End-to-End How-to Guide: Building SAPUI5 Applications on SAP NetWeaver AS ABAP 7.31 Consuming Gateway OData Services

Applies to:

SAP NetWeaver AS ABAP 7.03/7.31 SP4 (comprised in EhP6 for SAP ERP 6.0, SAP Business Suite 7 Innovations 2011), UI development toolkit for HTML5 1.6.4 (comprised in UI add-on 1.0 SPS01 for SAP NetWeaver 7.03/7.31), SAP NetWeaver Gateway 2.0 SP04, ABAP Development Tools for SAP NetWeaver, Eclipse 'Classic' IDE package (or higher versions 3.6.x 'Helios', 3.7.x 'Indigo', 4.2.x 'Juno')

Note: You can also run this end-to-end tutorial on a lower EhP2 for SAP NetWeaver 7.0 release (comprised in EhP5 for SAP ERP 6.0, SAP Business Suite 7 Innovations 2010) combined with SAP NetWeaver Gateway 2.0 and the UI add-on 1.0 SPS1 for SAP NetWeaver 7.02 with the limitation, however, that the SAPUI5 application resources cannot be stored in the ABAP backend using the new SAPUI5 ABAP Team Provider (requires SAP NetWeaver 7.03/7.31 SP4). In this case the developed SAPUI5 sample application is tested locally based on the SAPUI5 runtime libs provided by the SAPUI5 development environment. As a simple solution for the missing SAP ABAP Team Provider in SAP NetWeaver 7.00 to 7.31 SP03, SAP will offer an up-/download-report to connect the local SAPUI5 project from Eclipse to the SAPUI5 ABAP Repository (see SAP Note 1793771 for more details).

Summary

This document describes the end-to-end process required to develop a SAPUI5/HTML5 application user interface (with table, popup and ux3 shell) consuming a local Gateway OData service. The application runs on SAP NetWeaver AS ABAP 7.31 SP4 with two add-ons installed. Firstly, the SAP NetWeaver Gateway 2.0 SP4 add-on to build an OData service in the application backend using the Gateway Service Builder. The created OData service is based on the SAP NetWeaver Enterprise Sales and Procurement Model as a pre-installed demo/proxy application that provides a list of products as business data. Secondly, our end-to-end scenario requires the UI add-on 1.0 for SAP NetWeaver 7.31 to build a SAPUI5 application frontend with Eclipse-based SAPUI5 tools and to deploy it on the ABAP server using the SAPUI5 ABAP Team Provider. The described end-to-end process starts in a development system landscape, where we assume that all the necessary components will be installed on a single system.

Prerequisite: the reader should be familiar in general with SAP GUI handling and especially with the SAP Customizing Implementation Guide (transaction SPRO) and with the ABAP Workbench (transaction SE80).

Authors: Bertram Ganz, Bernhard Siewert

Company: SAP AG

Created on: 29 November 2012

Acknowledgement

Many thanks to development architect Thorsten Erlewein, SAP AG, for his excellent support, valuable feedback and helpful suggestions included in this article.

Author Bio

Bertram Ganz works in SAP NetWeaver Product Management in Walldorf, Germany. He joined SAP in 2001, and has focused on product management and knowledge transfer for various SAP UI technology offerings like UI add-on for SAP NetWeaver, UI development toolkit for HTML5 (aka SAPUI5), Web Dynpro ABAP, Floorplan Manager, Web Dynpro Java and SAP NetWeaver Business Client. Bertram is the co-author of the books "Maximizing Web Dynpro for Java" and "Java Programming with the Web Application Server". 2005-2008 he was recognized as a Top Contributor for the SAP Community Network in the Web Dynpro Java category.

Bernhard Siewert works as a development architect in the SAP Product Architecture for Consumption and Collaboration team. He joined SAP in 1992 and spent several years as a developer and consultant for SAP NetWeaver. Since 2001 he has been an architect for the SAP NetWeaver UI team in various SAP internal and customer projects. In 2010 he joined the Product Architecture team, where he focuses on User Interface Interoperability topics.

Table of Contents

Table of Source Codes

Table of Figures

Overview

The purpose of this end-to-end example is to show many of the capabilities of SAP NetWeaver Gateway and the UI development toolkit for HTML (SAPUI5). The used SAP NetWeaver platform is based on AS ABAP 7.31 SP4 with two installed add-ons: UI add-on for SAP NetWeaver 1.0 to use SAPUI5 and SAP NetWeaver Gateway 2.0 SP4 to build an OData service.

The example uses the following features:

- SAP NetWeaver Gateway add-on with Service Builder for OData service consumption in local ABAP system
- UI add-on for SAP NetWeaver with SAP's new UI development toolkit for HTML5 to quickly build a lightweight and stateless business UI running in a browser client
- SAP NetWeaver Sales and Enterprise Procurement Model as a demo/proxy application providing business data (list of products)
- SAPUI5 application Development Tools integrated in an Eclipse IDE
- SAPUI5 ABAP Team Provider to deploy a SAPUI5 application project on the ABAP server
- SAPUI5 control libraries *sap.ui.commons* and *sap.ui.ux3* to build the MVC-based application UI with a ux3 shell and a pageable/sortable table control
- SAPUI5 *Dialog* control to display product details for a selected product (table event handling)
- SAPUI5 core library *sap.ui.model.odata* to easily consume a local Gateway OData service providing a list of products (OData entity set)
- *SimpleProxyServlet* (provided by the SAPUI5 core JS framework) to comply with the same-origin-policy in a local dev/test scenario

End-to-End How-To Guide Steps

We recommend that you follow these end-to-end tutorial sections in the specified order:

Step		What You will learn
Step 1	**Installing SAP NetWeaver Gateway on SAP NetWeaver AS ABAP** Shows how to install and set up SAP NetWeaver Gateway locally on your AS ABAP application backend	• How to install the NetWeaver Gateway as add-on to your SAP NetWeaver AS ABAP backend • How to activate and configure SAP NetWeaver Gateway • How to test a delivered demo OData service
Step 2	**Creating a local Gateway OData Service** Shows how to create, implement and test an OData service for the EPM entity 'Product' using the Gateway Service Builder	• How to use Gateway Service Builder to centrally display and create the definition of a simple OData service based on an ABAP Dictionary structure provided by the SAP NetWeaver ESPM proxy application • How to implement a data provider with paging and sorting logic
Step 3	**Setting up Your SAPUI5 Development Environment on SAP NetWeaver AS ABAP** Outlines the installation of the UI add-on for SAP NetWeaver AS with the SAPUI5 Runtime Library and the installation of SAPUI5 development tools in an Eclipse IDE	• Where to find detailed documentation and SAP Notes on how to install the UI add-on for SAP NetWeaver 7.31 in your AS ABAP 7.31 backend system • Where to find Installation Guide and SAP note for setting up the SAPUI5 development tools in your Eclipse design time together with the ABAP developer tools for SAP NetWeaver
Step 4	**Building & Running the SAPUI5 application UI consuming a Gateway Service** Shows how to build and deploy the SAPUI5 application to the SAP NetWeaver AS ABAP	• How to building and deploy the SAPUI5 application • The concepts basics of the SAPUI5 application architecture and programming model • How to use the SAPUI5 ABAP Team Provider to share application resources on the ABAP server • How to consume a Gateway Service in SAPUI5 • How to use the *SimpleProxyServlet* for local testing

End-to-End Sample Application Architecture

Development scenario and architecture of the entire sample application developed in this document is illustrated in Figure 1. The tutorial focuses on UI and service definition using the corresponding IDE tools.

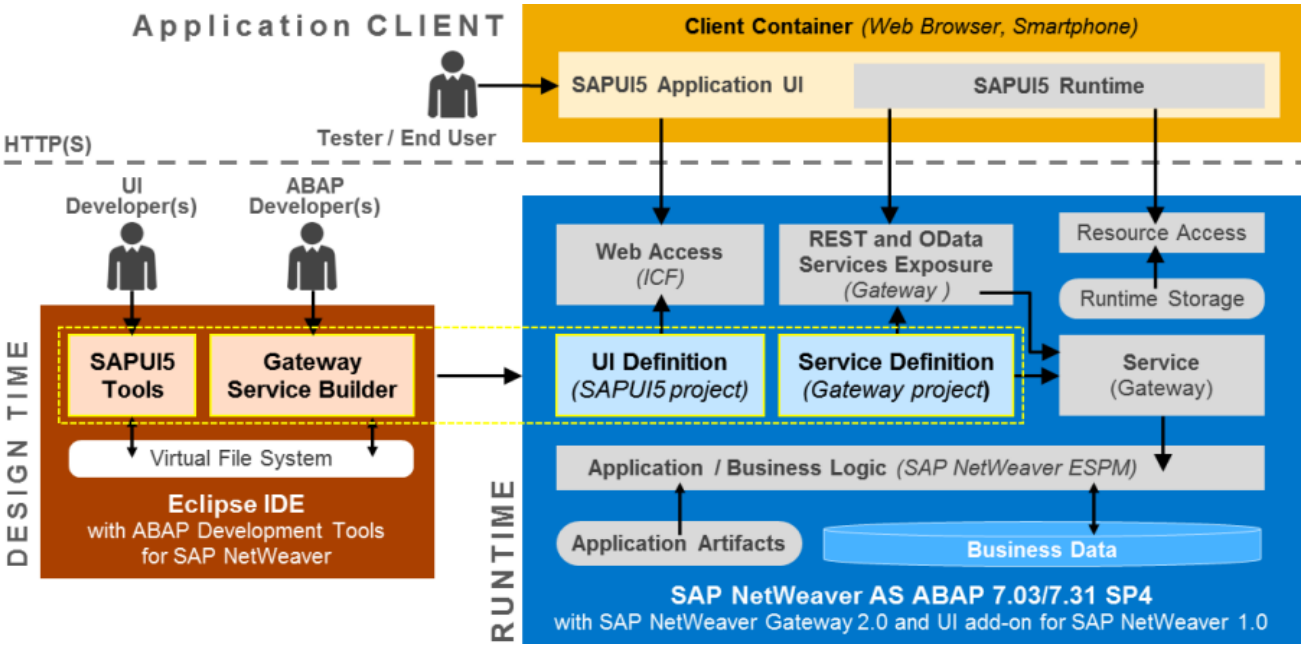
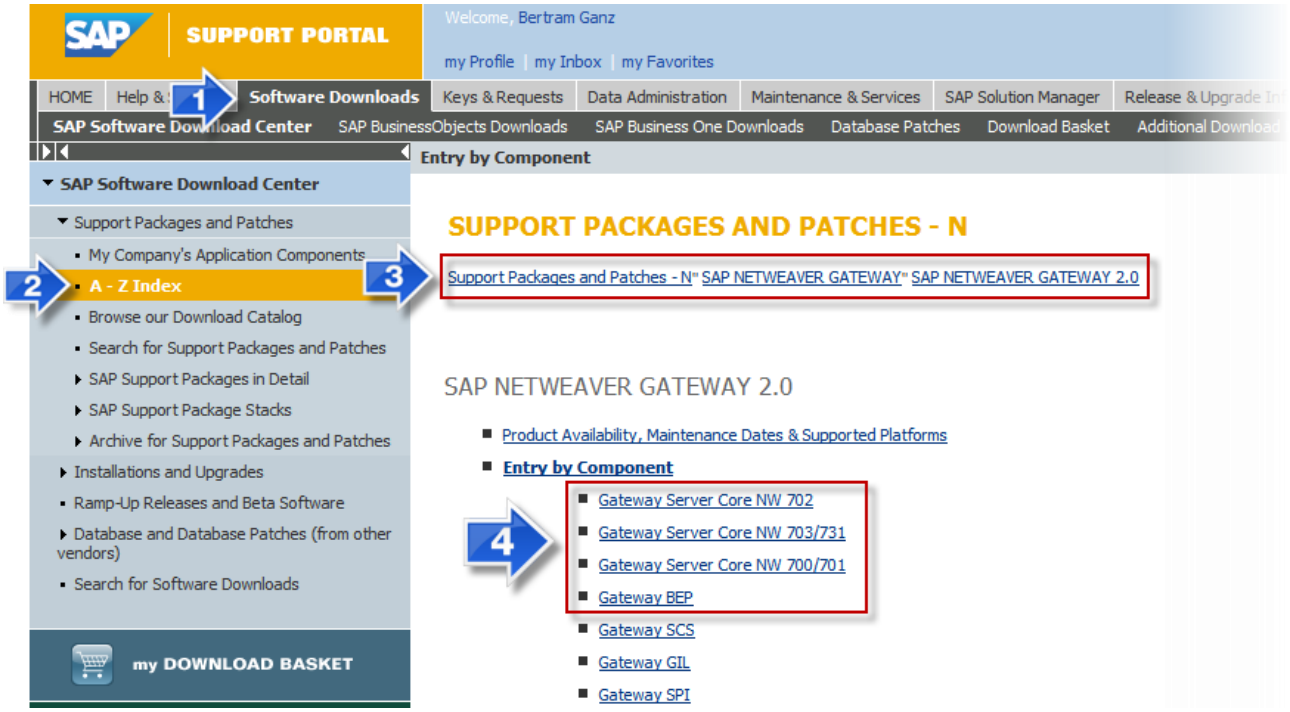


Figure 1: Development scenario and architecture of the SAPUI5 sample application developed in this end-to-end tutorial

Building the Application Backend with SAP NetWeaver Gateway

Installing SAP NetWeaver Gateway on SAP NetWeaver AS ABAP

If there is an application backend development system with SAP NetWeaver release 7.02 or 7.31 all the NetWeaver Gateway components can be installed on this system. These components can be downloaded from the SAP Service Marketplace http://service.sap.com/swdc (see Screenshot 1) and have to be installed with transaction *SAINT* (documentation).



Screenshot 1: SAP NetWeaver Gateway 2.0 on SAP Service Marketplace

What is SAP NetWeaver Gateway?

SAP NetWeaver Gateway is a technology that provides a simple way to connect devices, environments and platforms to SAP software based on market standards. The framework enables development of innovative people centric solutions bringing the power of SAP business software into new experiences such as social and collaboration environments, mobile and tablet devices and rich internet applications. It offers connectivity to SAP applications using any programming language or model without the need for SAP knowledge by leveraging REST services and OData/ATOM protocols.

- SAP Online Help: http://help.sap.com/nwgateway
- SCN Space: http://scn.sap.com/community/netweaver-gateway
- YouTube Channel: http://www.youtube.com/user/sapgateway
- Article: SAP Technical Brief – SAP NetWeaver Gateway

The following Gateway components are required for an OData service enablement of applications (OData Channel):

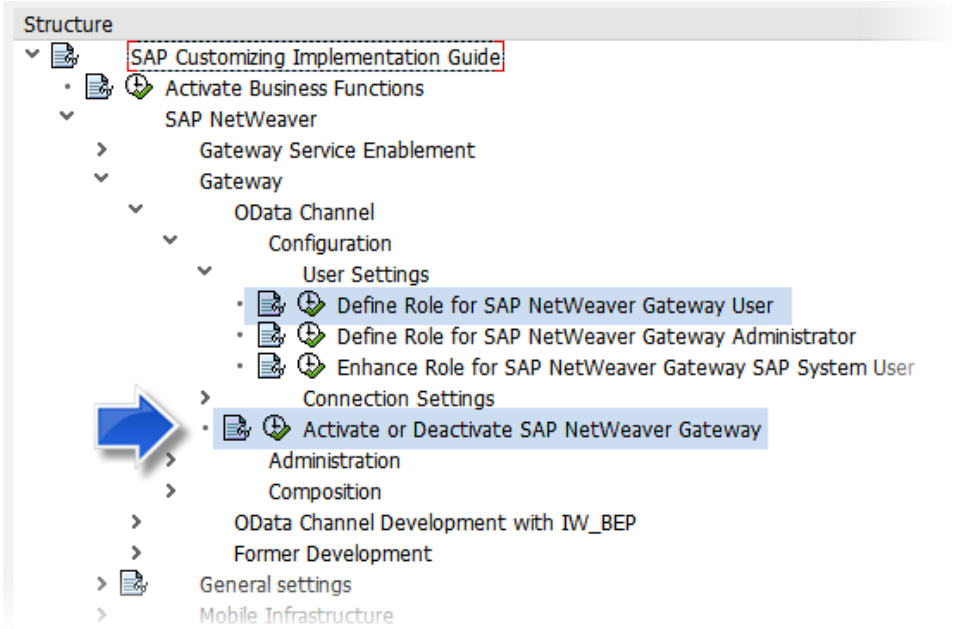
- *GW_CORE 200*,
- *IW_FND 250* (requires an installation of a corresponding WEBCUIF)
- *IW_BEP 200*

Further information on the SAP NetWeaver Gateway installations, its prerequisites, and information on the Add-On Installation Tool (SAINT) can be found in the SAP Help Portal (SAP NetWeaver Gateway - Installation Guide).

