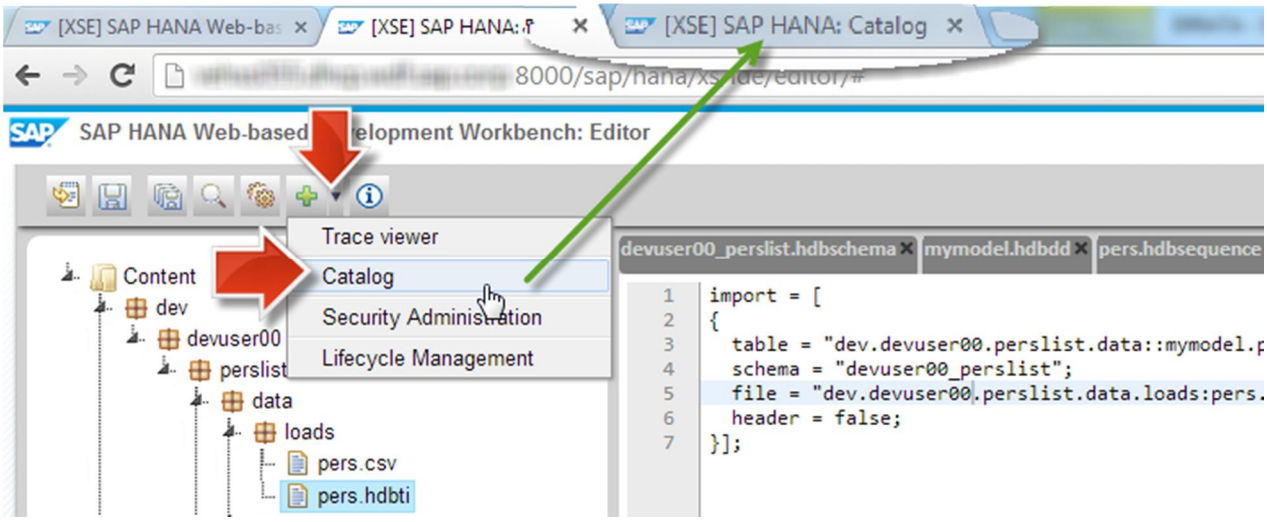
3.2.5.3 Test Initial Load of Person Table Data in SAP HANA Catalog

The previously activated table import definition file *pers.hdbti* implies the automatic initial load of csv-file data into the persons table (see Figure 5). 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

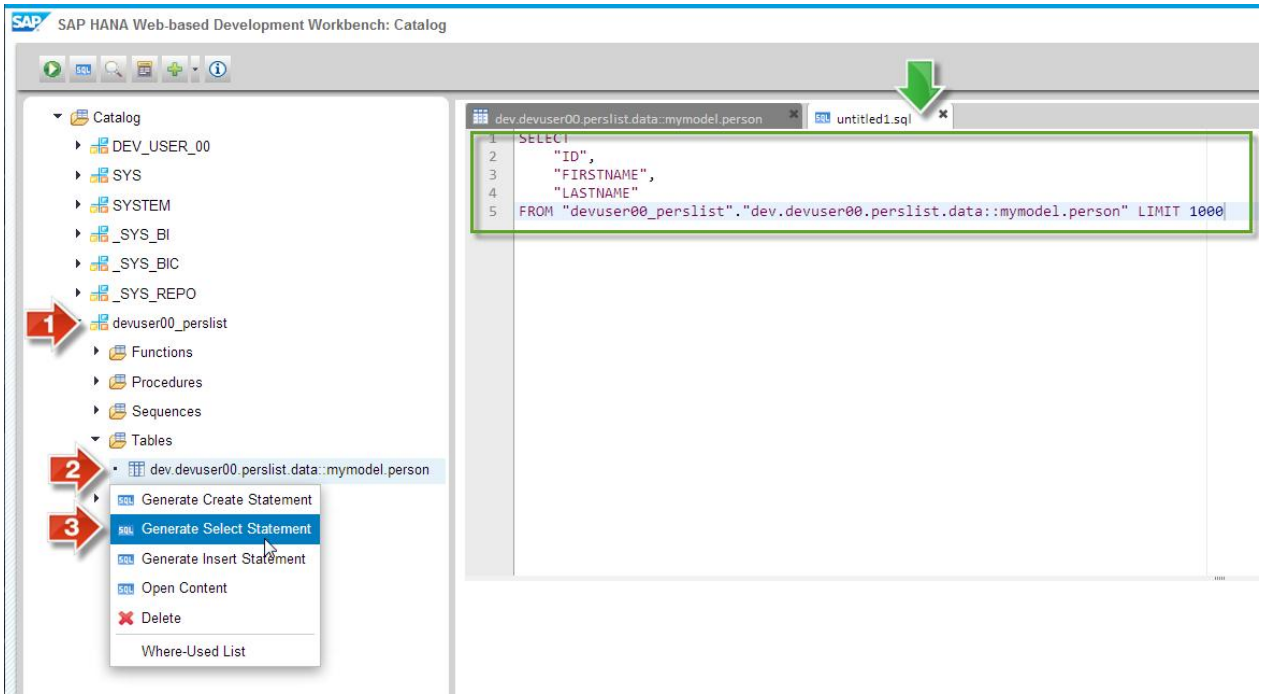
Hands-on Tasks

44. Open the SAP HANA catalog in a new browser tab by pressing toolbar button *More* and selecting menu item **Catalog.**



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

46. 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.

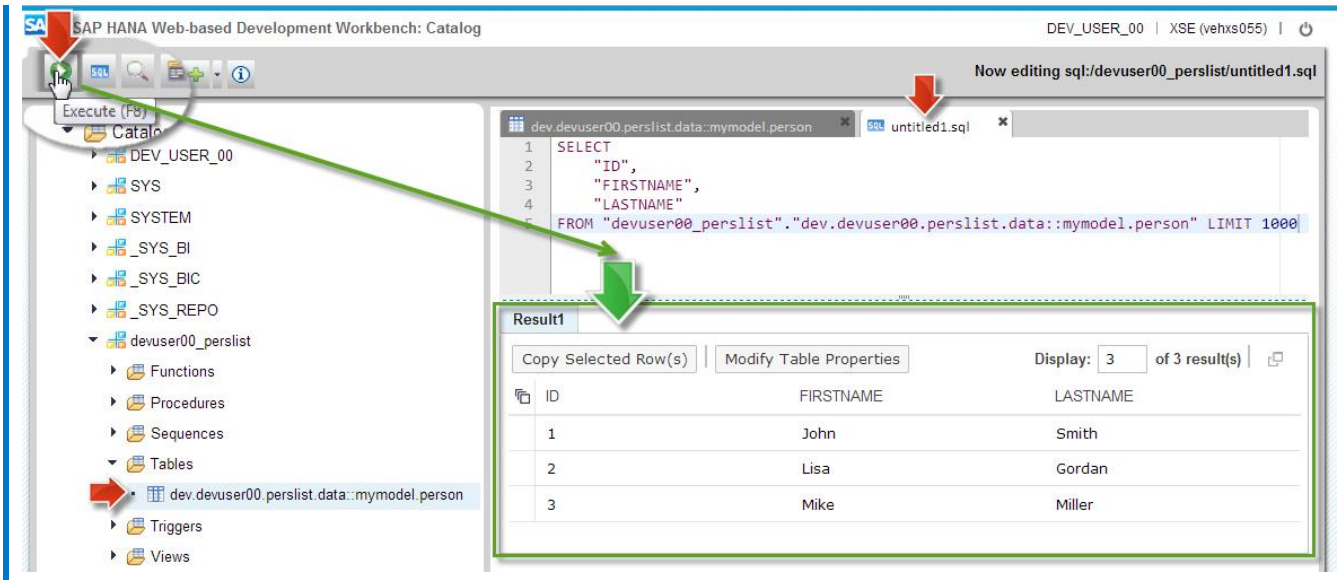


47. After successful initial load from the **pers.csv** file in the last step the SQL SELECT should return three records. To execute the generated SQL STATEMENT press keyboard shortcut **F8** or the execute button  in the catalog toolbar.

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 SQL select statement.



previous next

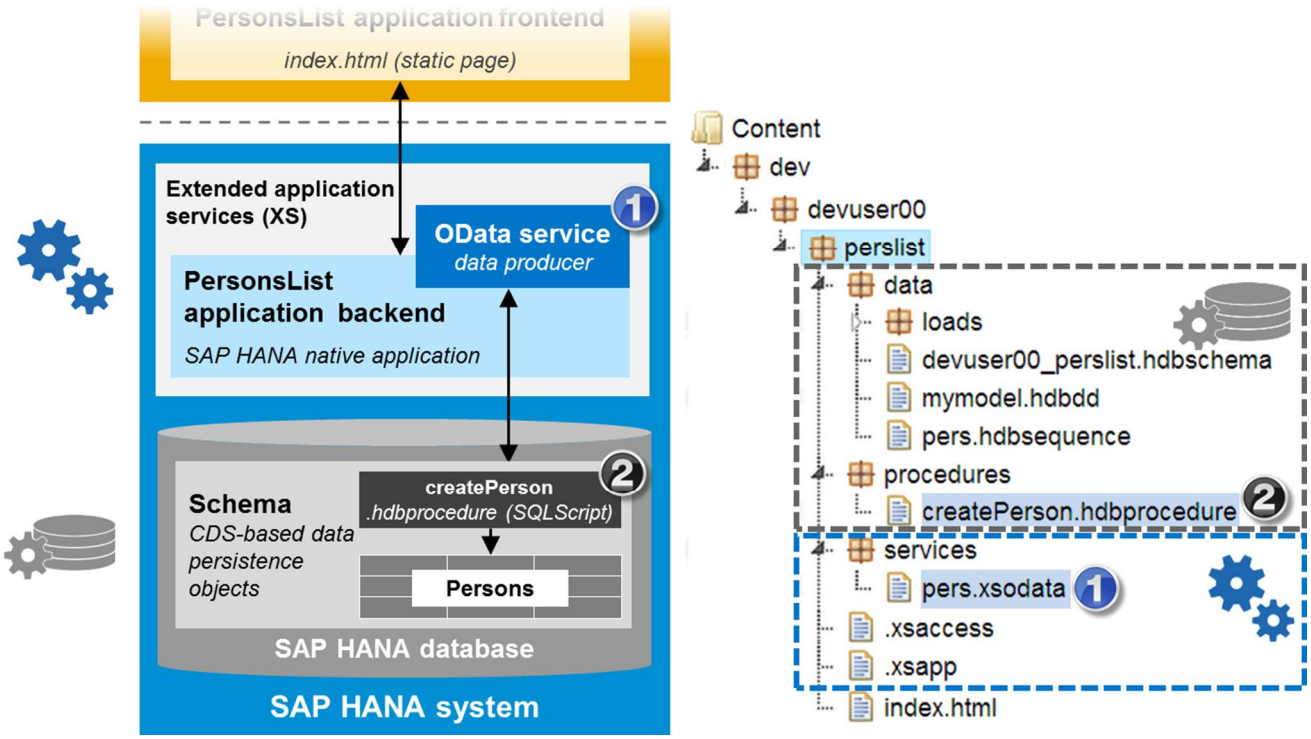
3.3 Step 3: Build the Application Backend with SAP HANA Extended Application Services

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)*.

Preview

Figure 6: 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
.xsodata	OData descriptor	A design-time object that defines an OData service that exposes SAP HANA data from a specified end point.
.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.

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, see section 3.2.2.2.