Note: If you would like to use for the application frontend development the UI Development Toolkit (aka SAPUI5) including the SAPUI5 ABAP Repository Team Provider to store and synchronize the UI artifacts in the application backend, you have to select the component Gateway Server Core NW 7.03/7.31. This tool is currently not available on SAP NetWeaver 7.02.

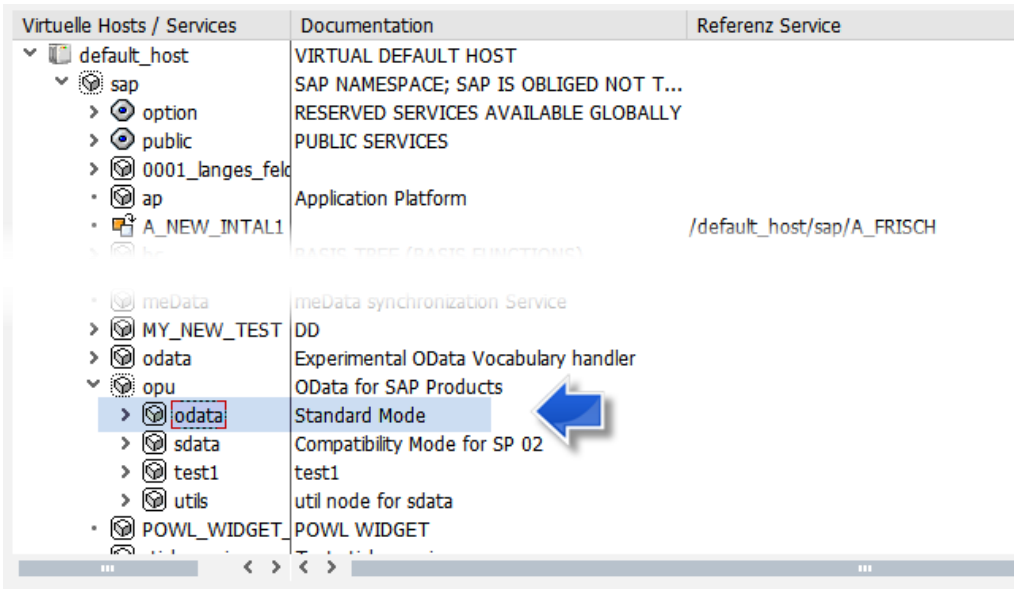
Activation and Configuration of SAP NetWeaver Gateway

The configuration of SAP NetWeaver Gateway can be found within the SAP Customizing Implementation Guide (IMG) – transaction *SPRO*. At first the user, who configure Gateway, has to get administration authorizations – SAP delivers the role templates */IWFND/RT_ADMIN* and */IWBEP/RT_MGW_ADMIN*. These templates should be used for the definition of the Gateway administrator role, which has to be assigned to the user (SAP Online Help on creating and assigning roles). Afterwards this user can activate the NetWeaver Gateway in the IMG:



Screenshot 2: Configuring SAP NetWeaver Gateway within the SAP Customizing Implementation Guide (transaction SPRO)

If Gateway itself has been activated the existing ICF node (Internet Communication Framework) for Gateway OData services has to be activated in transaction SICF (documentation).



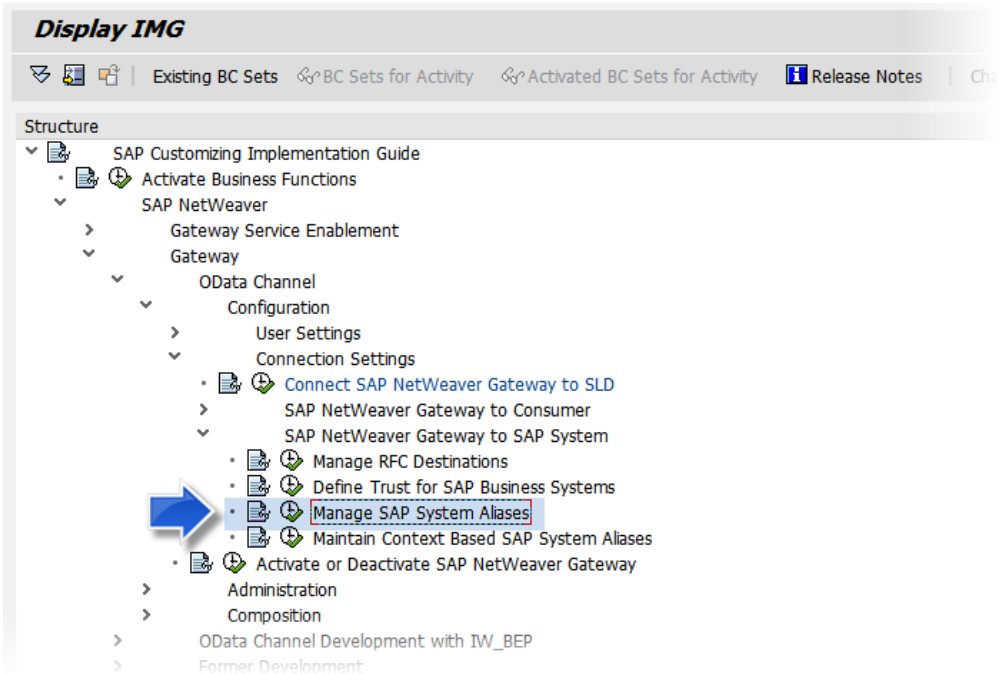
Screenshot 3: Activating ICF node 'odata' for Gateway OData services

Since Gateway is able to call service implementations in different systems all services have to be assigned to a System Alias (based on an RFC destination). To be consistent a system alias has to be created for local service calls with RFC destination *NONE* (Screenshot 4 and Screenshot 5).

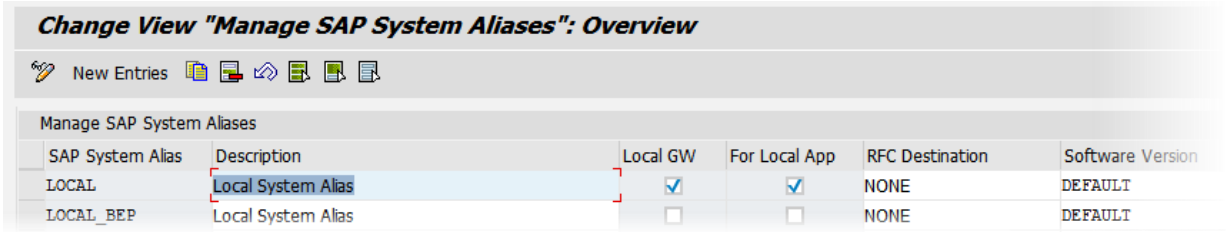
What is the Open Data Protocol (OData)?

OData is a Microsoft developed extension to the Atom Publishing and Atom Syndication standards, which in turn, are based on XML and HTTP(S). It was designed to provide a standardized implementation of a RESTful API. In doing so, it offers database-like access to server-side resources. Hence, OData has been described as: *"ODBC for the Web"*. It can be used freely without the need for a license or contract. OData is also extensible. This allows SAP to supplement the data types used by OData with extra information from the ABAP Data Dictionary.

- SAP Online Help: SAP NetWeaver Gateway and OData



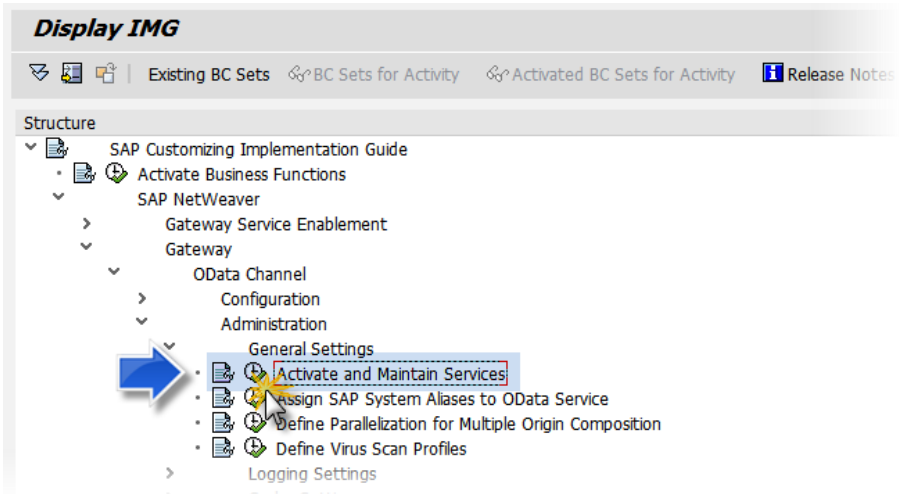
Screenshot 4: Managing SAP system aliases



Screenshot 5: Editing SAP System Alias LOCAL

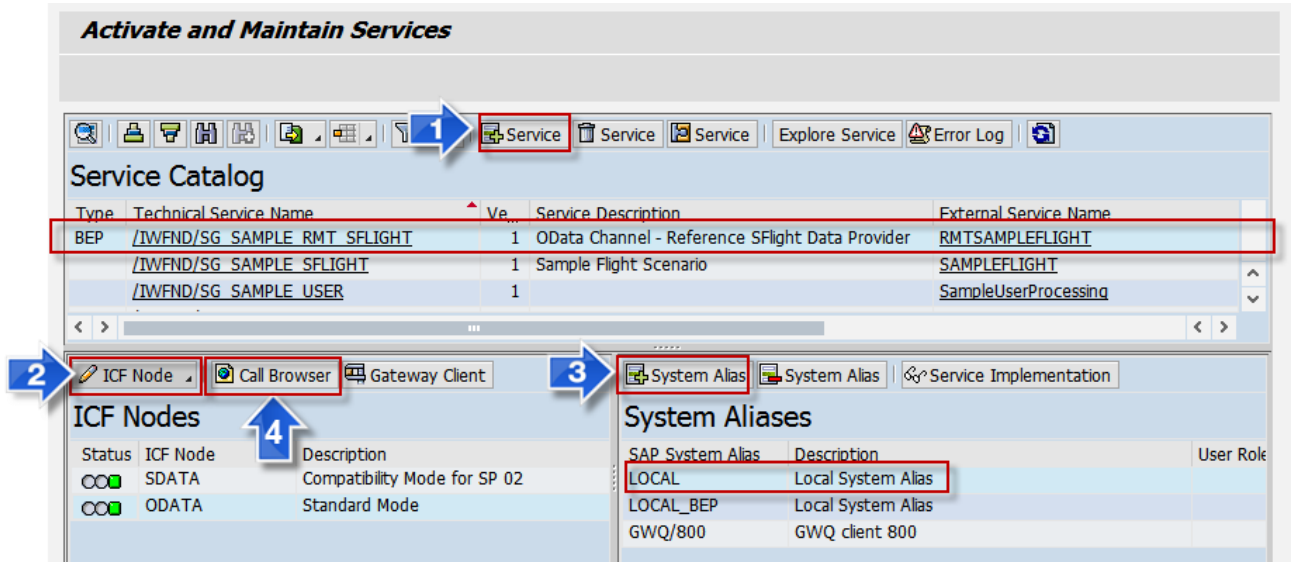
Activating and Testing a delivered Demo OData Service

Having setup the configuration of SAP NetWeaver Gateway an already existing and delivered service (`/IWFND/RMTSAMPLEFLIGHT`) can be activated, and after having the system alias assigned to the service it can be tested right from the SAP Customizing Implementation Guide (transaction `SPRO`).

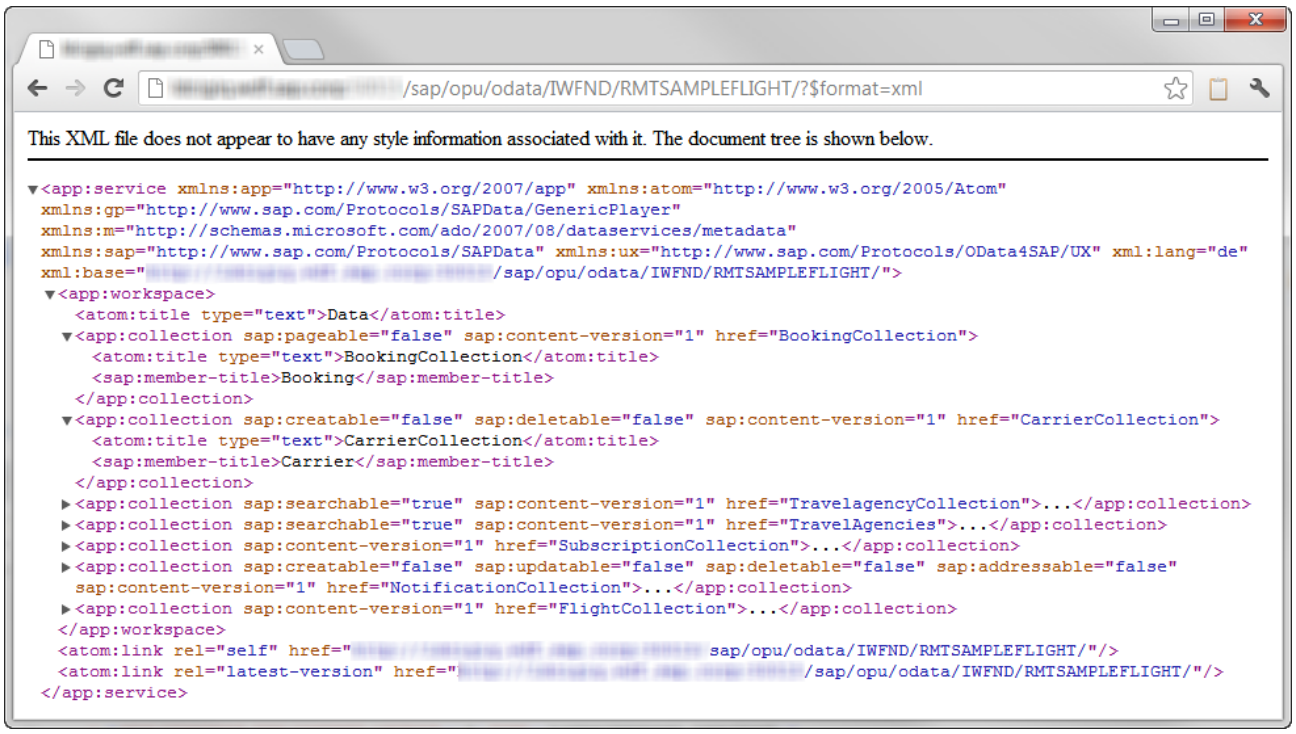


Screenshot 6: Activating and maintaining an OData service in SAP NetWeaver Gateway

Within the Service Catalog you will get a list of all activated Gateway services. If there is already the service with the external service name *RMTSAMPLEFLIGHT* and if the ICF node is activated and a system alias is specified, you can directly test the service with the *Call Browser* button (Screenshot 7, arrow 4, result is shown in Screenshot 8):

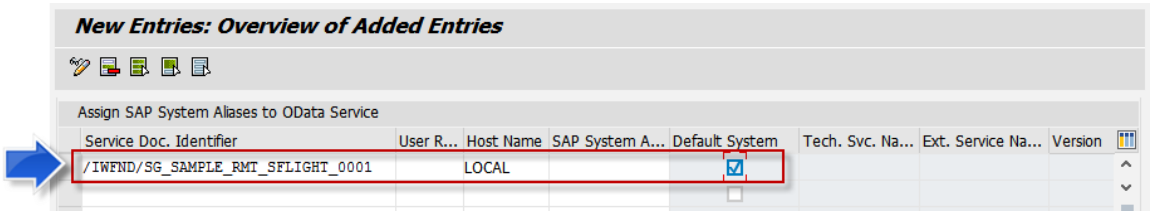


Screenshot 7: Activating and maintaining Gateway Service RMTSAMPLEFLIGHT (external service name)



Screenshot 8: Calling OData service RMTSAMPLEFLIGHT in a Web browser

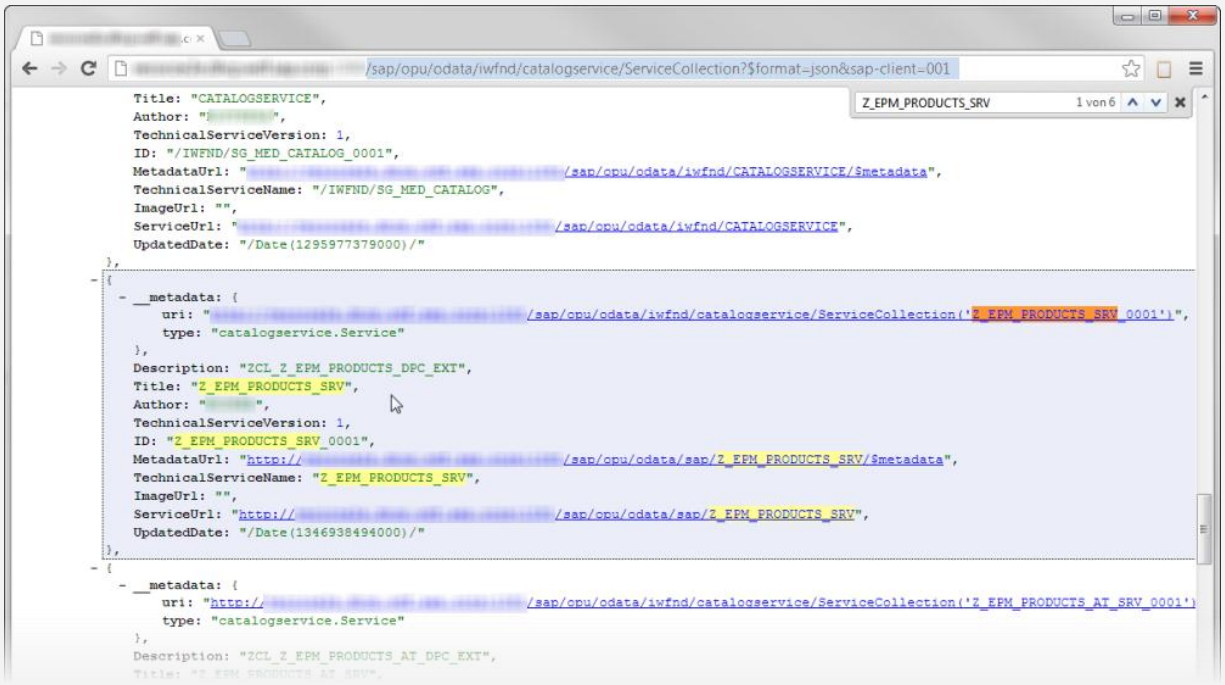
If the system alias is missing, you have to assign the created alias *LOCAL* to this service by choosing button *Create System Alias Assignment* (Screenshot 7, arrow 3):



Screenshot 9: Assigning SAP system alias 'LOCAL' to OData service

If the ICF service is not active it can easily be activated by button *Activate ICF Node* (Screenshot 7, arrow 2), and if the service is not listed in the catalog you have to add it there by choosing button *Add Service* (Screenshot 7, Step 1). There will come up a selection screen, where you can specify e.g. the system alias and you will get a list of all services from the respective system, which are not part of the service catalog, yet. By selecting the service in the list this service can be added to catalog.

Note: There is a catalogue of all OData services as an own OData service available: `/sap/opu/odata/iwfnd/catalogservice/`. This service contains currently one Entity Set called *ServiceCollection*. So by calling the URL `/sap/opu/odata/iwfnd/catalogservice/ServiceCollection?$format=json` a list of all existing and activated OData services will come up. The below screenshots show the OData service Z_EPPRODUCTS we will create in this tutorial.



Implementing and Testing a local OData Service with the SAP NetWeaver Gateway Service Builder

For the implementation and test of OData Services you need a user with Gateway development authorizations. So a role based on the templates */IWFND/RT_DEVELOPER* and */IWBEP/RT_MEG_DEV* has to be created and assigned to the user.

There are different options of creating an OData Service Model:

1. You can define an EDMX model in an external tool like Visual Studio Express (or reuse an existing one) and import it into Gateway to create the service (documentation)
2. Starting with SP4 you can use the SAP NetWeaver Gateway Service Builder (documentation).
3. Besides that you can also directly implement the metadata and data provisioning classes for a service (documentation).

What's the SAP NetWeaver Gateway Service Builder?

SAP NetWeaver Gateway Service Builder (transaction SEGW) is a design-time environment, which provides developers with an easy-to-use set of tools to centrally display and create the definition of OData services. It has been conceived for the code-based OData channel and supports developers throughout the entire development life cycle of a service comprising runtime artifacts (model provider class, data provider class, model and service), OData artifacts (entity set, entity type and properties) and used data sources and models. It is based on an open framework that allows adding own plug ins for special tasks (e.g. creation of Unit Test). The modeling environment follows a project based approach, all relevant data is consolidated in one project. The Service Builder is available as of SAP NetWeaver Gateway Release 2.0 Support Package 04. As of SAP NetWeaver Gateway 2.0 Support Package 05, you are strongly recommended to use the Service Builder (transaction SEGW) for your OData design-time development as opposed to the previous design time integrated in the ABAP Workbench (transaction SE80). SAP strongly recommends using the OData Channel (ODC) programming paradigm for all SAP NetWeaver Gateway development.

- SAP Online Help: SAP NetWeaver Gateway Service Builder
- SCN blog: The new SAP NetWeaver Gateway Service Builder: Build new OData Services in 3 Quick Steps
- YouTube video: SAP NetWeaver Gateway Service Builder, [5:28 min]

Within this article the SAP NetWeaver Gateway Service Builder will be used to create an OData service based on *the SAP NetWeaver ESPM (Enterprise Sales and Procurement Model)* demo model. At first it should be checked if the demo model data are already available by testing function module *BAPI_EPM_PRODUCT_GET_LIST*. If this function module doesn't deliver any data, the EPM Data Generator (transaction `SEPM_DG` and press F8) should be called (button H will display the program documentation).

In SAP NetWeaver Gateway Service Builder you need to do 3 main steps to build an OData service (see Figure 2):

1. Define or import the data model
2. Implement or generate the runtime logic for the service operations
3. Activate and run the service

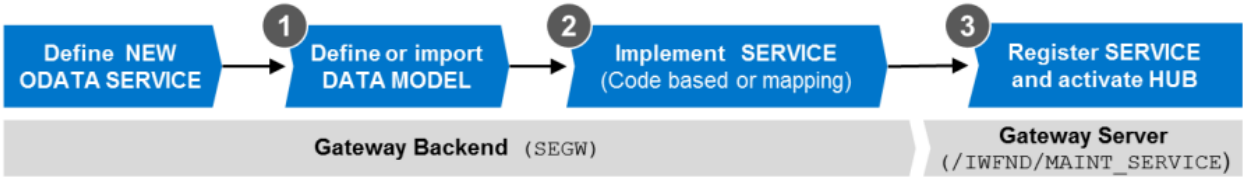


Figure 2: Development flow for OData service definition using the Gateway Service Builder

In SAP NetWeaver Gateway 2.0 SP5 the Service Builder is enhanced with modeling and generation capabilities that enable the user to generate not only the model definition but also the service implementation, so practically the user can create fully functional Gateway service without writing a single line of code. For sake of better understanding the Gateway OData service to be created we will not completely generate it but manually implement the data provider base class with sorting and paging logic. In Figure 3 you see the OData service artifacts we create for the Product entity of the SAP NetWeaver Enterprise Sales and Procurement Model (ESPM, also shortly named EPM).

Note: The other SAP NetWeaver ESPM entities *BusinessPartner*, *SalesOrder* and *SalesOrderItems* are not used in this end-2-end tutorial.

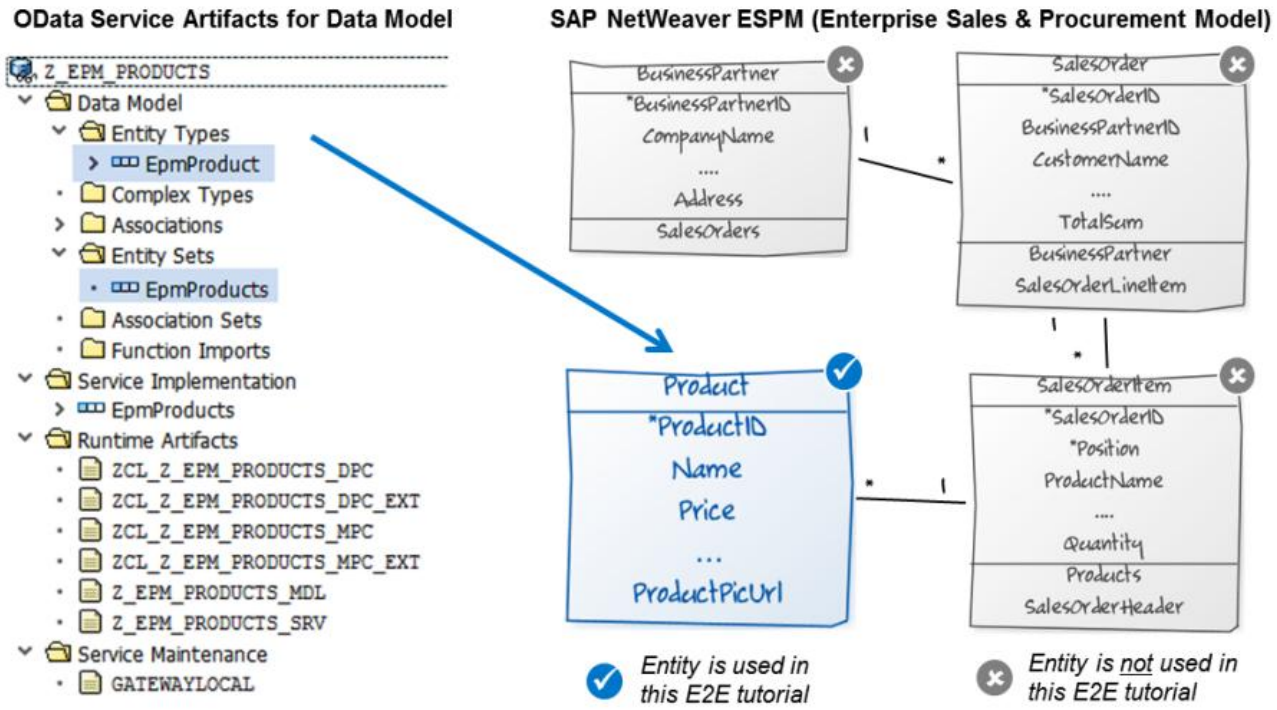
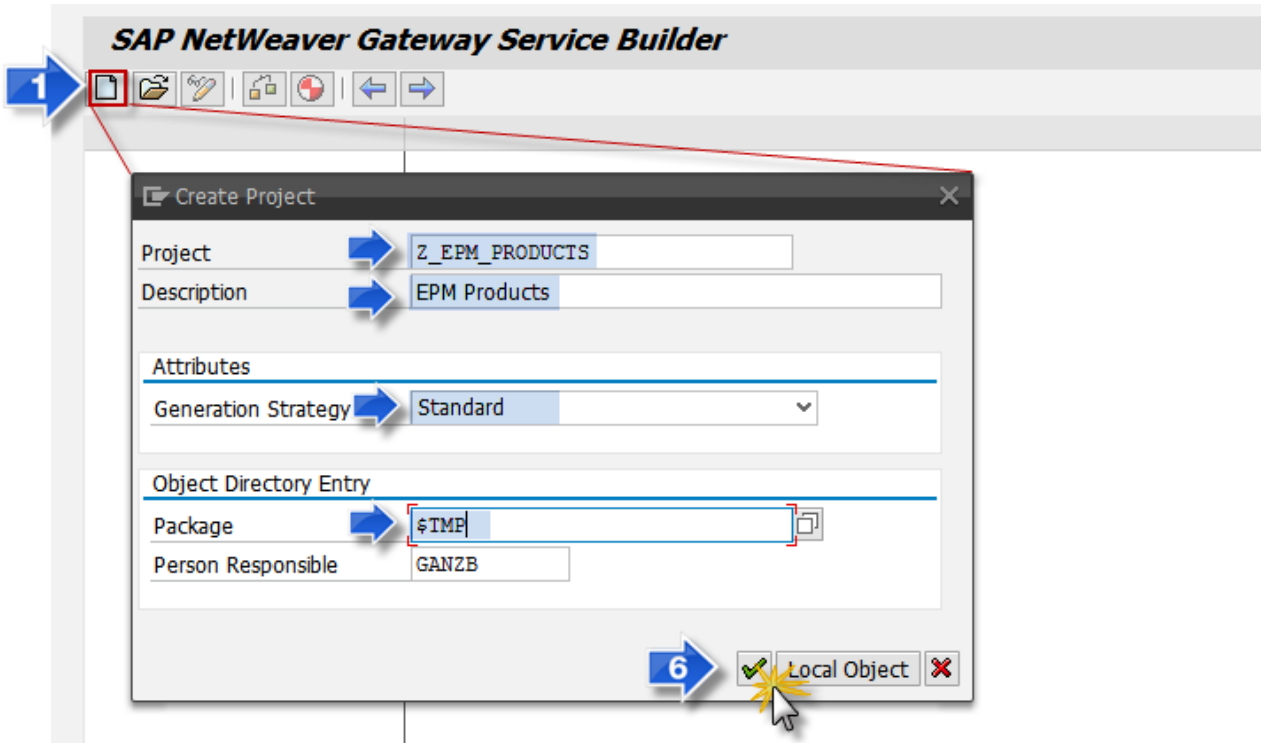


Figure 3: OData service artifacts created for the 'Product' entity of SAP NetWeaver ESPM

The following sections will walk you through the process of generating a Gateway service based on an existing ABAP DDIC structure defined by the SAP NetWeaver ESPM (Enterprise Sales and Procurement Model).