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:**

48. In editor tab select repository package **Content → dev → devuserxx → perslist**
49. 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 only need to define …

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

50. Under new sub-package **services** add new file `pers.xsodata`
51. Add the following code to the **pers.xsodata** file:

Source Code 8: OData service definition file pers.xsodata for service consumption of entity Person

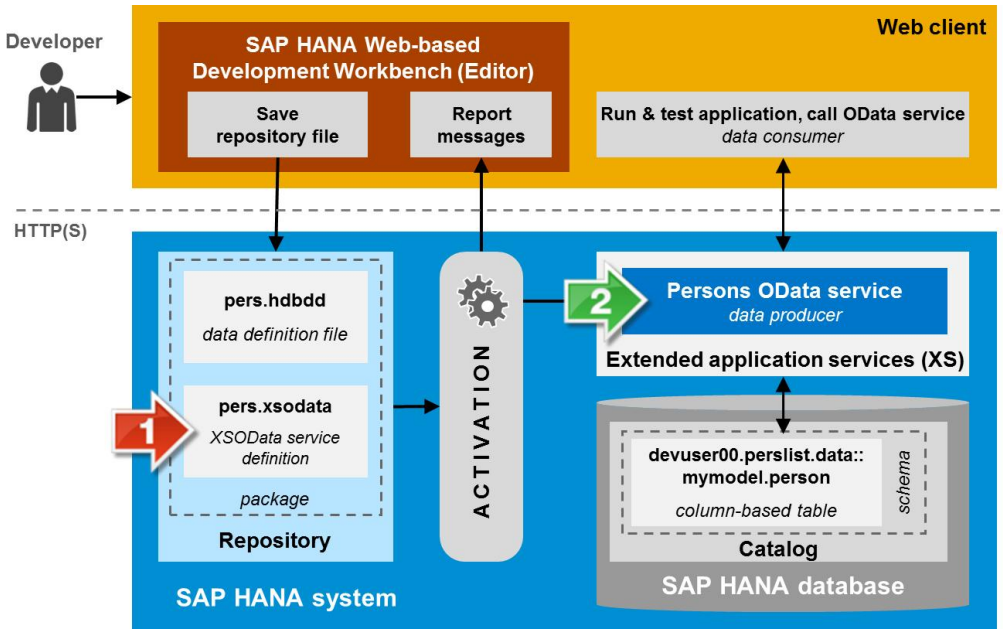
```
service {
  "dev.devuserxx.perslist.data::mymodel.person" as "Persons";
}
```

52. Save the new XSOData service descriptor

Result

On activation of the **pers.xsodata** repository file (Figure 7, 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 7, arrow 2).

Figure 7: 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.

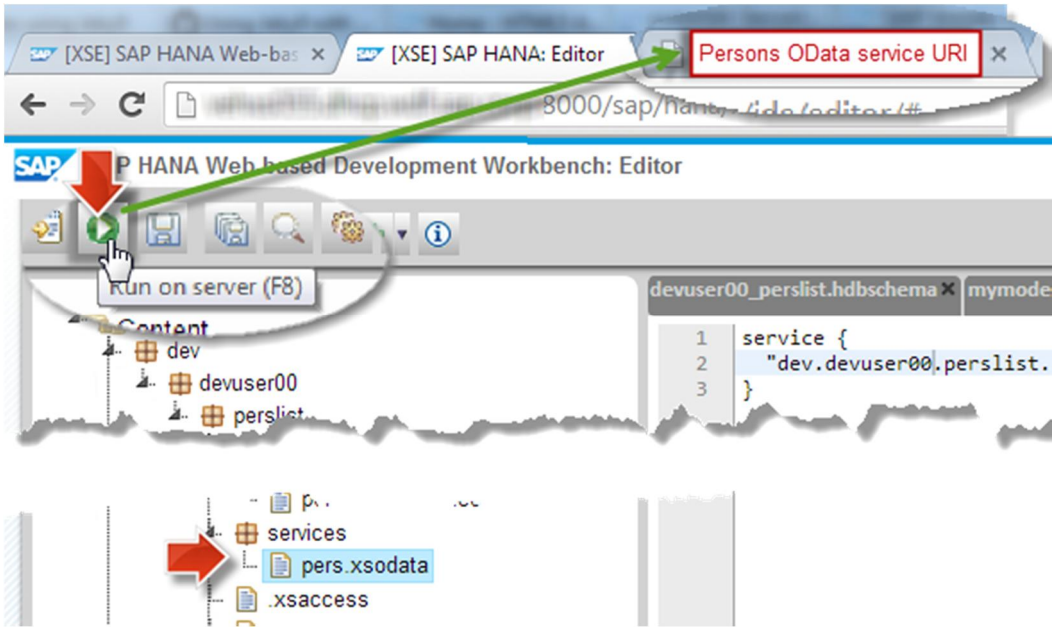
previous next

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

53. Select the **.xsodata** repository file **Content → dev → devuserxx → perslist → perslist → pers.xsodata** and press toolbar button  *Run on server* (or use keyboard shortcut **F8**) to call the *Persons* OData service in a new browser tab.



The HANA Web IDE passes the root URI of the *Persons* OData service to a new browser tab and executes an http request ((in general `https://<Web server host name>:80<SAP HANA instance>/dev/<dev user's package name>/perslist/pers.xsodata`).

The correctly 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"] } }`

54. Add to the browser URL the following string `/Persons?$format=xml` and press the Refresh button (or keyboard shortcut **F5**): `http://...perslist/services/pers.xsodata?$format=json`

    An example of the response returned for the Persons query is shown below. In the 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 an SQLScript procedure with signatures that follow specific conventions. SAP HANA XS supports two type 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

55. 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 → dev → devuserxx → perslist → data → mymodel.hdbdd** to open the editor tab again.
56. 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**.

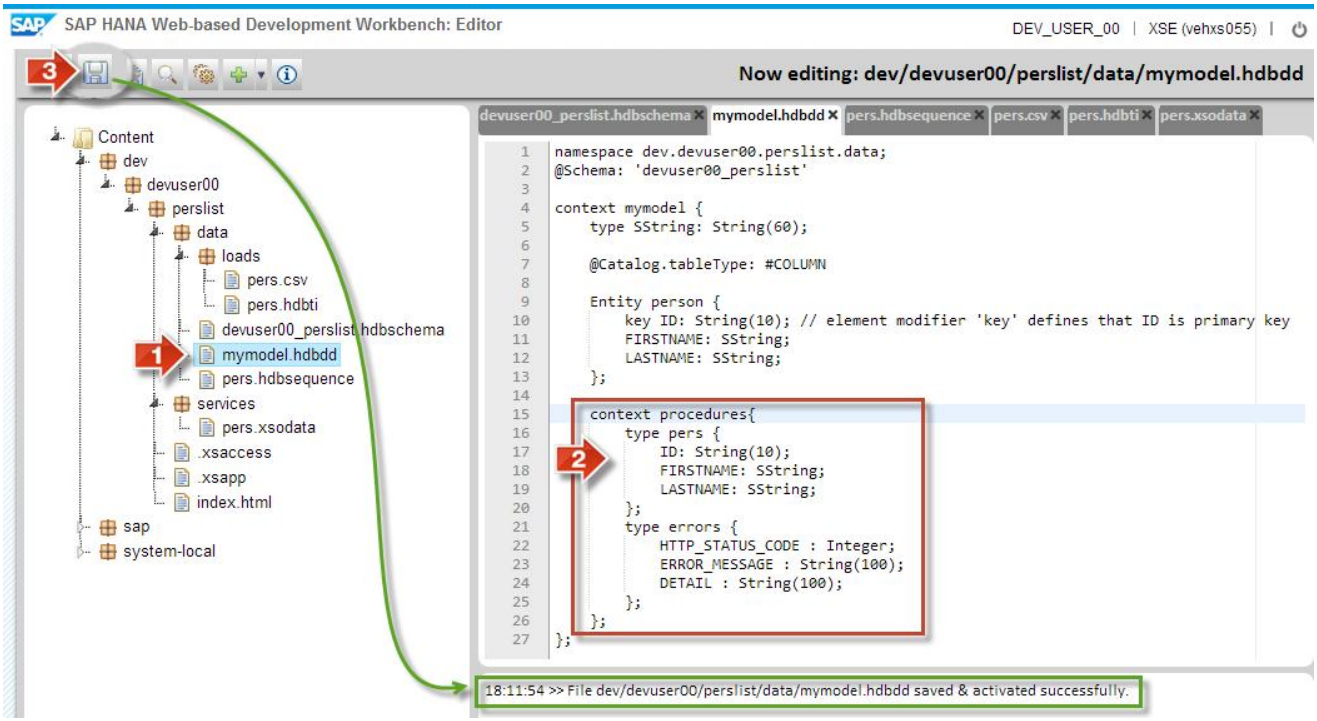
Source Code 9: New context 'procedures' with CDS-user-defined datatypes pers and errors in CDS Data Definition File mymodel.hdbdd

```
namespace dev.devuserxx.perslist.data;
@Schema: 'devuserxx_perslist'

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);
        };
    };
};
```

57. 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.



previous next

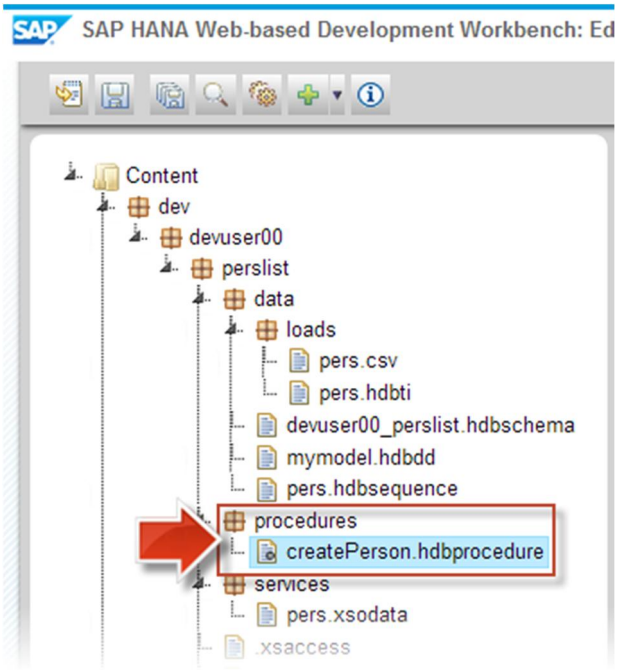
3.3.2.2 Implement Modification Exit in SQLScript Procedure to add new Records in Persons Table

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

previous next

Hands-on Tasks

58. In editor tab select repository package **Content → dev → devuserxx → perslist**
59. Create new sub-package `procedures` with context menu item ***Create Package***
60. Under new sub-package procedures create new file `createPerson.hdbprocedure`



61. Copy Source Code 10 with the procedure logic to insert new records into the Persons table:

Source Code 10: SQLScript Procedure createPerson.hdbprocedure to insert table records (via modification exit for OData service)

```
PROCEDURE
    "devuserxx_perslist"."dev.devuserxx.perslist.procedures::createPerson" (1)
        IN intab "devuserxx_perslist"."dev.devuserxx.perslist.data::mymodel.procedures.pers", (2)
        OUT outtab "devuserxx_perslist"."dev.devuserxx.perslist.data::mymodel.procedures.errors"
    )
  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 "dev.devuserxx.perslist.data::mymodel.person" (3)
         values ("dev.devuserxx.perslist.data::pers".NEXTVAL, lv_firstname, lv_lastname); (4)
end if;

end;
```

NOTE: Know the code

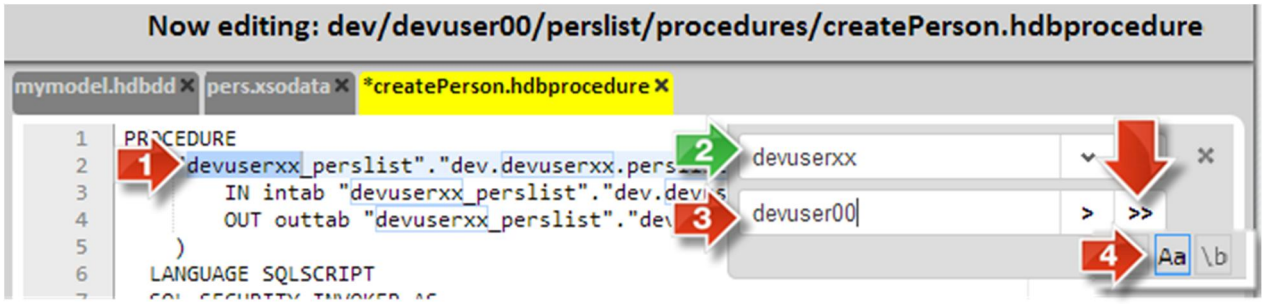
(1) Fully qualified name of SQLScript procedure `createPerson` starts with schema name `"devuserxx_perslist"` followed by repository package name `"dev.devuserxx_perslist.procedures"`

(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 10).