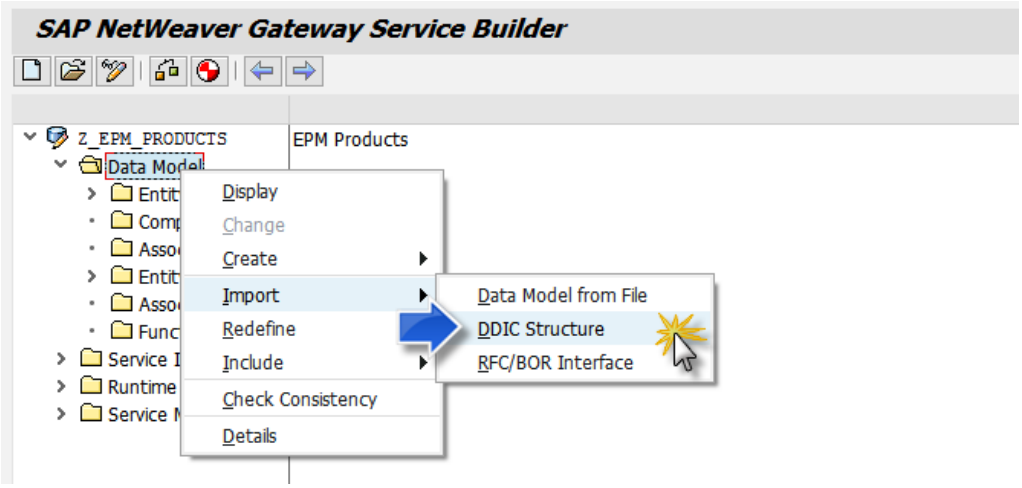
Importing the Data Model from ABAP DDIC structure with SAP NetWeaver Gateway Service Builder

The SAP NetWeaver Gateway Service Builder can be called with transaction `SEGW`. Here at first a new project has to be created:



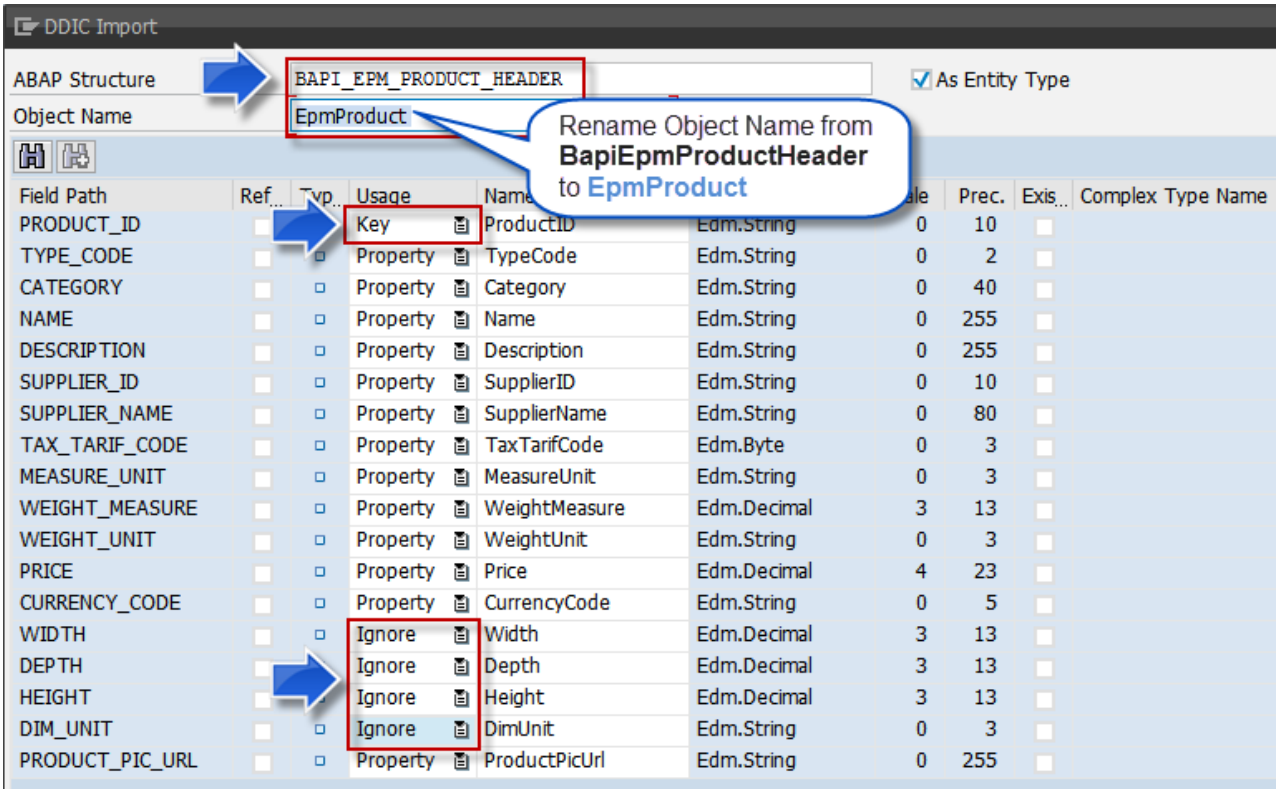
Screenshot 10: Creating a project 'Z_EPM_PRODUCTS' with SAP NetWeaver Gateway Service Builder

The result will be a project structure containing the data model, the service implementation, the runtime objects and the service information. At first the generated project should be saved. Within the service builder the data model can be imported from an external source (an EDMX file) or it can be generated from the Business Object Repository (BOR), RFC or ABAP DDIC information. Of course it can also be built up from scratch by defining all the properties, complex types, entity types and associations from scratch.



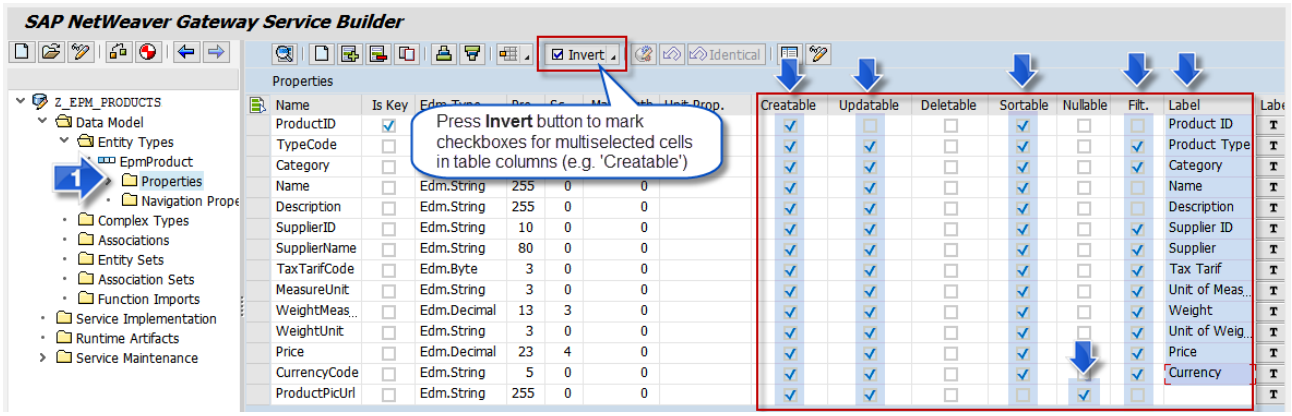
Screenshot 11: Importing a DDIC structure into the data model

Since we would like to see a product catalog in this sample application the application logic will be based on the function module *BAPI_EPM_PRODUCT_GET_LIST*, whose result is a table of the structure *BAPI_EPM_PRODUCT_HEADER*. By importing this DDIC structure into the model (see Screenshot 11) it can be decided, which fields will be key or property and which fields will be ignored. In our sample the Object Name is set to *EpmProduct* (from default name '*BapiEpmProductHeader*',see Screenshot 12).



Screenshot 12: Setting field usage types of imported ABAP Structure BAPI_EPM_PRODUCT_HEADER

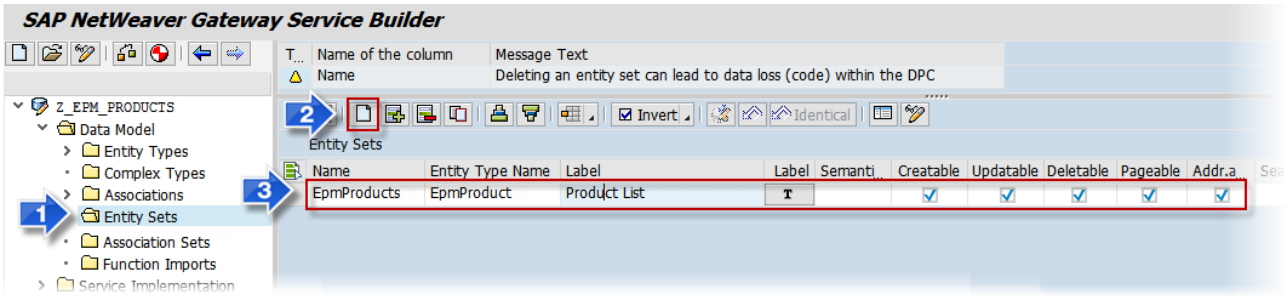
Afterwards the SAP annotations should be maintained – there you can set label texts (as free text or defined by a program or DDIC text), and whether the property can be updated or if the entities can be sorted or filtered by this property:



Screenshot 13: Setting properties of entity type 'EpmProductHeader'

Creating Entity Set 'EpmProducts' for the Entity Type 'EpmProduct'

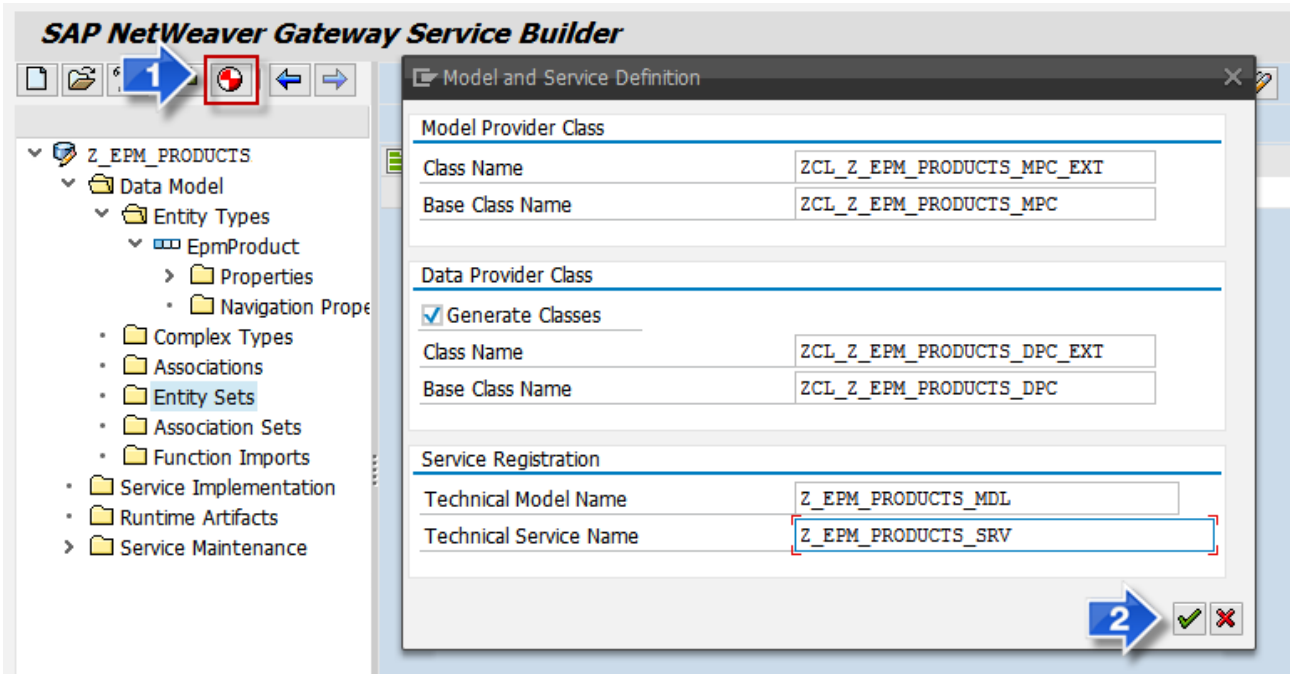
Based on this new entity *EpmProduct* the corresponding entity set *EpmProducts* can be created (by double-click on the left hand side on tree node *Entity Sets* and selecting either the *Append Row* or *Insert Row* button on the right hand side). Here you can decide if instances of the entity type can be created, updated or deleted and if the collection supports e.g. paging (see Screenshot 14).



Screenshot 14: Creating new entity set' EpmProducts'

Generating and Registrating the new Gateway Service

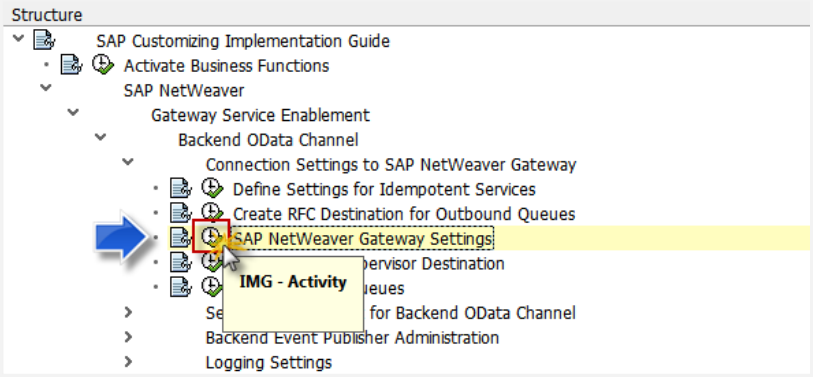
Now the service can already be generated, which means that the model and data provider classes are generated and the service is registered (names are suggested by the system, but can be overwritten):

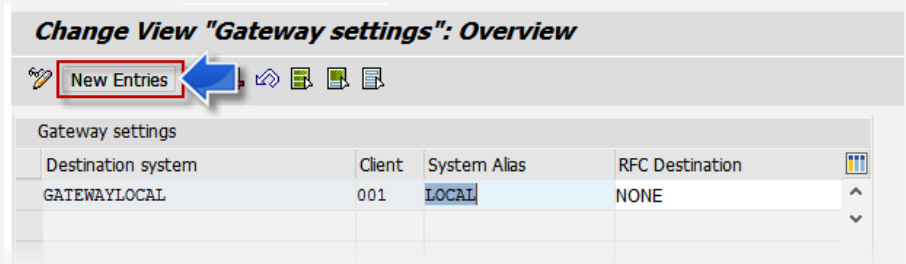


Screenshot 15: Generation of runtime objects for project Z_EPM_PRODUCTS

After generating the runtime artifacts the service can be activated by double-clicking on *GATEWAYSERVER*. catalog.

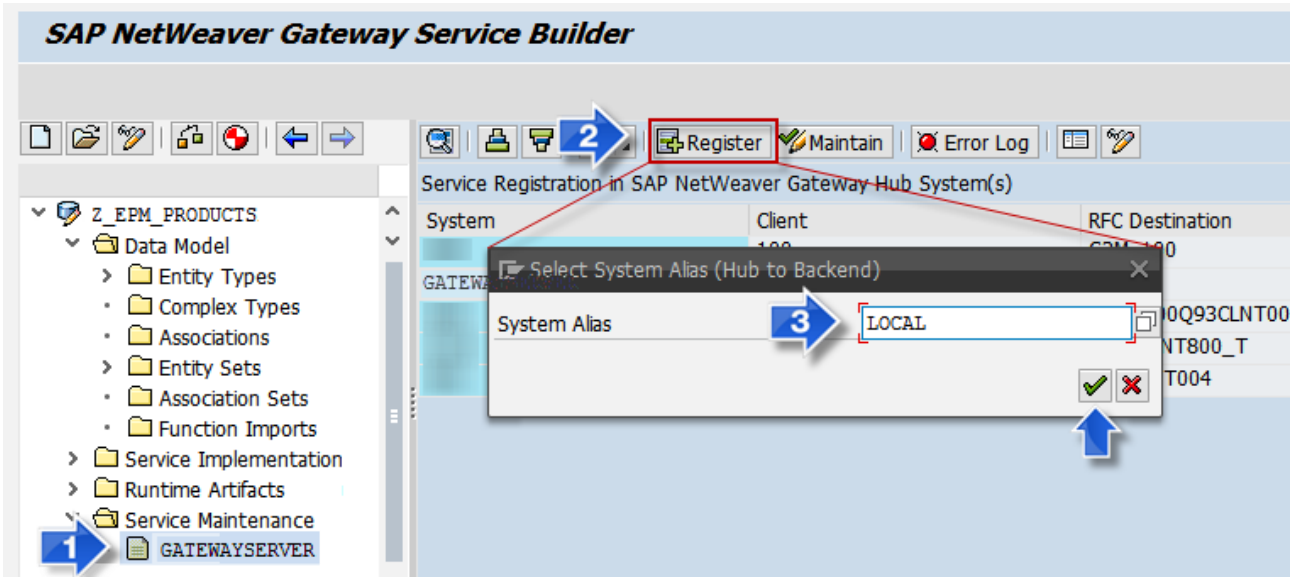
Note: If there is no server configured for *Service Maintenance* the connection to the Gateway Server has to be created. Therefore select *SAP NetWeaver Gateway Settings* in the Implementation Guide (transaction SPRO, see screenshot below), create a new entry with system name (e.g. *GATEWAYSERVER*) and the client & RFC destination of the Gateway server.