(3) Reference to entity `"dev.devuserxx.perslist.data::mymodel.person"` defined in CDS file **mymodel.hdbd**

(4) Reference to database sequence `"dev.devuserxx.perslist.data::pers"` defined in file **pers.hdbsequence**, see Source Code 5.

62. To replace the corresponding user string `devuserxx` (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 `devuserxx`
- 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 `devuserxx`. Enter your user-specific package name e.g. `devuser00` into the second input field.
- Press **Aa** button to assure case sensitive search

- Press **>>** button to replace all occurrences of the search string `devuserxx`.
- 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).

63. Save the new SQLScript procedure to activate it.

Result

All strings `devuserxx` 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

64. 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 11: Add modification exit to call SQLScript procedure createPerson.hdbprocedure in OData service

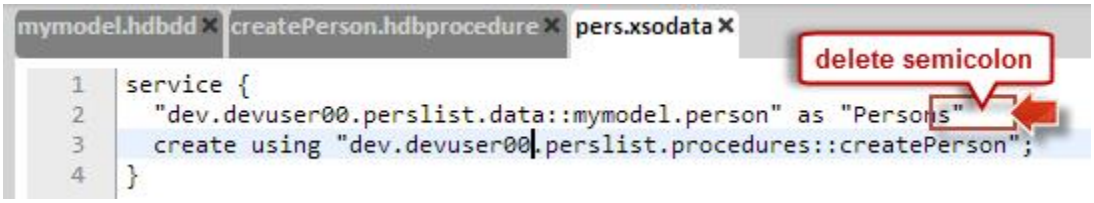
```
service {
  "dev.devuserxx.perslist.data::mymodel.person" as "Persons"
    create using "dev.devuserxx.perslist.procedures::createPerson";
}
```

add this code line

65. **Remove the semi-colon** at the end of the existing code line the service definition file.
66. Then copy the bold marked line `create using …` and paste into the **pers.xsodata** file before the closing curly bracket.
67. In the new codeline replace `devuserxx` with your user name.
68. 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

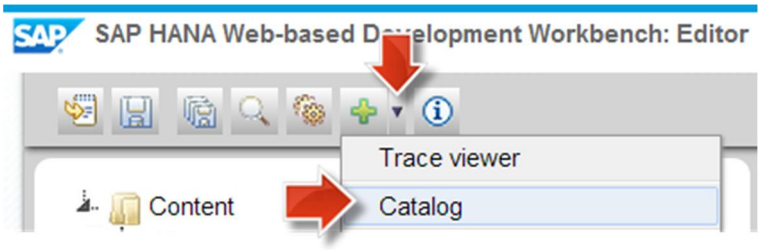
3.3.3 View the Application Backend in SAP HANA Catalog

previous next

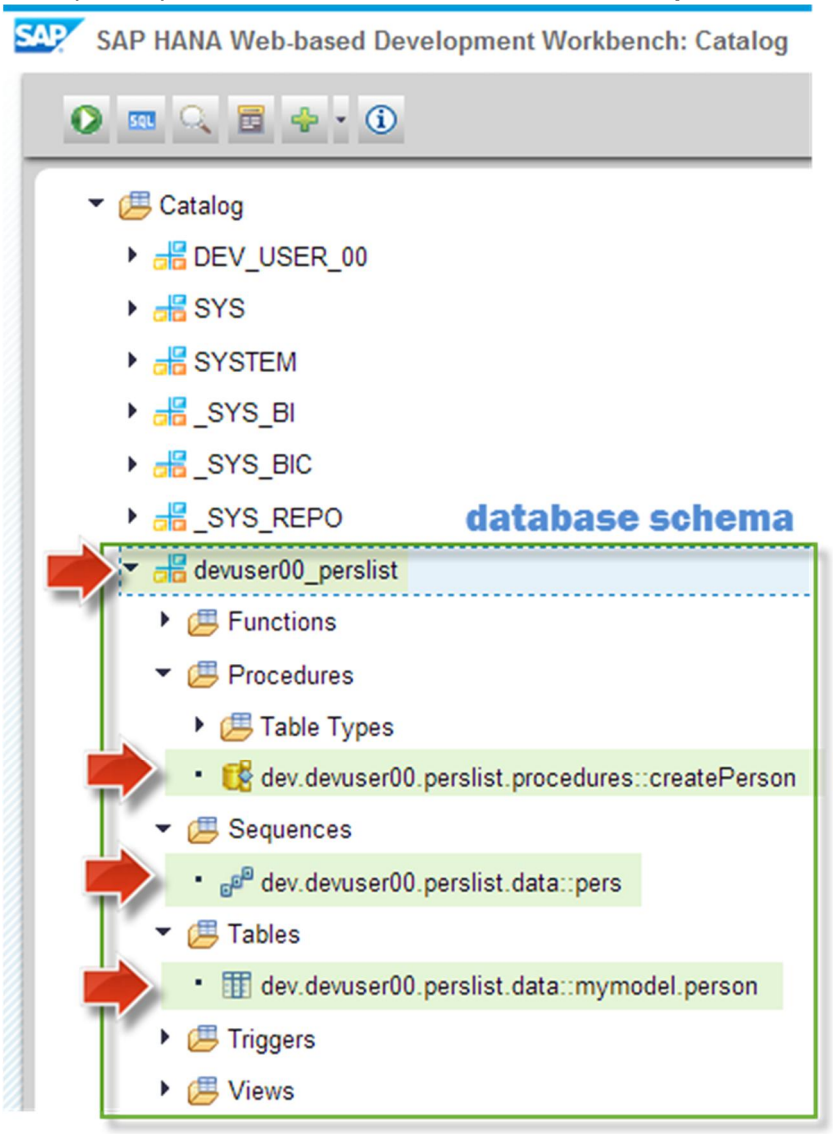
Hands-on Tasks

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

Catalog.



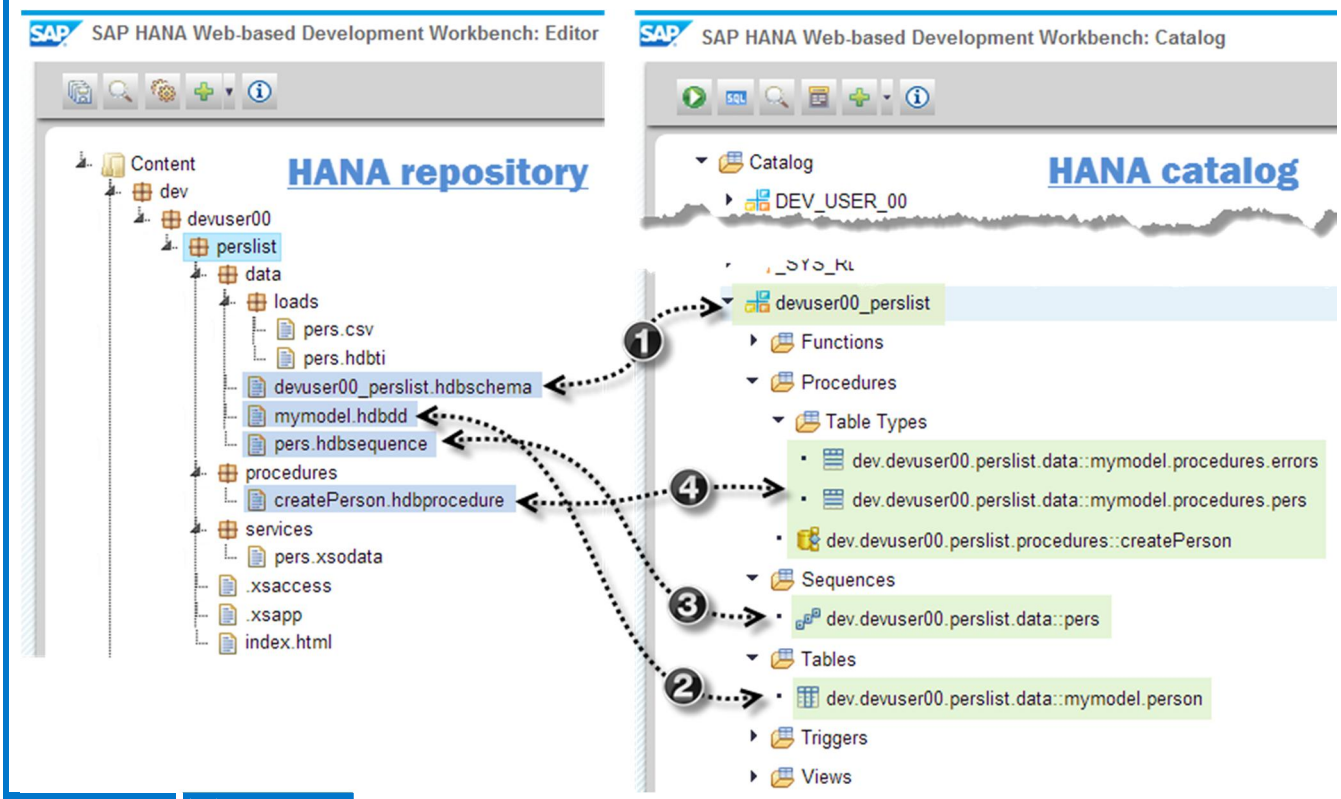
70. The HANA catalog tree gets displayed in a new browser tab. Under tree node **Catalog > devuserxx_perslist** (= new database schema where the runtime artifacts of your *PersonsList* application reside) for expand the three sub-nodes **Procedures**, **Sequences** and **Tables.**



Result

In database schema `devuserxx_perslist` 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 (see table on next page):

	Repository content	Catalog content
1	Schema definition file `dev/devuserxx/perslist/data/devuserxx_perslist.hdbschema`	`devuserxx_perslist`
2	CDS-based data definition file `dev/devuserxx/perslist/data/mymodel.hdbdd`	`devuserxx_perslist/Tables/dev.devuserxx.perslist.data::mymodel.person`
3	Sequence definition file `dev/devuserxx/perslist/data/pers.hdbsequence`	`devuserxx_perslist/Sequences/dev.devuserxx.perslist.data::personsSeq`
4	SAP HANA database procedure file `dev/devuserxx/perslist/procedures/createPerson.hdbprocedure`	`devuserxx_perslist/Tables/dev.devuserxx.perslist.procedures::createPerson`



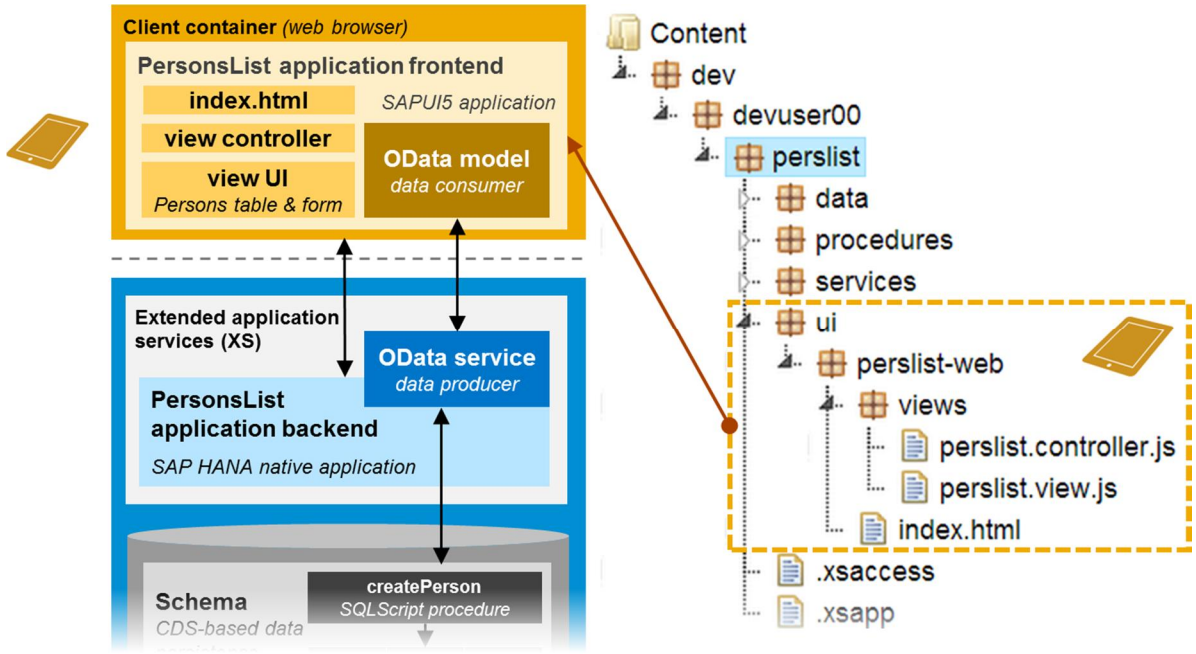
previous next

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



Design-Time Application Artifacts Created in this Step

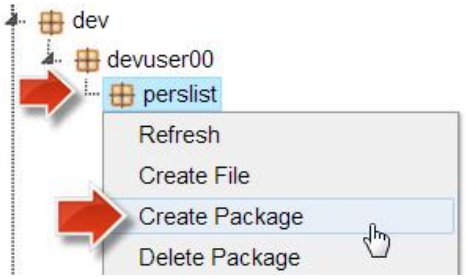
File extension	Object	Description
index.html	SAPUI5 application entry page	HTML file containing bootstrap script (to load SAPUI5 libraries and theme at the client), application script (to load the application's view layout and controller logic) and html body (to embed views).
.view.js	SAPUI5 JavaScript 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