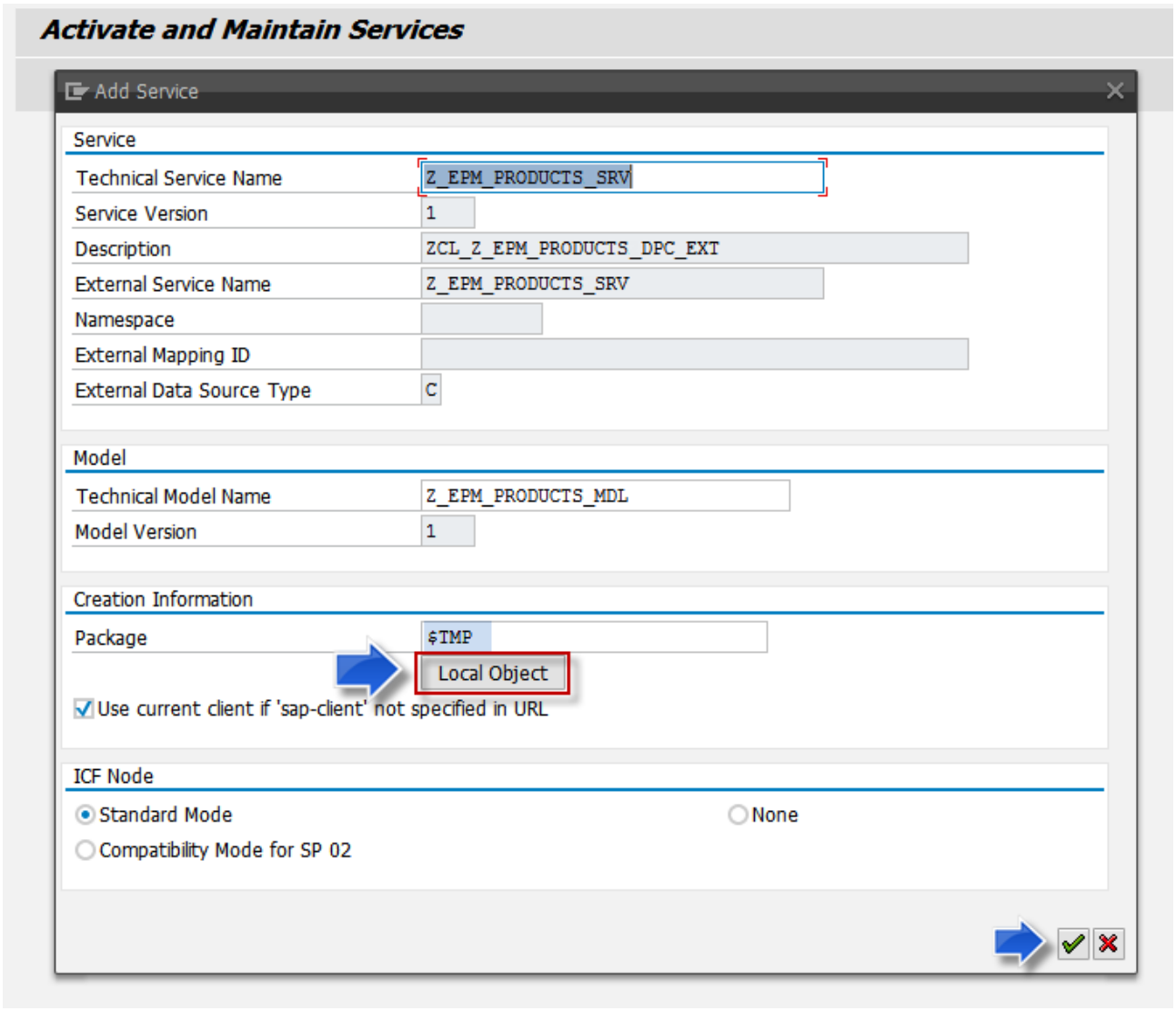
Since in this use case SAP NetWeaver Gateway is embedded in the application backend the RFC destination has to be *NONE*.

Within the *Service Maintenance* screen the *Register Service* button has to be selected for the dedicated Gateway server. Then the warning that all operations of the Service Maintenance will be redirected to the corresponding server (which is here the same, since we have an embedded Gateway) has to be accepted and the system alias (*LOCAL*) has to be specified.

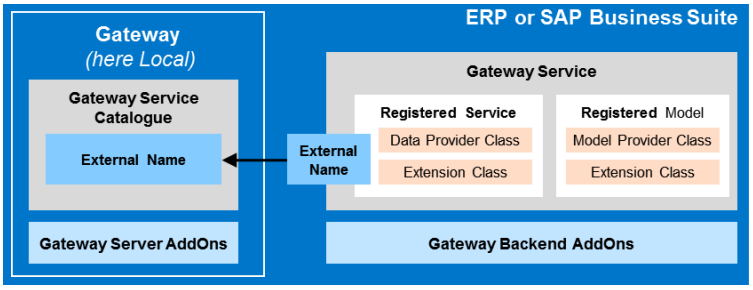


Screenshot 16: Service registration in local system (system alias 'LOCAL')

With the next screen the service will be added to the service catalog and the field *Registration Status* of the *GATEWAYSERVER* should show a green light afterwards.



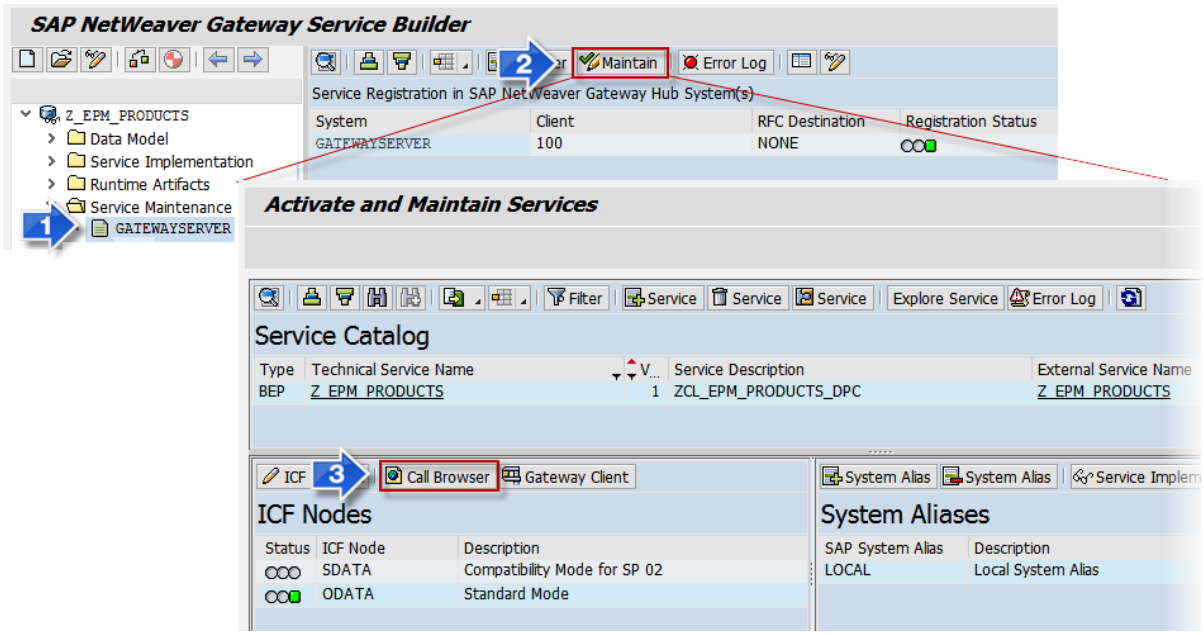
Screenshot 17: Adding new Gateway service



SEGW created 4 classes, and registered one model and one service	
Data Provider Base Class	ZCL_Z_EPM_PRODUCTS_DPC
Data Provider Extension Class	ZCL_Z_EPM_PRODUCTS_DPC_EXT
Model Provider Base Class	ZCL_Z_EPM_PRODUCTS_MPC
Model Provider Extension Class	ZCL_Z_EPM_PRODUCTS_MPC_EXT
Registered Model	ZCL_Z_EPM_PRODUCTS_MDL
Registered Service	ZCL_Z_EPM_PRODUCTS_SRV

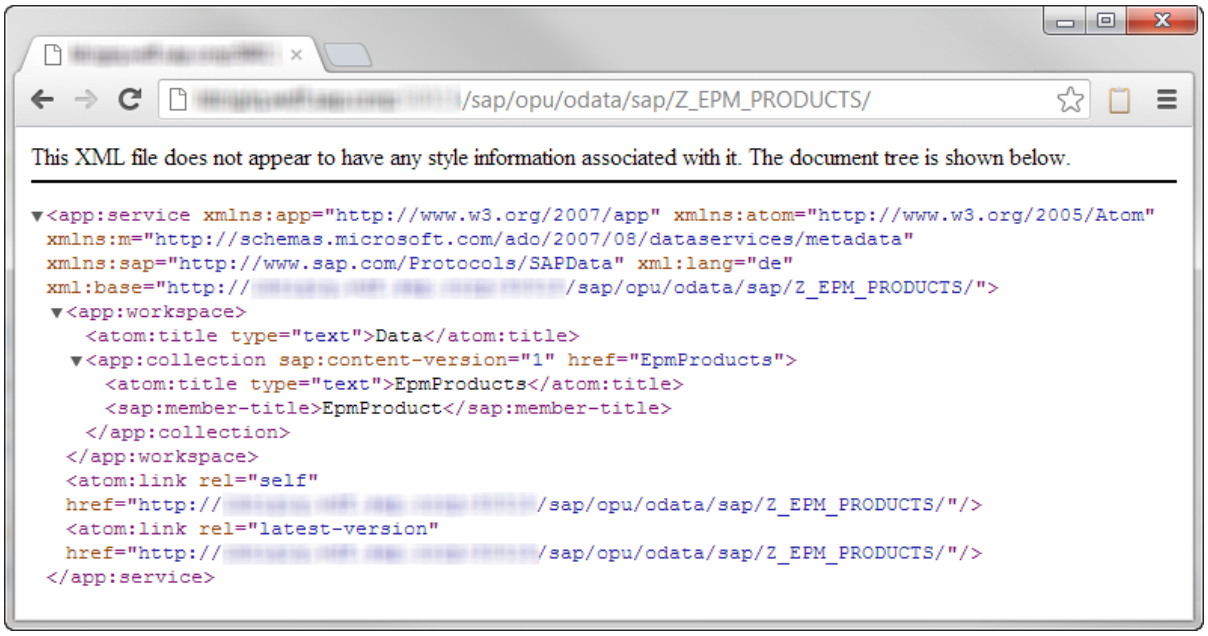
Figure 4: Gateway service artifacts created by Gateway Service Builder

Now the service document (Screenshot 19) and the metadata document (Screenshot 20) should already be accessible by the browser. To call the service document in a browser press the *Maintain* button followed by button '*Call Browser*' in the '*Activate and Maintain*' dialog (Screenshot 18).



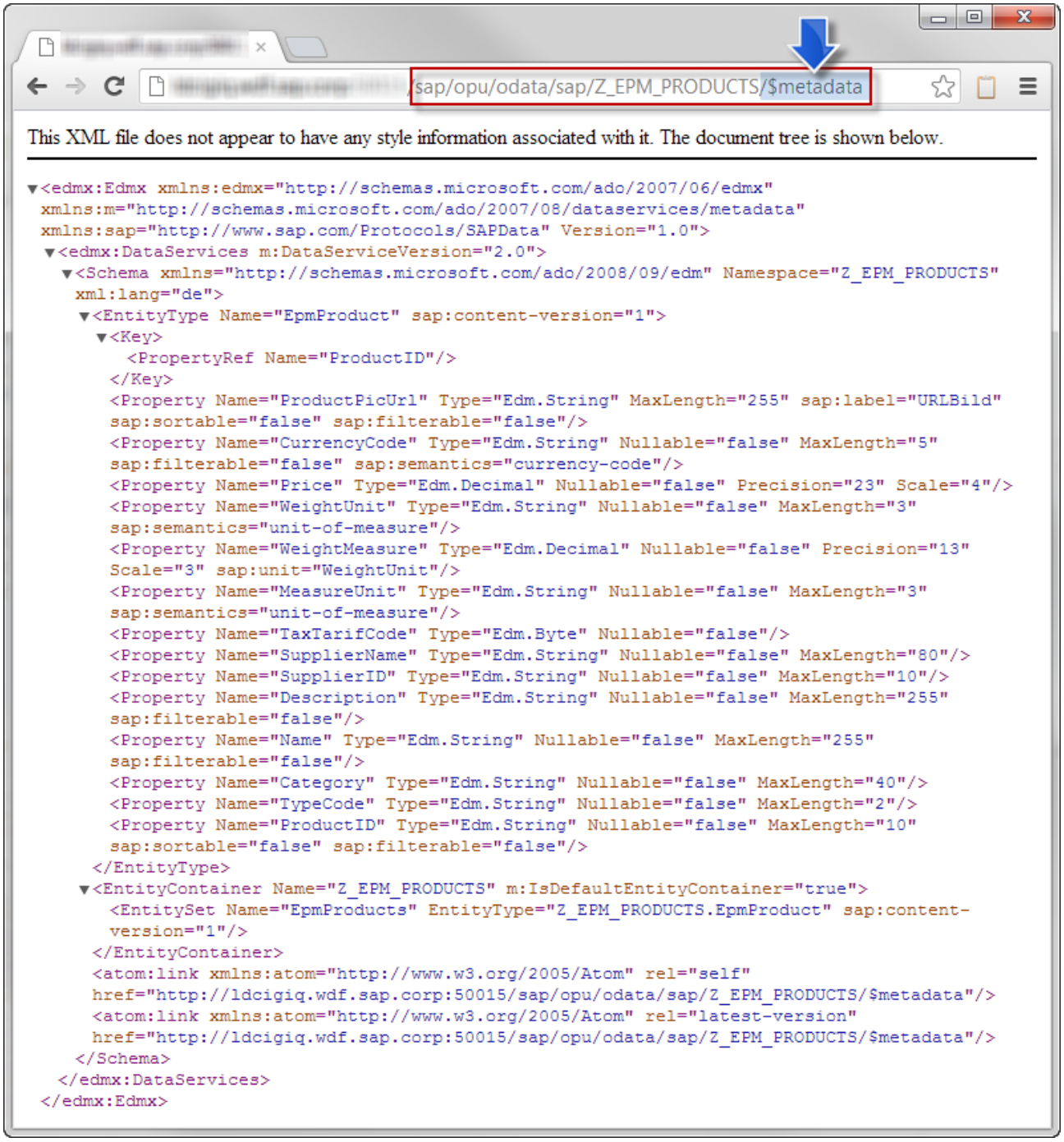
Screenshot 18: Calling the service document in browser window

The ICF path will be */sap/opu/odata/sap/z_epm_products*.