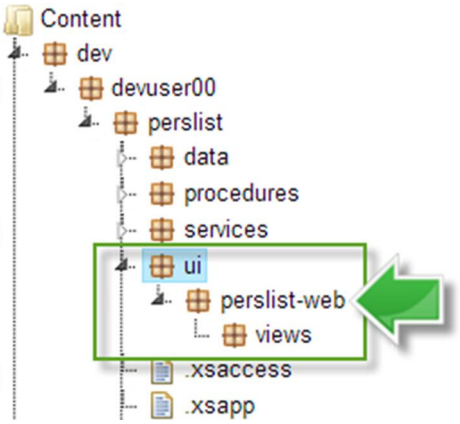
71. In editor tab select repository package **Content→ dev → devuserxx → perslist**



72. Create new sub-packages **ui**, **perslist-web** and **views** by choosing context menu item **Create Package**
73. 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*).



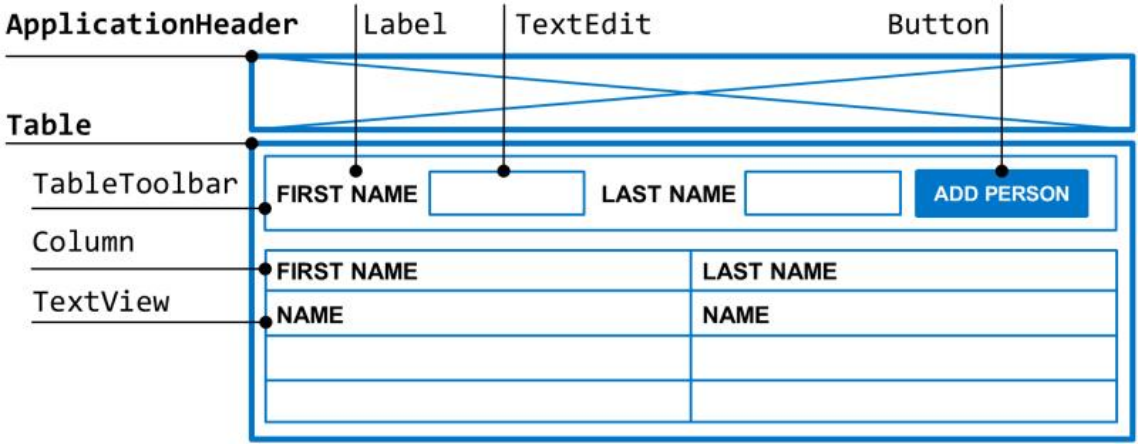
previous next

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 9: 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

74. Select package **dev/devuserxx/ui/perslist-web/views** and choose context menu item **Create File**
75. Enter new file name `perslist.view.js` (UI5 JavaScriptView for the view layout implementation)
76. Copy Source Code 12 into the **perslist.view.js** editor

copy Source Code 12 *from next page …*

Source Code 12: SAPUI5 application UI implemented in JavaScriptView file perslist.view.js

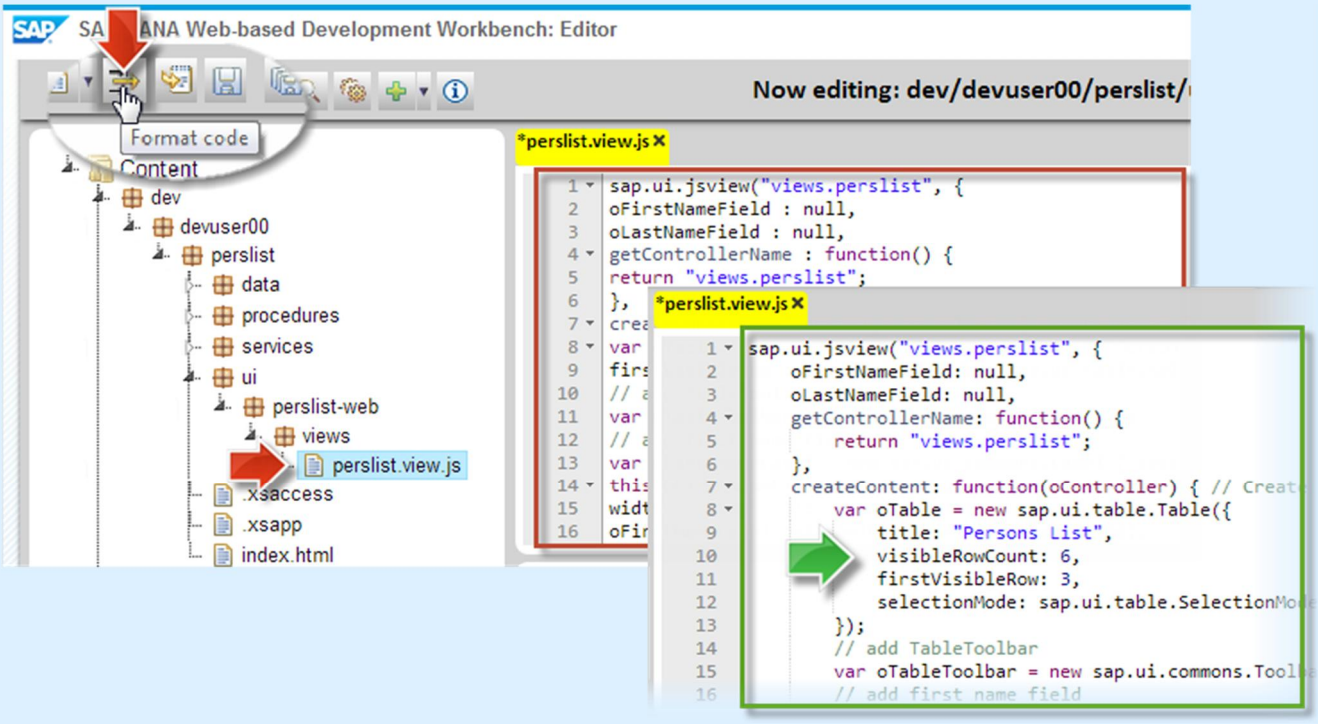
```
sap.ui.jsview("views.perslist", {

  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).

77. 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

78. Select package **dev/devuserxx/ui/perslist-web/views** and choose context menu item **Create File**
79. Enter new file name `perslist.controller.js` (view controller logic implemented in JavaScript)

80. Copy Source Code 13 into the **perslist.controller.js** editor:

Source Code 13: 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
        + "/dev/devuserxx/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)
  }
});
```

NOTE: Know the code

(1) Generate absolute URL of OData service `pers.xsodata` added in section *3.3.1 Expose the Person Database Entity by means of OData Service.*

(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.

81. Replace string `devuserxx` with your own package name.

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

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

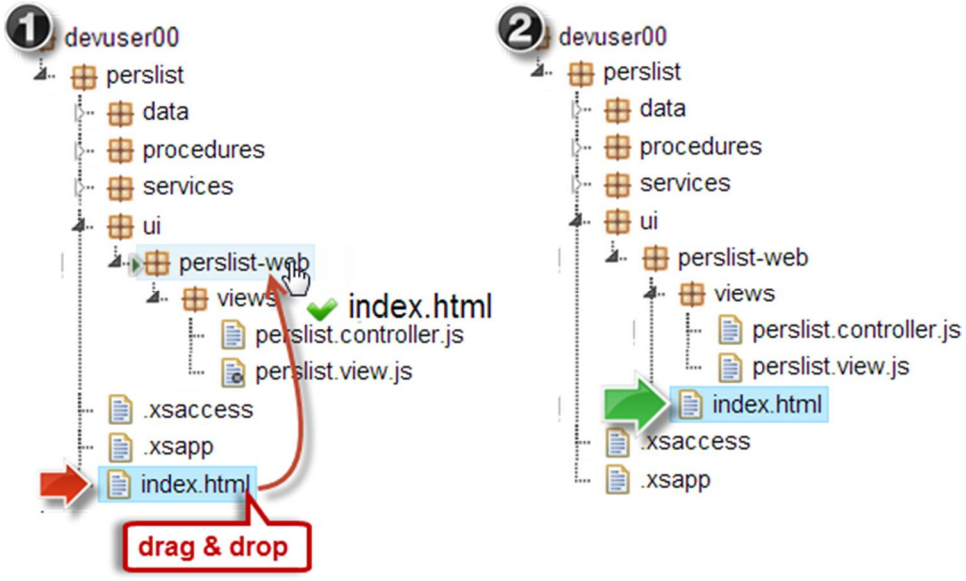
previous next

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

previous next

Hands-on Tasks

83. 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.



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

85. After the drag & drop the **index.html** editor is opened. In the editor tab remove the entire content and replace it with Source Code 14:

Source Code 14: 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 Platform E2E Dev Scenario: SAP HANA native
  PersonsList application</title>
<script src="/sap/ui5/1/resources/sap-ui-core.js" id="sap-ui-bootstrap" 1
  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");2

  var oAppHeader = new sap.ui.commons.ApplicationHeader("appHeader");
  oAppHeader.setLogoSrc("http://www.sap.com/global/images/SAPLogo.gif");
  oAppHeader.setLogoText("SAP HANA 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",3
    type : sap.ui.core.mvc.ViewType.JS });
  view.placeAt("content");
</script>

</head>
<body class="sapUiBody" role="application"> 4
  <div id="header"></div>
  <div id="content"></div>
</body>
</html>
```

NOTE: Know the code

1. **Bootstrap script:** SAPUI5 is implemented in JavaScript, so for loading the SAPUI5 runtime library `sap-ui-core.js` from SAP HANA repository folder `"/sap/ui5/1/resources/"` at the client, its bootstrap just needs to be included with a `<script>` tag. The last two attributes select the visual design to apply initially and the UI control library/libraries to use.

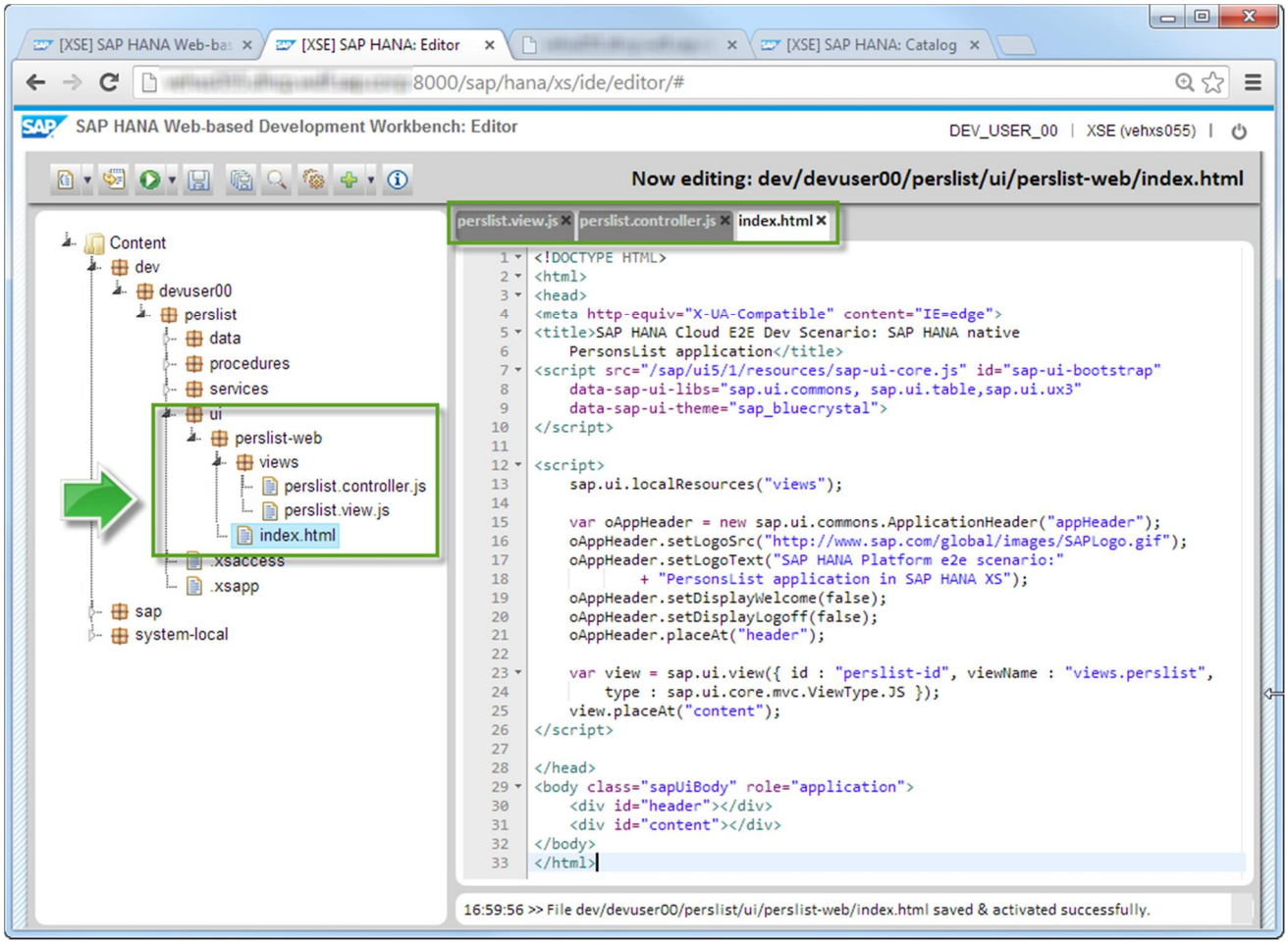
2. **Application script:** SAPUI5 is based on the MVC paradigm. To create view and controller, the SAPUI5 runtime first needs to know where to load the related resources (here in relative subfolder `"/views"`).
3. The newly created instance of the `perslist` view in sub-folder `views` is then placed (like a control) into a HTML element with id `"content"`. SAPUI5 knows different view types (JS, XML and JSON); here the JS (JavaScript) view type is used.

4. **Html body:** The HTML element with ID `"content"` where the view was placed at (in application script) must exist somewhere in the HTML page. So we add a corresponding `<div>` block with `id="content"` into the HTML body. The same applies to the *ApplicationHeader* control that is placed at the second `<div>` block with `id="header"`. The `<body>` attribute `class="sapUIBody"` defines the SAPUI5 CSS class to be used, so the page background and some other styles are properly set. Attribute `role="application"` set the WAI-ARIA landmark role.

86. 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:

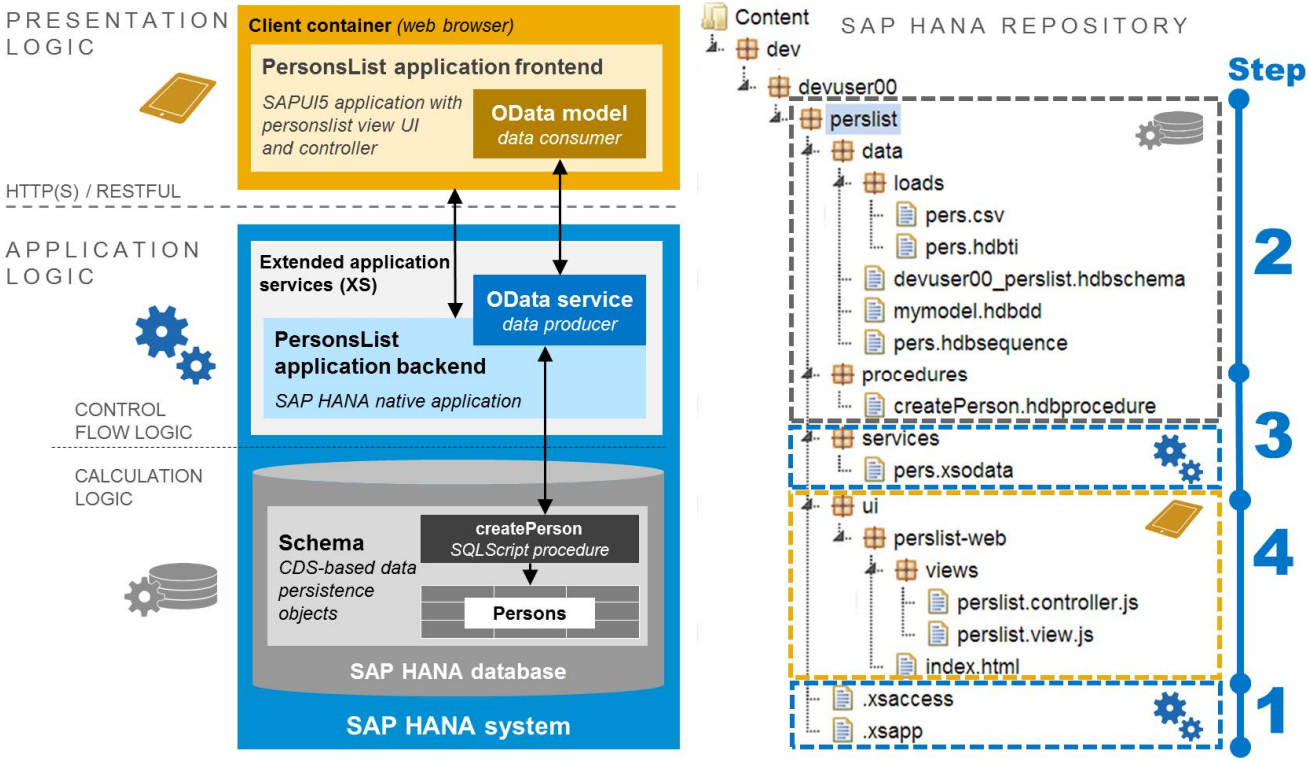
- SAPUI5 Demo Kit - Developer Guide - SAPUI5 HelloWorld
- SAPUI5 Demo Kit - Developer Guide - Bootstrapping

previous next

3.5 Run and Test the Final PersonsList Application

3.5.1 Recap: Anatomy of the Whole PersonsList Application

Figure 10: Anatomy of the whole PersonsList application implemented with SAP HANA extended application services



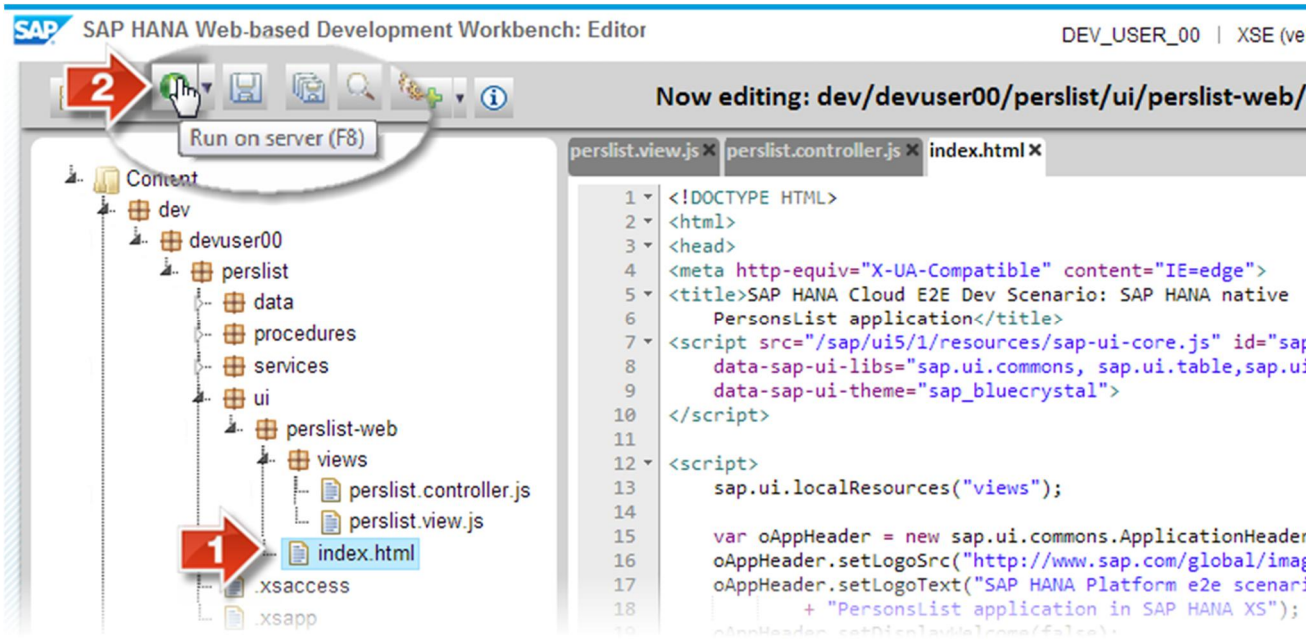
3.5.2 Run PersonsList Application on SAP HANA XS Server

previous next

Hands-on Tasks

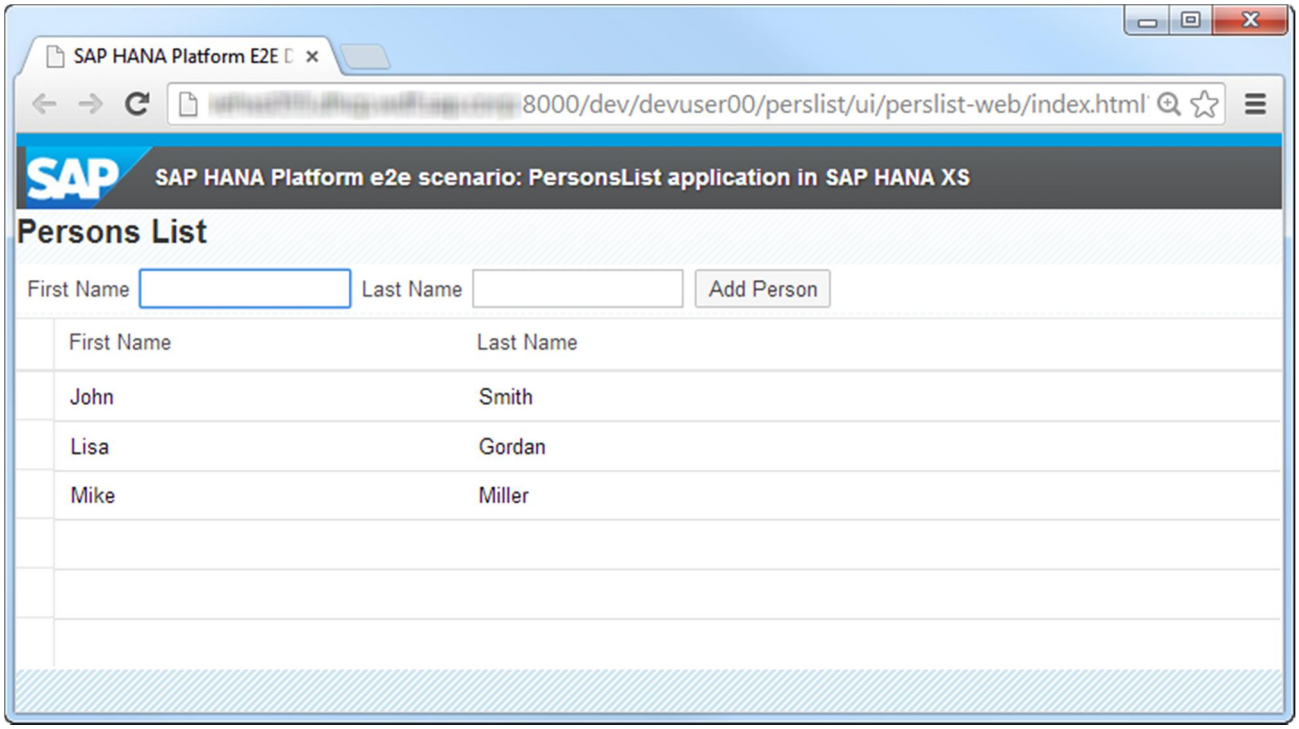