Screenshot 19: Service XML document for Gateway service Z_EPM_PRODUCTS

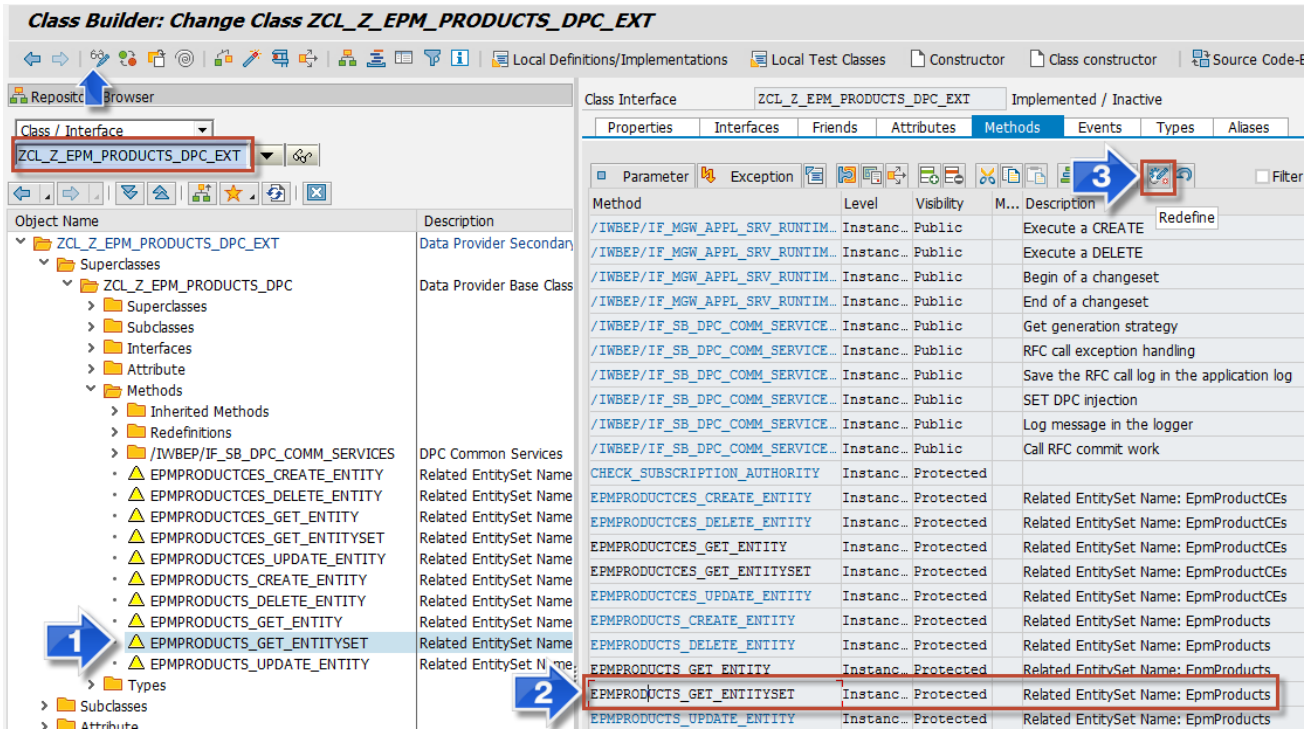
To request the service metadata XML document in your web browser append String '`/$metadata`' to the service document URL (Screenshot 20).



Screenshot 20: Service metadata XML document for Z_EPM_PRODUCTS displayed in web browser

Implementing the Data Provider Class ZCL_Z_EPM_PRODUCTS_DPC_EXT with Paging and Sorting Logic

The next step is the implementation of the data provider class ZCL_Z_EPM_PRODUCTS_DPC_EXT to fetch the EPM product data. Therefor the method *EPMPRODUCTS_GET_ENTITYSET* should be implemented (via redefinition) at first:



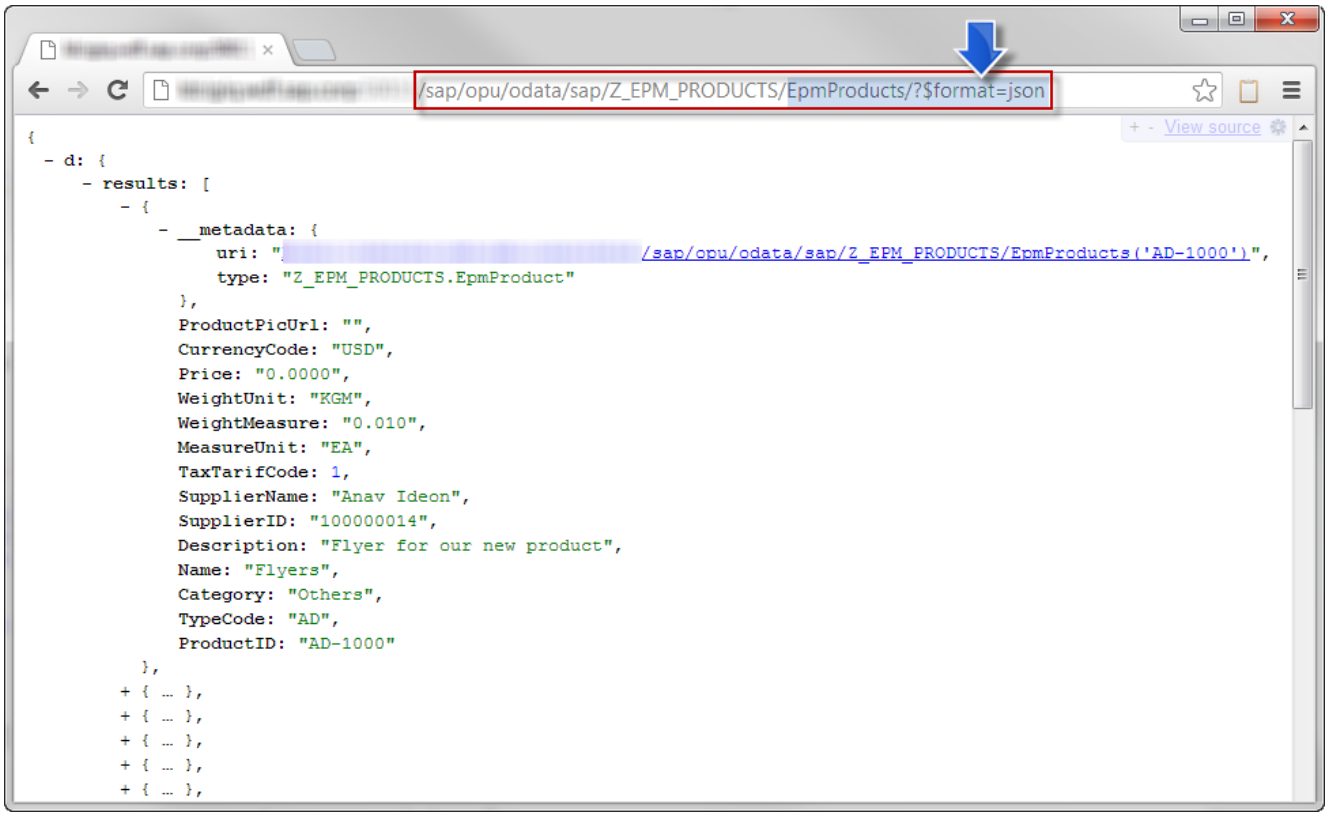
Screenshot 21: Implementing method EPMPRODUCTS_GET_ENTITYSET in class ZCL_Z_EPM_PRODUCTS_DPC_EXT

```
METHOD epmproducts_get_entityset.

  DATA lt_products  TYPE TABLE OF bapi_epm_product_header.
  DATA lt_return    TYPE TABLE OF bapiret2.
  DATA lv_min       TYPE i.
  DATA lv_max       TYPE i.
  DATA lt_techorder TYPE /iwbep/t_mgw_tech_order.
  DATA lt_sortorder TYPE abap_sortorder_tab.
  FIELD-SYMBOLS <lf_order>     TYPE /iwbep/s_mgw_tech_order.
  FIELD-SYMBOLS <lf_sortorder> TYPE abap_sortorder.
  FIELD-SYMBOLS <lf_products>  TYPE bapi_epm_product_header.
  FIELD-SYMBOLS <lf_entityset> TYPE zcl_epm_products_mpc_base=>ts_epmproduct.

* Get technical request information
  IF io_tech_request_context IS BOUND.
    lt_techorder = io_tech_request_context->get_orderby( ).
  ENDIF.
* Get List of Products (to avoid a read with every call a cache could be implemented)
  CALL FUNCTION 'BAPI_EPM_PRODUCT_GET_LIST'
    TABLES
      headerdata = lt_products
      return     = lt_return.
* Sorting
  LOOP AT lt_techorder ASSIGNING <lf_order>.
    APPEND INITIAL LINE TO lt_sortorder ASSIGNING <lf_sortorder>.
    <lf_sortorder>-name = <lf_order>-property.
    IF <lf_order>-order = `desc`.
      <lf_sortorder>-descending = abap_true.
    ENDIF.
    IF <lf_order>-property = `PRODUCT_ID` OR
       <lf_order>-property = `DESCRIPTION` OR
       <lf_order>-property = `NAME` OR
       <lf_order>-property = `CURRENCY_CODE` OR
       <lf_order>-property = `SUPPLIER_NAME`.
      <lf_sortorder>-astext = abap_true.
    ENDIF.
  ENDLOOP.
```

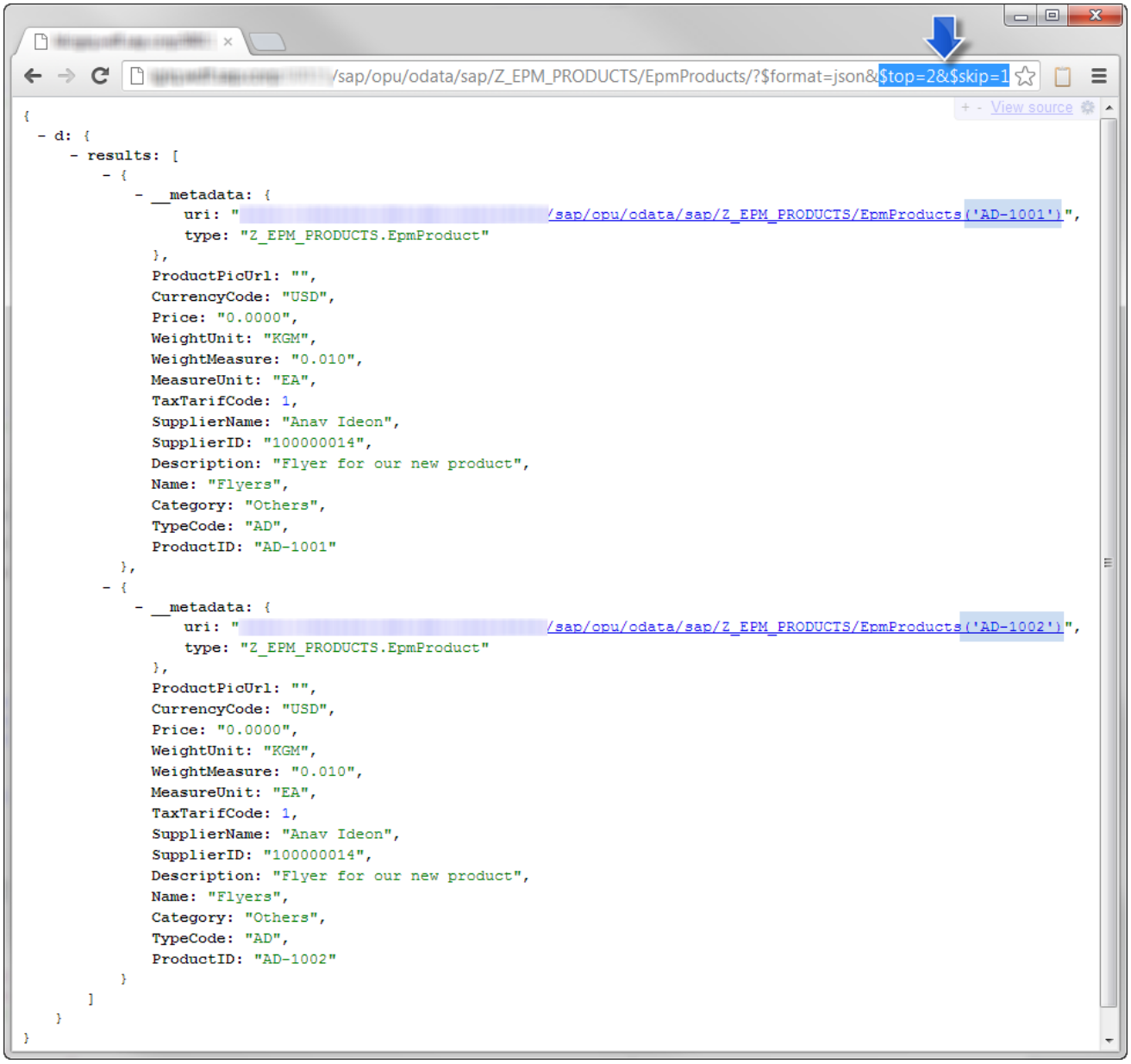
```
  SORT lt_products by (lt_sortorder).
* Paging
  IF is_paging-skip IS NOT INITIAL.
    lv_min = is_paging-skip + 1.
  ELSE.
    lv_min = 1.
  ENDIF.
  IF is_paging-top IS NOT INITIAL.
    lv_max = is_paging-skip + is_paging-top.
  ELSE.
    lv_max = lines( lt_products ).
  ENDIF.
  LOOP AT lt_products FROM lv_min TO lv_max ASSIGNING <lf_products>.
    APPEND INITIAL LINE TO et_entityset ASSIGNING <lf_entityset>.
    MOVE-CORRESPONDING <lf_products> TO <lf_entityset>.
  ENDLOOP.

ENDMETHOD.
```

Source Code 1: Class ZCL_Z_EPM_PRODUCTS_DPC_EXT (Data Provider), redefine method EPMPRODUCTS_GET_ENTITYSET

Now the service can be executed with the entity set */sap/opu/odata/z_epm_products/EpmProducts/?$format=json*, which should return the following result:



Additionally, paging by setting $skip and $top as URL parameter should work as well, e.g. */sap/opu/odata/z_epm_products/EpmProducts/?$format=json&$top=2&$skip=1* should return the second and the third entry of the list of products.



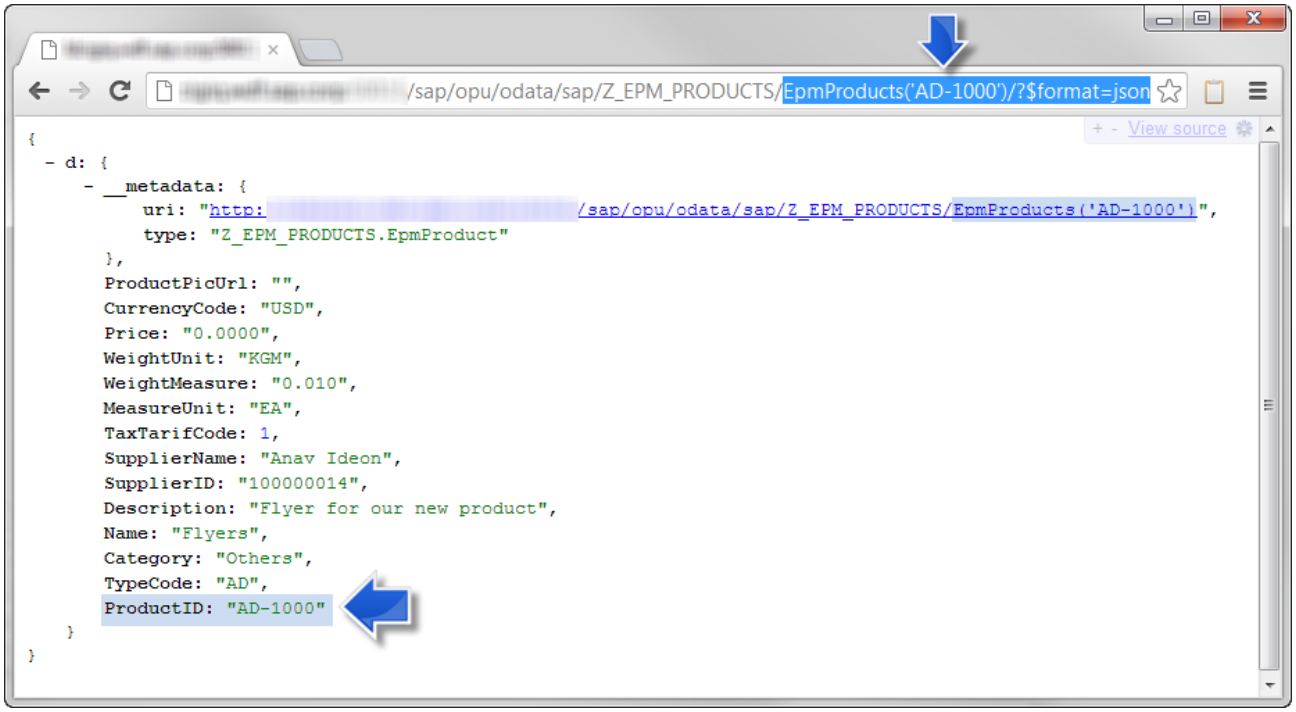
To enable requests for a single entity of EPM products the method *EPMPRODUCTS_GET_ENTITY* has to be implemented/redefined in class ZCL_Z_EPM_PRODUCTS_DPC_EXT.

```
METHOD epmproducts_get_entity.

  DATA lv_product_id TYPE bapi_epm_product_id.
  DATA ls_product    TYPE bapi_epm_product_header.
  DATA lt_return     TYPE TABLE OF bapiret2.
  FIELD-SYMBOLS <lf_key>       TYPE /iwbep/s_mgw_name_value_pair.

  READ TABLE it_key_tab INDEX 1 ASSIGNING <lf_key>.
  IF sy-subrc = 0 AND <lf_key> IS ASSIGNED.
    lv_product_id = <lf_key>-value.
    CALL FUNCTION 'BAPI_EPM_PRODUCT_GET_DETAIL'
      EXPORTING
        product_id = lv_product_id
      IMPORTING
        headerdata = ls_product
      TABLES
        return     = lt_return.
```

```
    MOVE-CORRESPONDING ls_product TO er_entity.
  ENDIF.

ENDMETHOD.
```

Source Code 2: Class ZCL_Z_EPM_PRODUCTS_DPC_EXT (Data Provider), redefined method EPMPRODUCTS_GET_ENTITY

Now the detail data for example the product *AD-1000* can be requested directly by using the URL */sap/opu/odata/z_epm_products/EpmProducts('AD-1000')/?$format=json*:



Note: The product picture URL path will usually be /sap/public/bc/NWDEMO_MODEL/. This path has to be activated in transaction SICF to enable that these pictures can be displayed in the browser.

Having implemented and tested this service, this is the perfect point in time to move from the application backend development to the application frontend development using the UI development toolkit for HTML5 (also known as SAPUI5).

Building the Application Frontend with the UI Development Toolkit for HTML5

Once you've built a working Gateway Service, the next step is to create an application that can consume this service. SAP places no restrictions on how Gateway services are to be consumed. In this article we focus on JavaScript using the UI development toolkit for HTML5 with its generic OData proxy object provided by the SAPUI5 runtime libraries.

SAPUI5 Application Frontend Architecture

The following figure illustrates the principal architecture of the SAPUI5 application frontend we will develop in the second part of this how-to guide. The technical details of this architecture will be explained in the corresponding sections of this chapter.

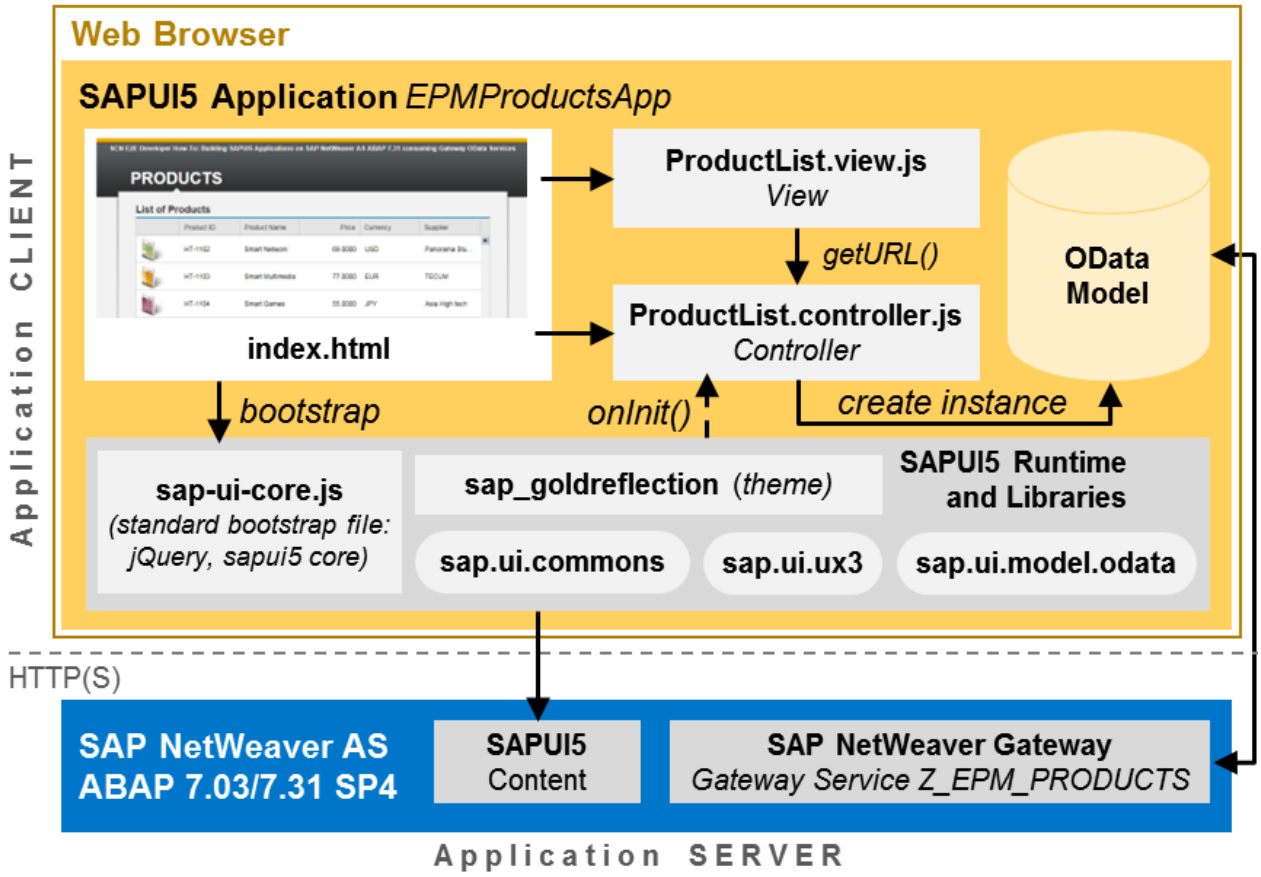


Figure 5: Architecture of SAPUI5 sample application frontend with Gateway service consumption (runtime view)

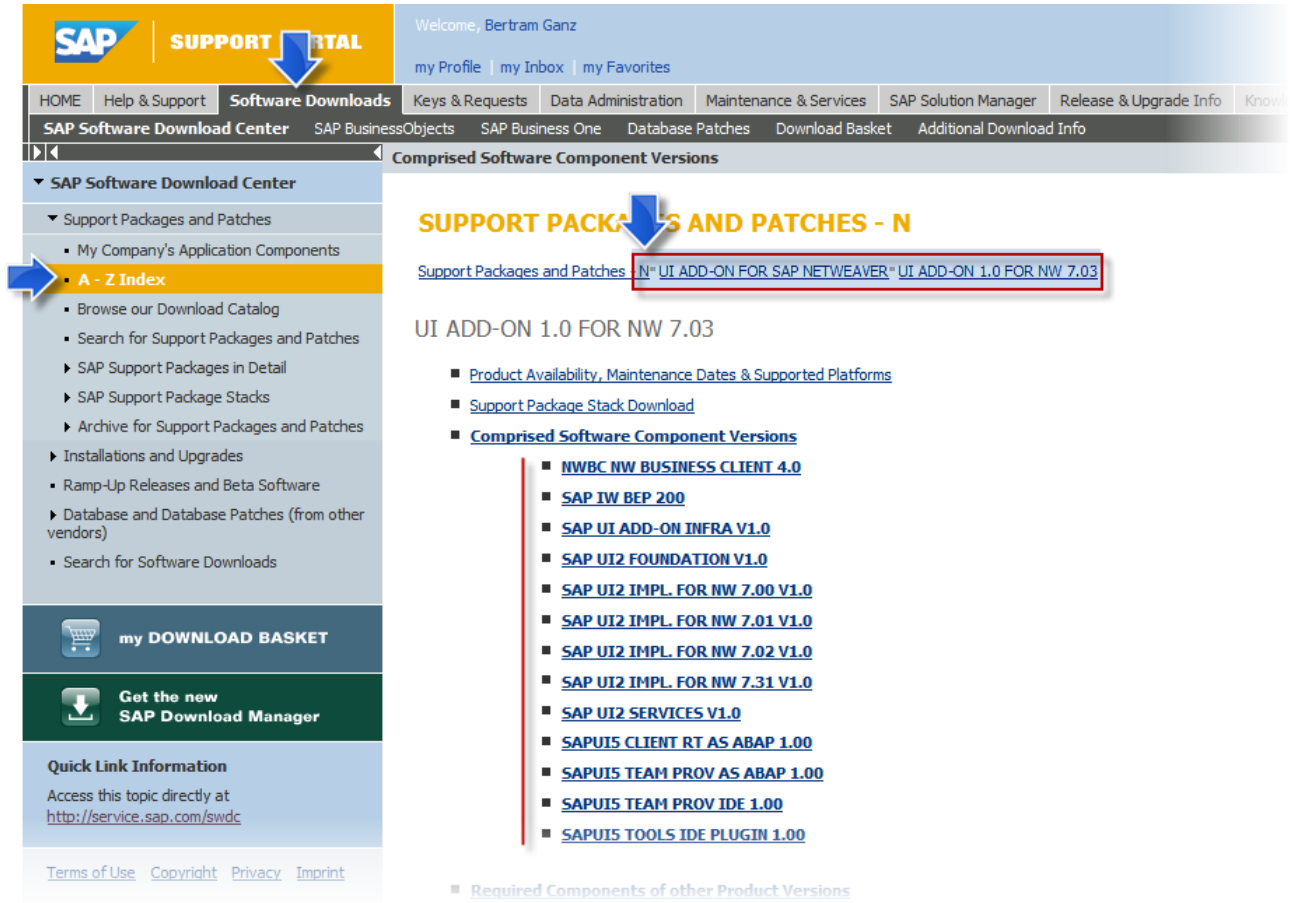
Setting up Your Development Environment

Installation of user interface add-on 1.0 for SAP NetWeaver (SAPUI5, UI2 Services, UI2 Backend …)

The **user interface add-on for SAP NetWeaver 1.0 SP01** (in short UI add-on for SAP NetWeaver) provides a number of tools and services enabling the implementation and lifecycle of HTML5 applications by using the UI development toolkit for HTML5 (also named SAPUI5) and UI integration services providing e.g. role-dependent content and its integration with as well as navigation to existing applications.

The UI add-on can be found on SAP Support Portal and has to be imported with transaction SAINT (see SAP Note 1666368):

- SAP Support Portal: http://service.sap.com/swdc → Support Packages and Patches → A – Z- Index → N → UI add-on for SAP NetWeaver (see Screenshot 22)
- SAP Online Help: http://help.sap.com/nw-uiaddon → Master Guide, Installation Guide
- SAP Note: 1666368 - Installing NetWeaver UI Extensions Version 1.0



Screenshot 22: Downloading the UI add-on for SAP NetWeaver 7.03 from SAP Support Portal

What's the user interface add-on for SAP NetWeaver?

The *User interface add-on for SAP NetWeaver* (AS ABAP 7.0/7.01/7.02/7.03/7.31) can be used to rapidly provide new user interaction and technology without disrupting the underlying business applications. It addresses mainstream SAP landscapes and deployment scenarios at existing customers to help improve their user interface (UI) independently of application releases and without major investments, such as upgrading existing systems or challenging their IT landscapes. The goal is to integrate existing UI functionality while making use of state-of-the-art UI technology and functionality provided by SAP to bring significant UI improvements to customers and end users irrespective of the application lifecycle and the SAP NetWeaver platform version used.

User interface add-on for SAP NetWeaver contains the UI development toolkit for HTML5 (SAPUI5), SAP NetWeaver Business Client 4.0 with a completely refined user experience as well as SAP NetWeaver user interface services for HTML5 and mobile application developers based on SAP NetWeaver Gateway, including a RESTful OData service for navigation based on the launchpad.

- SCN blog: Introducing the new UI Add-On for SAP NetWeaver, Filip Misovski, SAP AG, September 2012
- SAP Online Help: http://help.sap.com/nw-uiaddon
- SAP Support Portal: http://service.sap.com/swdc → Support Packages and Patches → A-Z Index → N → UI add-on for SAP NetWeaver
- SAP Service Marketplace: SAP Note 1759682 – UI Add-On for SAP NetWeaver: Central Note

Installing SAPUI5 development tools (Eclipse, SAPUI5 ABAP team provider …)

Although there are a number of ways to implement HTML/JavaScript applications SAP recommends the usage of the SAPUI5 developer tools (including SAPUI5 ABAP repository team provider and the SAPUI5 application development toolkit), which are delivered as an Eclipse plugin. The download and installation of the UI development toolkit for HTML (SAPUI5) packages and all the release-dependent information is described in more detail in SAP note 1747308 - UI development toolkit for HTML5 (SAPUI5) where you can download the Installation Guide – UI development toolkit for HTML5.The installation guide comprises detailed information on the installation prerequisites and procedures (SAPUI5 runtime on SAP NetWeaver AS ABAP, SAPUI5 tools).

The *SAPUI5 ABAP repository team provider* (in short SAPUI5 ABAP team provider) is responsible for storing and synchronizing all the relevant UI artifacts within the ABAP backend system, which results in a common lifecycle of the OData services and the user interface. The SAPUI5 application development feature supports the integration of SAPUI5 development in Eclipse with e.g. wizards for SAPUI5 application projects, views & controllers, JavaScript code completion, templates, snippets, etc.).

As a prerequisite a SAP GUI installation at least 7.20 Patch Level 9 and the Microsoft VC runtime libraries (download) are required.

Note: Currently the SAPUI5 ABAP repository team provider just works with the 32-bit installation of Eclipse and it is only available with SAP NetWeaver 7.03/7.31 SP4 and the corresponding UI add-on 1.0 SP01 for SAP NetWeaver 7.03/7.31. Additionally the Team Provider relies on the ABAP in Eclipse communication framework, which is part of the ABAP Development Tools for SAP NetWeaver. For more details on these tools and an ABAP Development Tools for SAP NetWeaver installation guide see SAP note 1718399. On SAP NetWeaver 7.00, 7.01, 7.02 and 7.03/7.31<SPS04 it is not possible to directly deploy or upload applications from the local Eclipse installation to the ABAP server. Deployment must take place manually or by applying SAP Note 1793771.

If you have an SAP system with a NetWeaver release prior to 7.03/7.31 SP4, you may install a NetWeaver instance 7.03/7.31, connect the SAPUI5 ABAP Repository Team Provider to this system and transport afterwards the UI artifacts to the system with the lower release.

SAPUI5 SDK - Demo Pages and Documentation

But before creating the project and implementing the sample application it might be useful to refer to the *SAPUI5 SDK Demo Kit*: https://sapui5.netweaver.ondemand.com/sdk/#content/Overview.html. After installation of the UI Extension Add-On you should be able to access the Demo Kit by URL *http://<host>:<port>/sap/public/bc/ui5_ui5/demokit/*.

It provides a developer guide with a complete list of controls (with sample source code), a JsDoc API reference, and a test suite, where all the controls are visualized with numerous configurations. With the help of this demo kit you are able to enhance the sample application or to create your own application project from scratch.

Implementing the application frontend with the UI development toolkit for HTML5 (SAPUI5)

What's the UI development toolkit for HTML5 also known as SAPUI5?