87. In editor tab select repository package **Content → dev → devuserxx → perslist → ui → perslist-web → index.html**

88. 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 XSOData service.



previous next

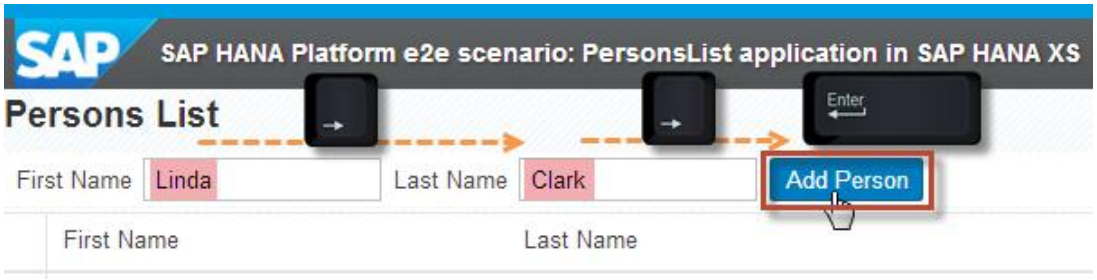
3.5.3 Test PersonsList Application in Web Browser

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

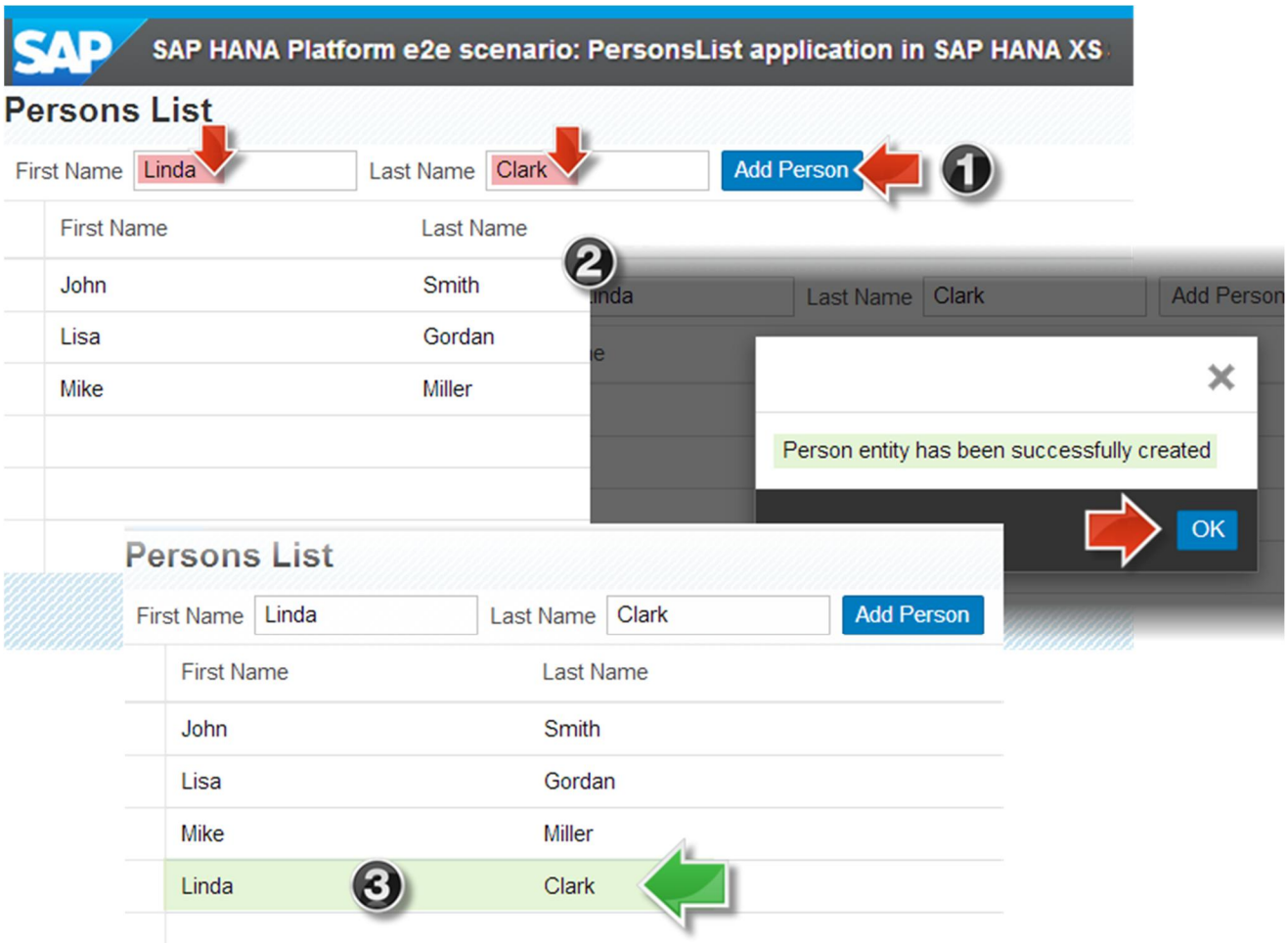
89. Enter first name `Linda` into the first input field that is already focused on application startup.
90. Press the *right arrow* keyboard button to move focus to the next input field ‘*Last Name*’



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).

91. Enter `Clark` in the second input field “Last Name” and press the *right arrow* keyboard button to select the Add **“Person Button”**
92. Press form button **“Add Person**” to send the new person record to the application backend.
93. 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.

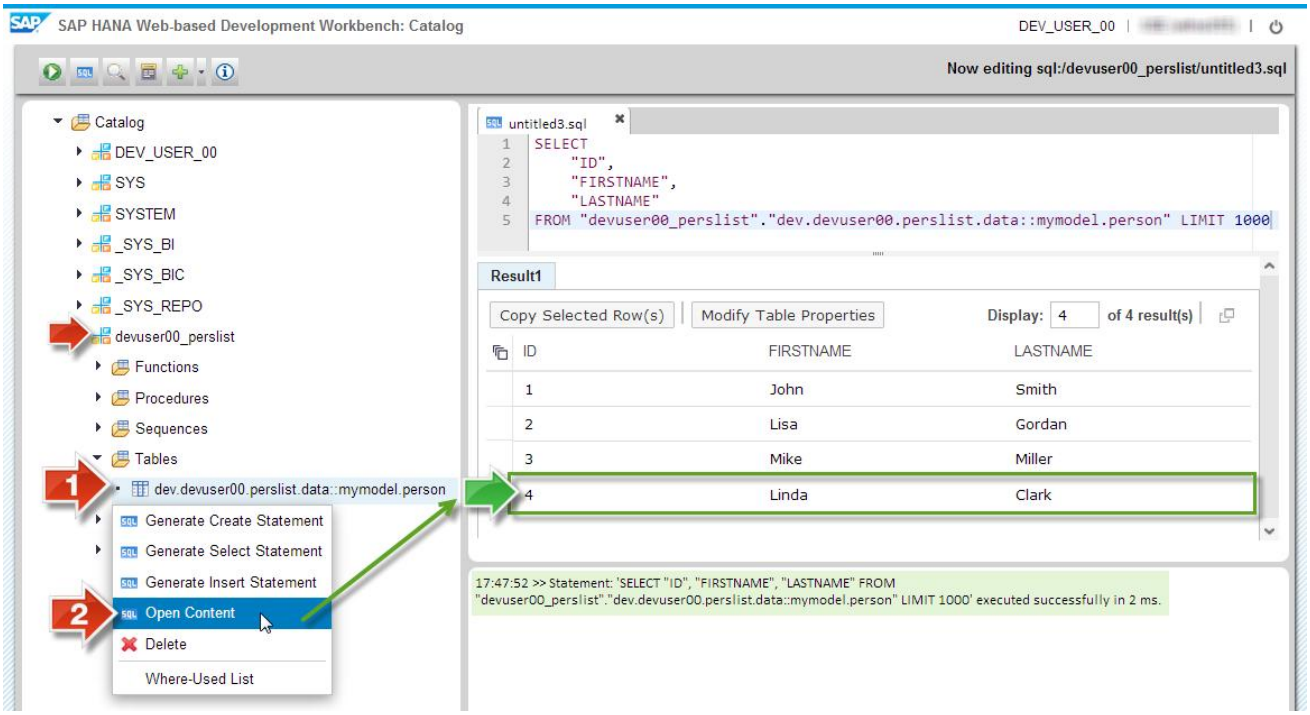
previous next

Hands-on Tasks

94. Open SAP HANA catalog
95. Select node **Catalog → devuserxx_perslist → Tables → dev.devuserxx.perslist.data::mymodel.person**
96. Select context menu item **Open Content**

Result

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



previous next

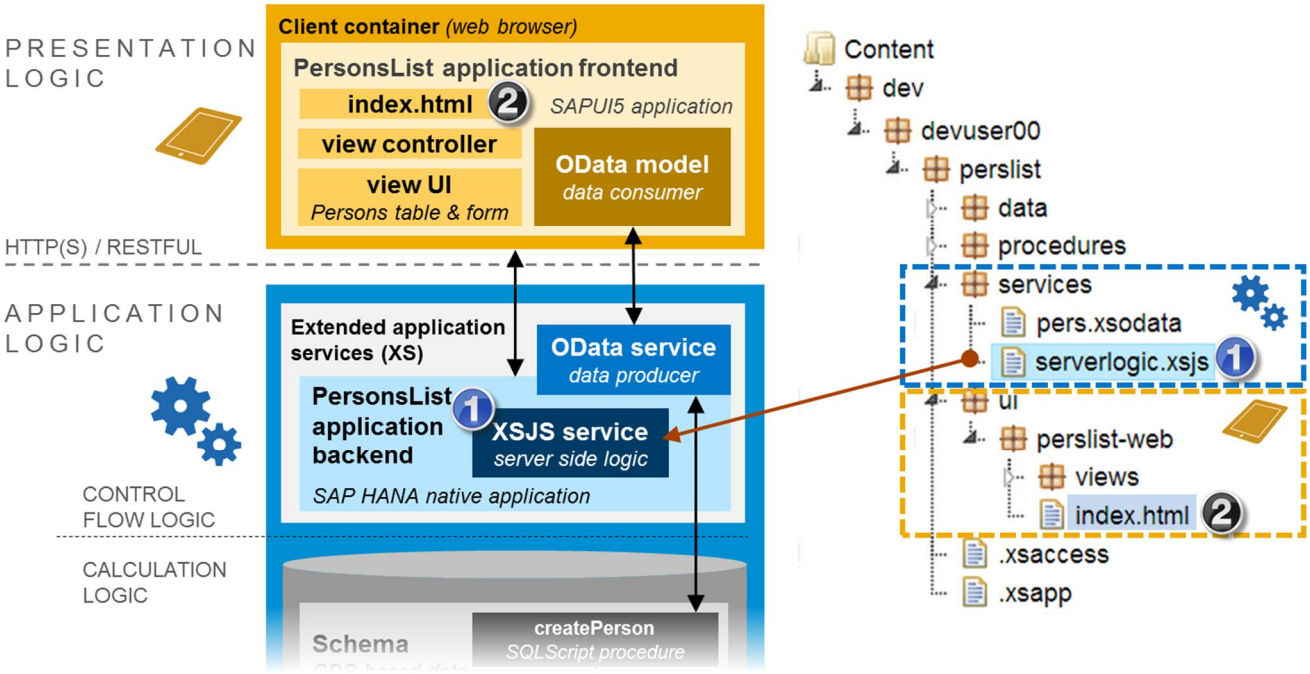
3.6 1. Extension: Writing Server-Side JavaScript Code

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). The added **serverlogic.xsjs** file contains the function `getUserName()` to retrieve the session user. This **serverlogic.xsjs** is called from a script section in **index.html** to display the result, i.e. the logged on user name, as welcome message in the application header.

Preview

Figure 11: 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: 8 Writing Server-Side JavaScript Code

Design-Time Application Artifacts Created in this Step

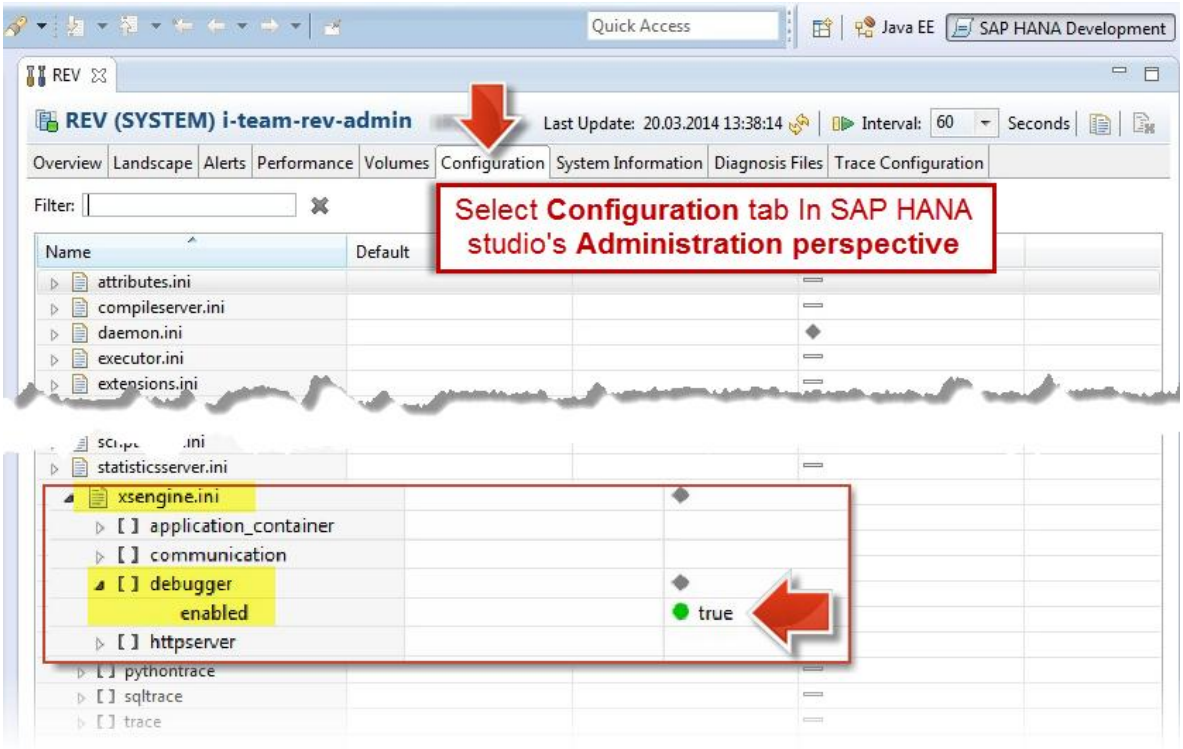
File extension	Object	Description
.xsjs	Server-Side JavaScript Code	A file containing JavaScript code that can run in SAP HANA Extended Application Services and that can be accessed via URL

Start Hands-on (1. Extension)

Prerequisites

- You have already successfully executed the PersonsList HANA XS application. The PersonsList application is running and works as described in chapter 3.5.
- **Done by SAP HANA system administrator**: To use the debugging functions of the HANA Web IDE the section ***debugger*** with the parameter `enabled` and the value `true` must be added to the file `xsengine.ini` in the SAP HANA studio administration perspective, see .

Screenshot 5: SAP HANA system administrator enables the debugger in xsengine.ini configuration



Further information:

- SAP HANA Administration Guides > 2.3 Configuring SAP HANA System Properties
- SAP HANA Developer Guide > 8.9 Debugging Server-Side JavaScript

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.

97. Under the package **services** add a new file **serverlogic.xsjs**
98. Add the following code to the **serverlogic.xsjs** file:

Source Code 15: Server-side JavaScript service definition file serverlogic.xsjs for retrieving the session user name

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

99. 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.

100. In the Editor browser tab (of SAP HANA Web-based Development Workbench) select the file **Content → dev → devuserxx → perslist → ui → perslist-web → index.html** to open the corresponding editor tab. Add the `highlighted code` from next page to the **index.html**.

Source Code 16: 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 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 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.

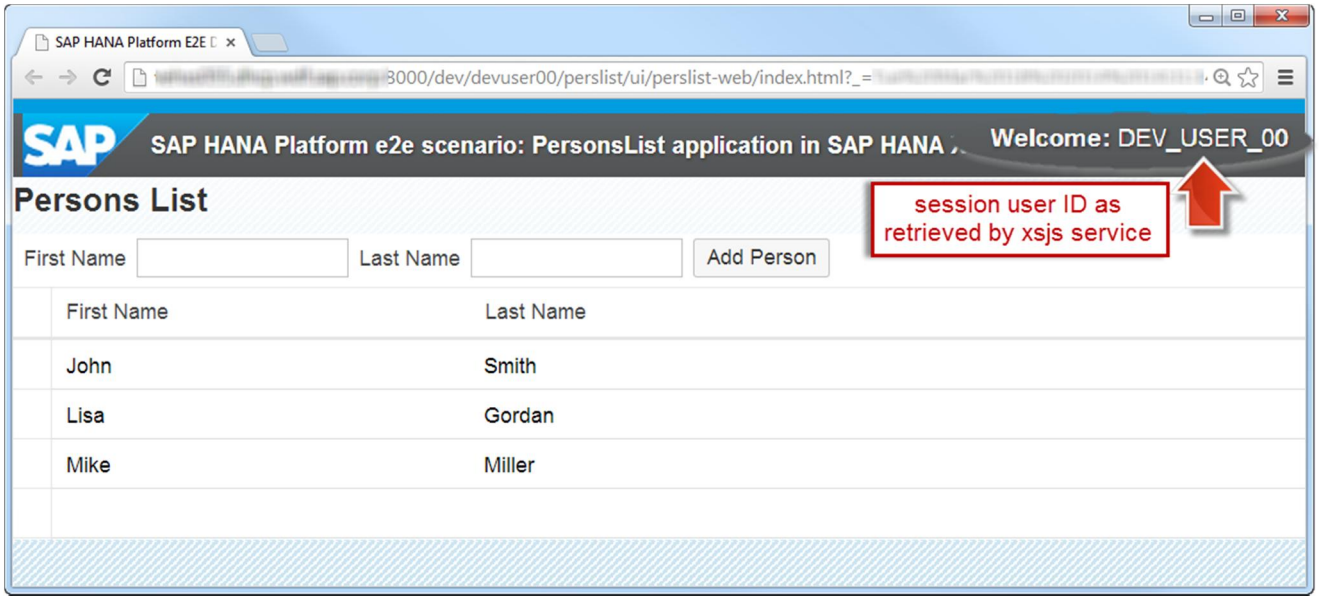
The `$` sign is an alias for the *jQuery* object that is commonly used for jQuery source code like `$( document ).ready()` or `$get()`.

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

102. 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.



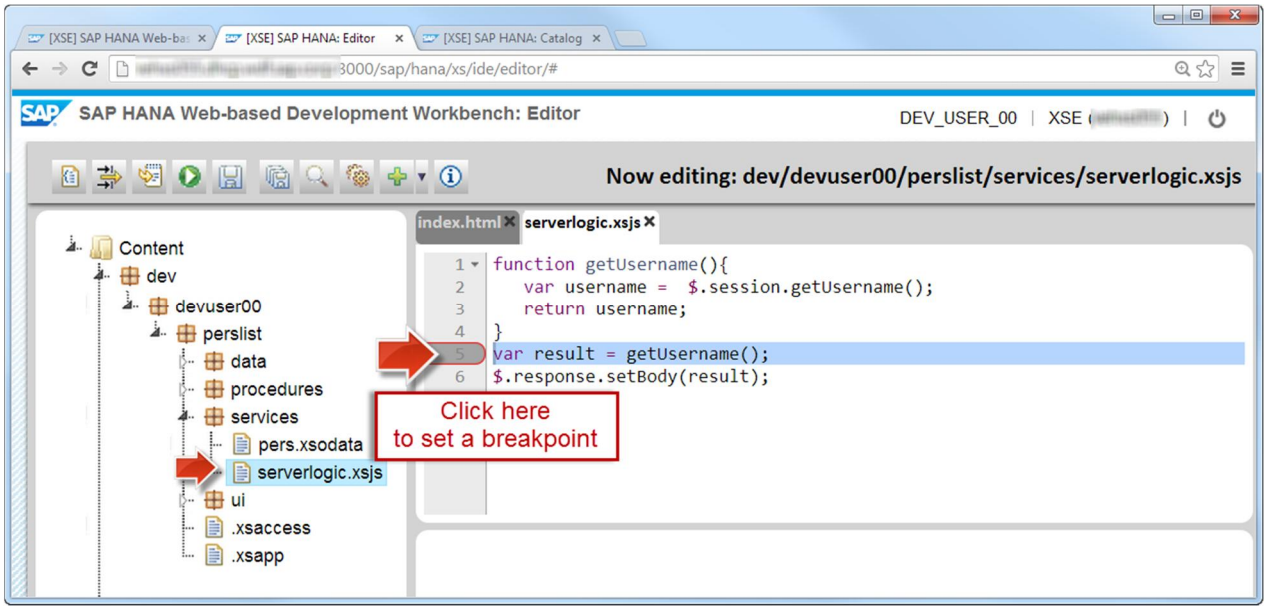
3.6.3 Debugging of HANA XS Server-Side JavaScript Code

As it is quite handy to debug HANA XS server-side JavaScript code we quickly demonstrate how easy it is to debug **.xsjs** code.

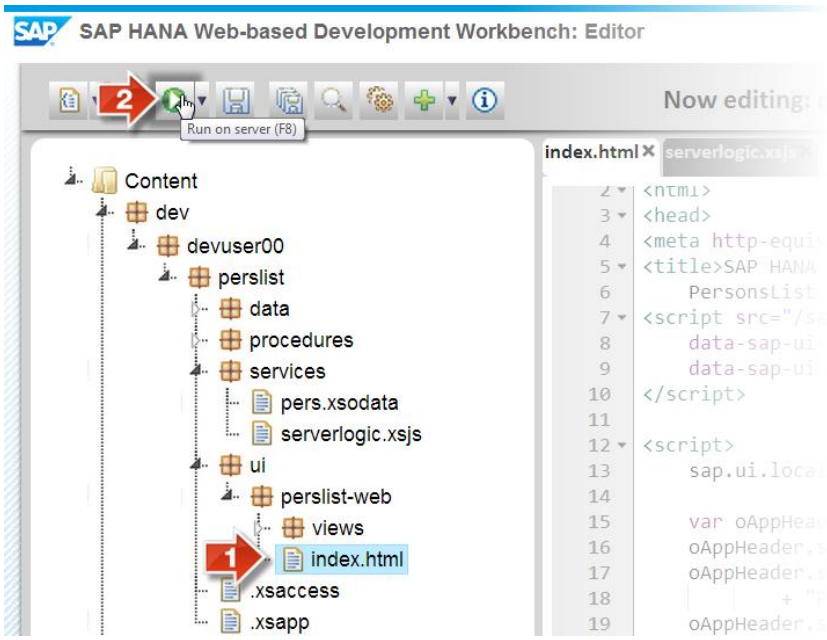
Hands-on Tasks

103. In the Editor browser tab select the file **Content → dev →devuserxx → perslist → services → serverlogic.xsjs** to open the corresponding editor tab.

104. Click in front of a code line to set a break point, e.g. before line "`var result = getUsername();`"

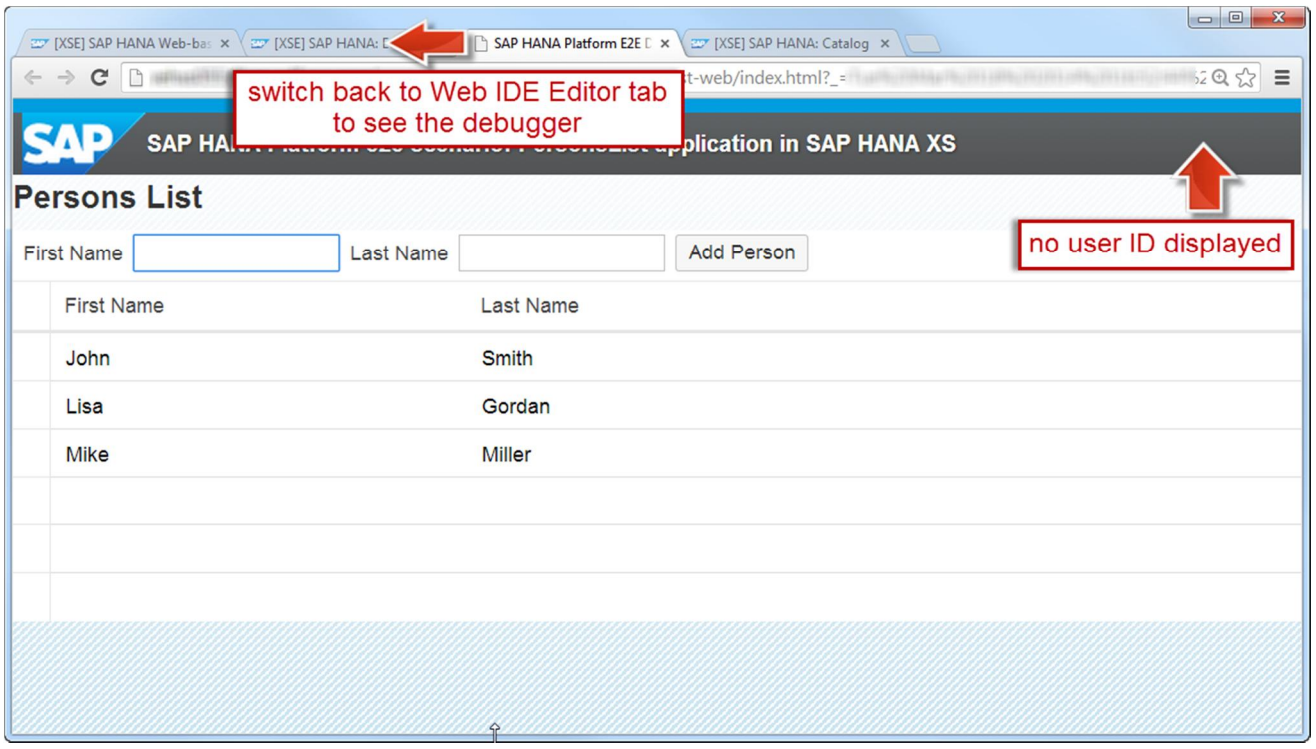


105. Then select the ***index.html*** and run the application by choosing the green run icon from the toolbar.

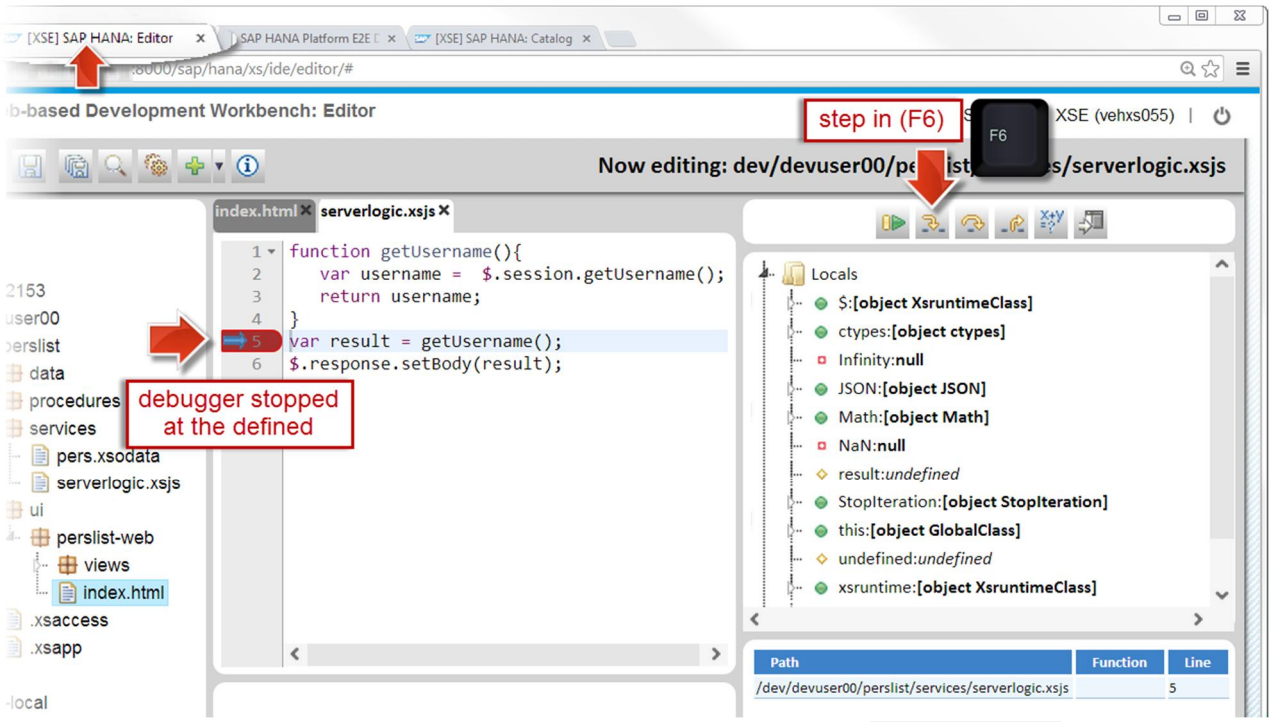


Result

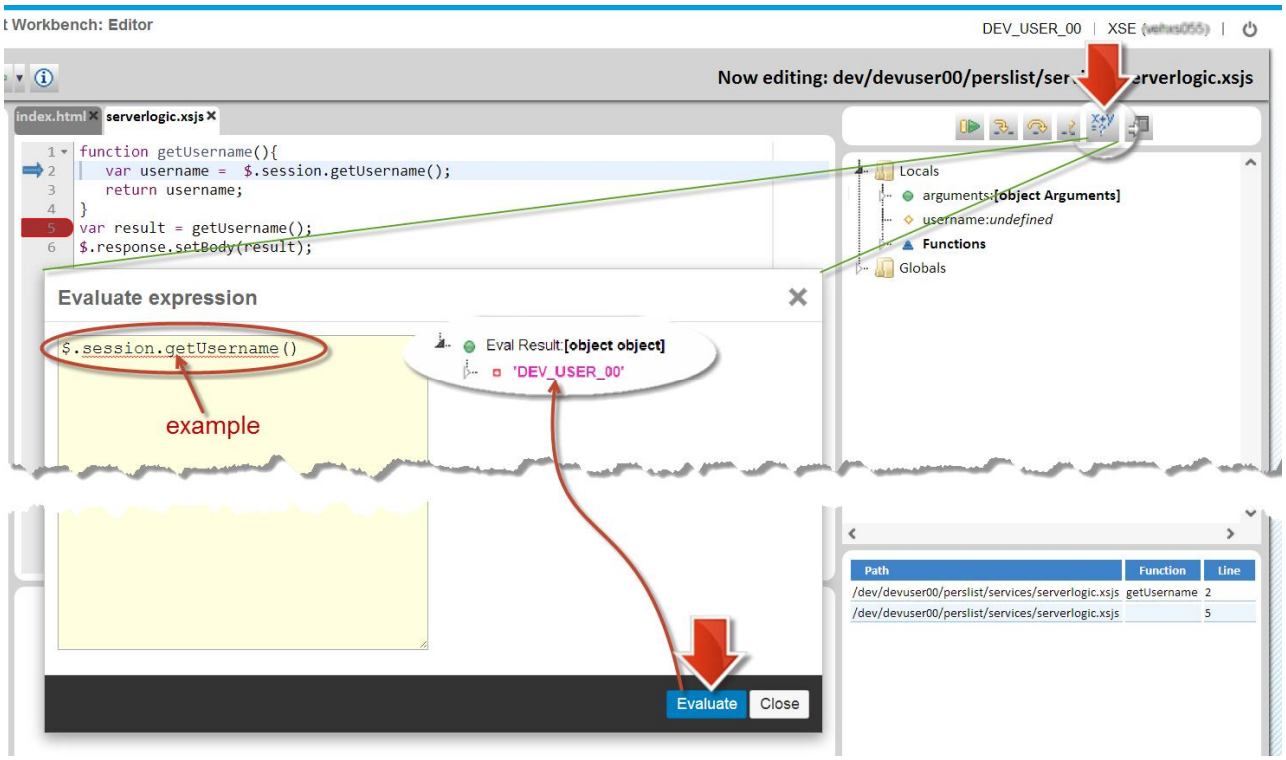
The PersonsList application opens but the user ID is not displayed in the application header because the debugger stopped at the breakpoint you set and therefore the application is waiting for that part.



106. Switch back to the HANA Web IDE Editor to see the XSJS debugger stopping at the before defined breakpoint.

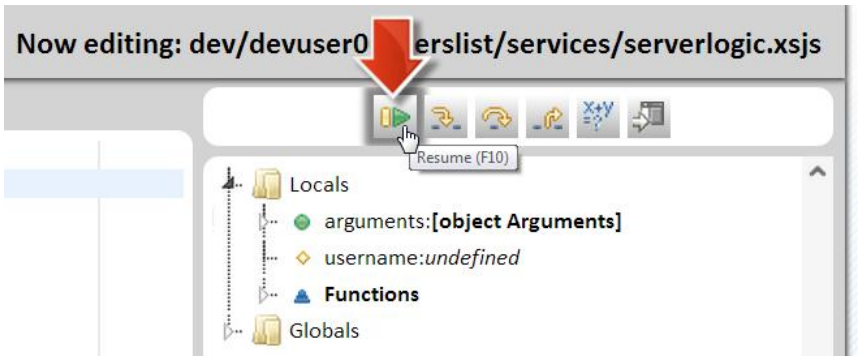


107. Use the ***Step in*** **(F6)** toolbar icon of the XSJS debugger to step into the `getUsername()` function.

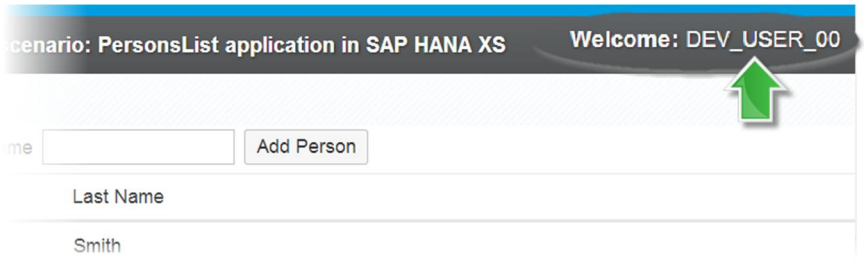


108. Inside the `getUsername()` method you can now e.g. call the **Evaluate** expression window by clicking the corresponding toolbar icon as shown in the above screenshot. In the popup window you can enter an expression to be evaluated (e.g. example the `$.session.getUsername()`). When you click **Evaluate** the result is displayed as shown above.

109. Close the “Evaluate expression” window and click **Resume (F10)** to finish the debugger session.



110. Switch to the PersonsList and see that the user ID is correctly 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
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. Example: `DEV_USER_00` Placeholder in tutorial (e.g. for package name): `devuserxx`
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 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.
UI development toolkit for HTML5	SAPUI5	SAP's enterprise-ready HTML5 rendering library for client-side UI rendering and programming

Read more:

SAP HANA Platform help documentation → Developer Guide:

- 1.3.1 Developing SAP HANA native Applications
- 3.8 Design-Time Application Artifacts

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 Platform using the SAP HANA Web-based Development Workbench*.

5.1 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 > 3 Setting up Your Application > 3.9 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

5.2 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