It is SAP's HTML5 toolkit for quickly building lightweight business UIs on multiple platforms. It provides a new enterprise-ready HTML5 rendering library for client-side UI rendering and programming by combining the advantages of being open and flexible as well as being enterprise ready supporting all SAP product standards. While Web Dynpro is best suited to heavyweight transactional applications for expert usage, SAPUI5 is designed for building lightweight consumer-grade UIs for casual usage. It targets developers at SAP and customers with web development skills (HTML, CSS3, JavaScript). SAPUI5 provides extensible controls and powerful theming but it is easy to consume, based on open standards and integrates with 3rd-party JavaScript libraries like JQuery. SAPUI5 applications run on a wide range of devices (smartphone, tablet, desktop) and on multiple server platforms, like SAP NetWeaver AS ABAP or Java, SAP NetWeaver Cloud or Sybase Unwired Platform.

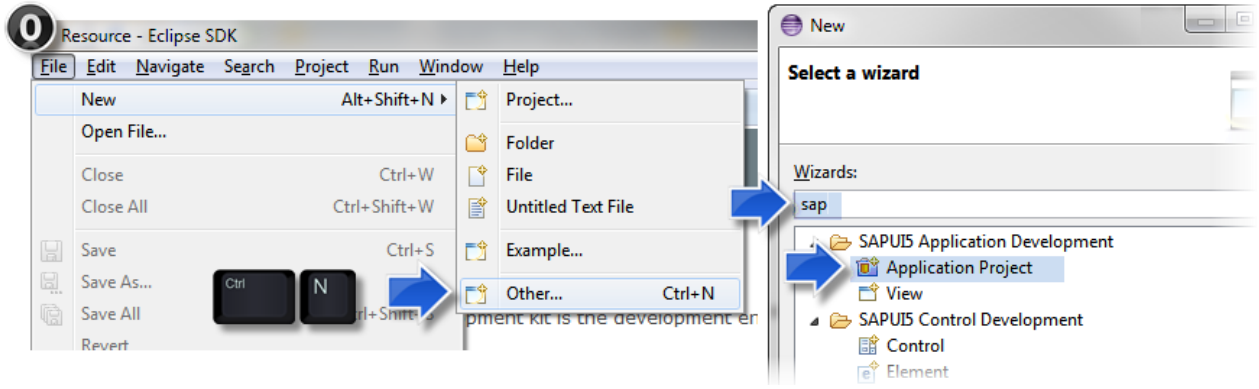
- SAP Community Network article: Get to Know the UI Development Toolkit for HTML5 (aka SAPUI5)
- SAP Community Network space: UI Development Toolkit for HTML5 Developer Center
- SAP Service Marketplace: SAP note 1747308 - UI development toolkit for HTML5 (SAPUI5)

Creating an Application Project for SAPUI5 with a JavaScript View

The procedure for creation of a new SAPUI5 application project is divided into the following steps:

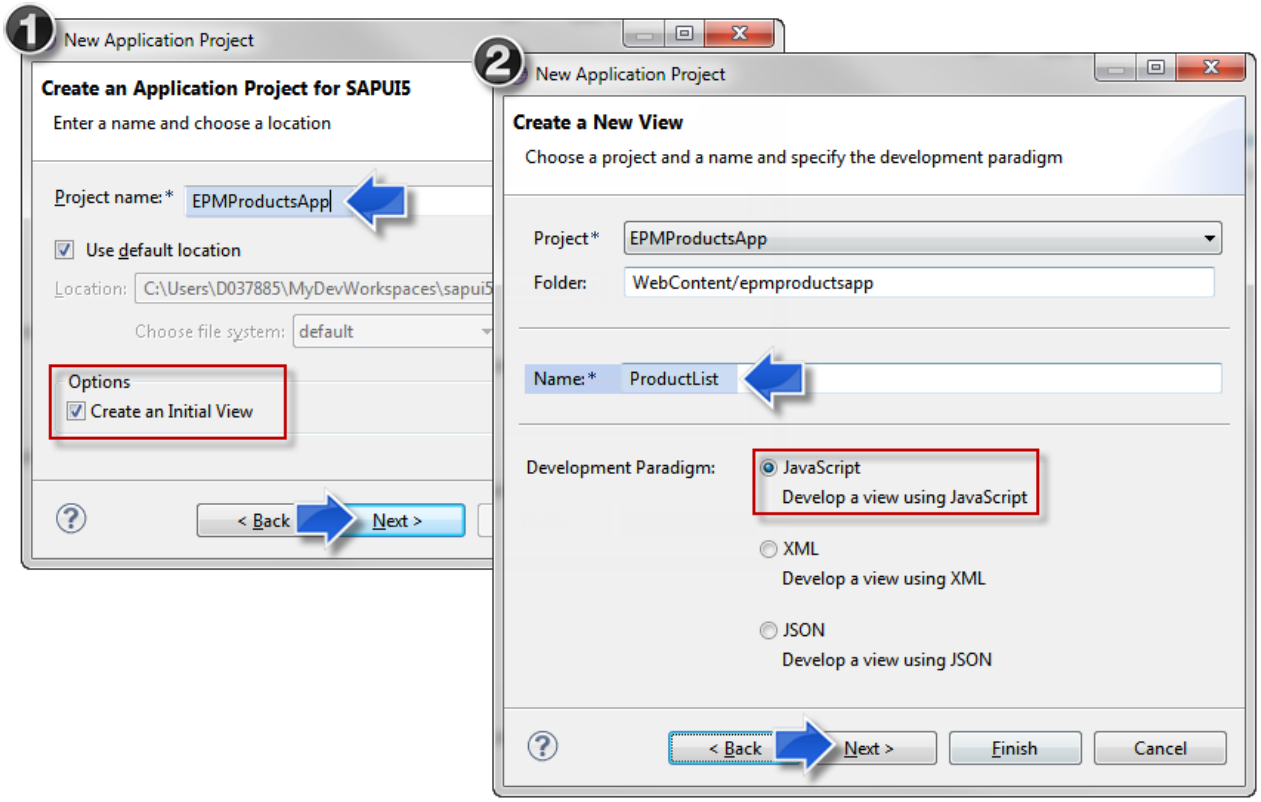
- Step 0: Start SAPUI5 application creation wizard
- Step 1: Fill project related data
- Step 2: Fill view related data
- Step 3: Display confirmation page (optional)

Within the project explorer of the SAPUI5 Eclipse installation you have to select the menu entries *File / New / Other* and by writing query string "**ui5**" in the 'Wizards' input field you will get a list of all SAPUI5 wizards. Here you choose item *SAPUI5 Application Development → Application Project* (Screenshot 23, step 0).



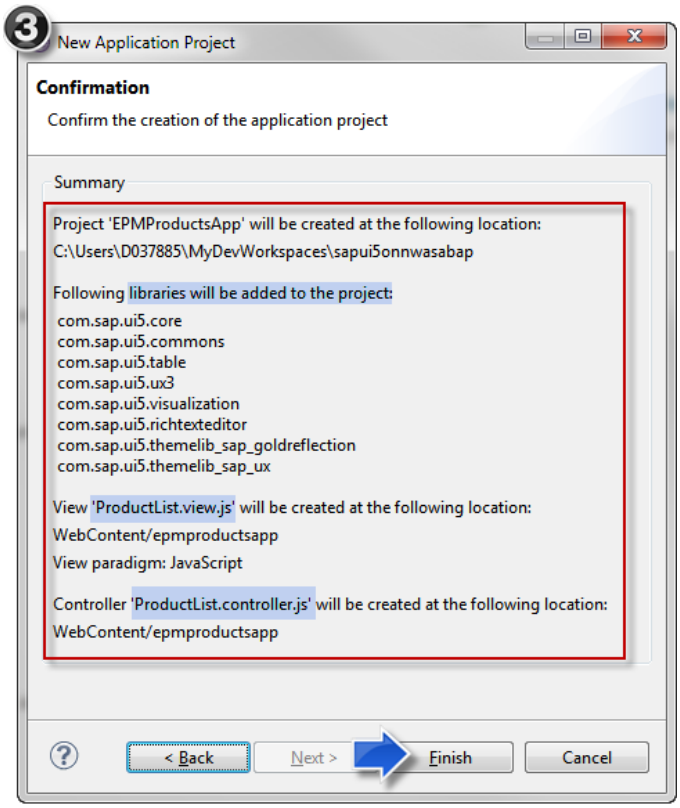
Screenshot 23: Step 0 – starting the SAPUI5 application project creation wizard

On the next screen of the wizard a project name has to be entered (here *EPMProductsApp* was used) and you can optionally choose the location, where this project will be stored locally and if an initial SAPUI5 view should be created (Screenshot 24, step 1). In our case keep the initial view checkbox checked. In the next dialog step you fill the view related data: folder location within the project, the view name *ProductList* and the view type (development paradigm) *JavaScript* (Screenshot 24, step 2).



Screenshot 24: Creating a new application project for SAPUI5 – entering project name and view name

Before finishing the wizard the summary information about selected data in previous pages is displayed in case the initial view should be created (Screenshot 25). With „Back"-navigation the entered data could be corrected and displayed/controlled again before the SAPUI5 application project is created.

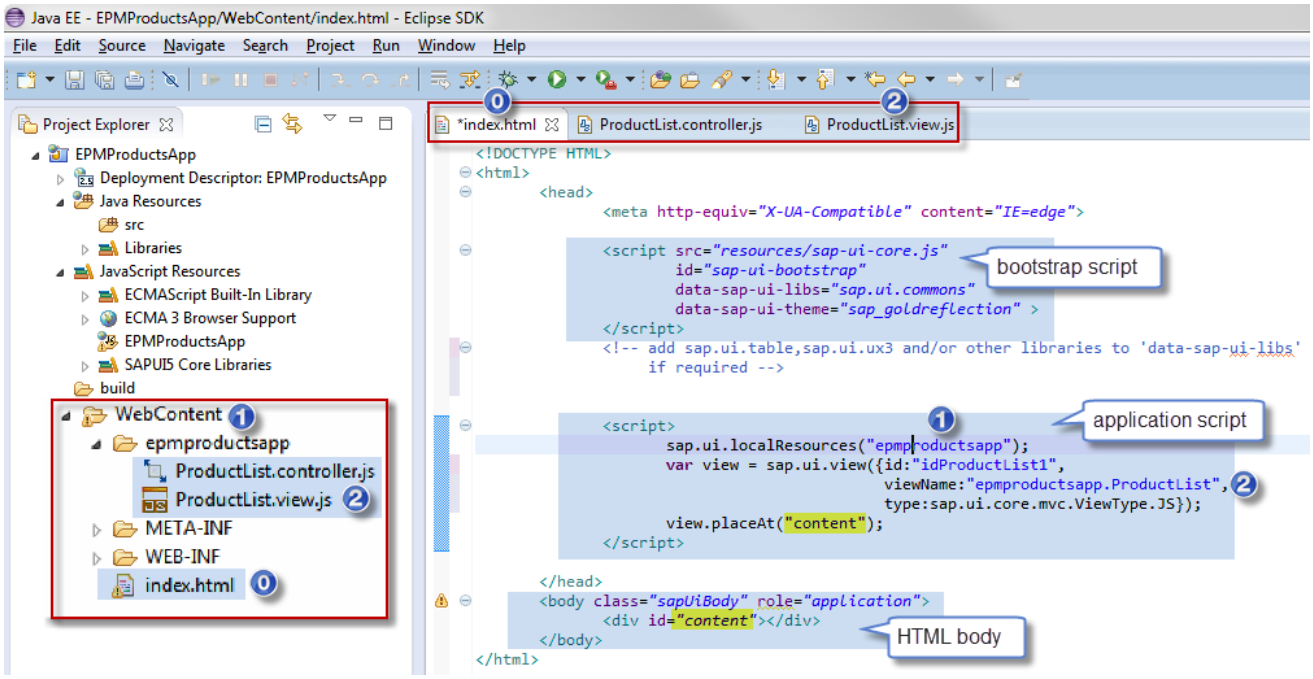


Screenshot 25: Optional confirmation page for creation of new SAPUI5 application project

As a result, the following SAPUI5 application parts are created:

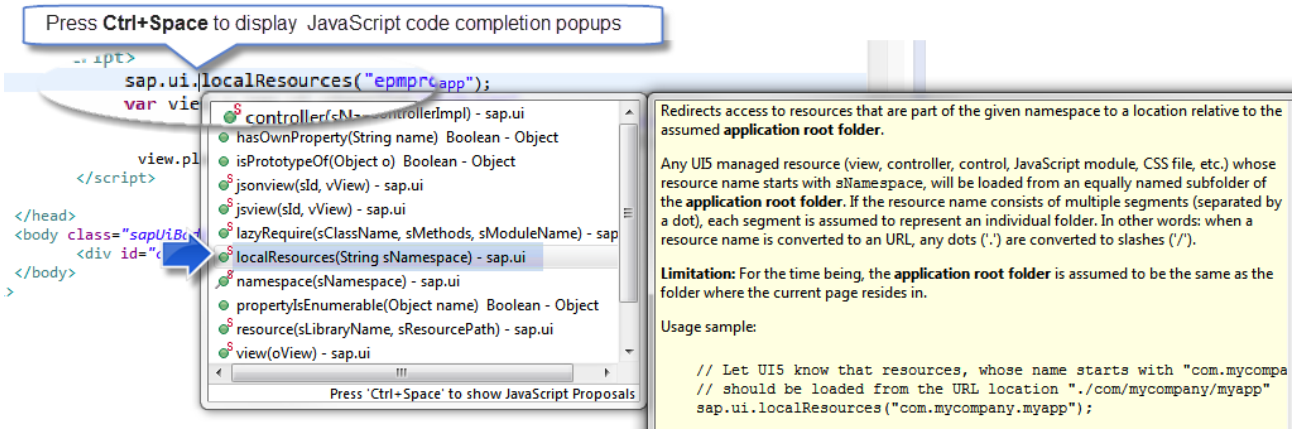
- **ProductList.view.js:** A view file of type JavaScript: (see Screenshot 26, point 2)
- **ProductList.controller.js**: A controller file of type JS
- **index.html:** file containing references for the libraries *sap.ui.core* and *sap.ui.commons*, the used theme where the default theme is *sap_goldreflection*, as well as information for the script type and the script id (bootstrap script, see Screenshot 26). In a second script block (application script), the index file refers to the project name, view type and name, a content default for placing the controls on the UI later on (in HTML body), the CSS class and the ARIA application role. Inside the JavaScript block of the index.html there is JavaScript code completion for SAPUI5 controls. We take a closer look at the *index.html* source code later in section Understanding the SAPUI5 Application Code (Basics), see Source Code 4.

Additionally the following functions were also processed: creation of a new *dynamic web project*, creation of a **web.xml** file containing settings for the *ResourceServlet* and *SimpleProxyServlet* (usage of *SimpleProxyServlet* is restricted to localhost and only intended for testing purposes and not for productive use), addition of installed SAPUI5 UI lib plugins to the Java build path and to the deployment assembly to enable local testing and addition of the SAPUI5 class path container (if available) to the JavaScript include path to enable the JavaScript code completion for SAPUI5 controls.



Screenshot 26: index.html with SAPUI5 bootstrap script, application script and HTML body

Note: If you rename the view or controller file or move them to a different folder, the coding in the view and controller and in the places where the view is used need to be adapted *manually*!



Screenshot 27: JavaScript code completion for SAPUI5 API

Using SAPUI5 ABAP Team Provider to deploy the SAPUI5 Application Project on the ABAP Server

Now this SAPUI5 application project exists locally in the working directory of the Eclipse IDE. Often it's useful to have a common lifecycle for the UI coding and for the services (e.g. all the development objects can be transported together into a productive environment). By using the *SAPUI5 ABAP team provider* all the sources can be stored within the backend application system SAP NetWeaver AS ABAP.

RECOMMENDATION: Enable SSO for Your ABAP System

For the sake of convenience, and again for security reasons, use also the single sign-on (SSO) option for system authentication (if it is available in your SAP NetWeaver system landscape). Compared with SNC, SSO meets even more the security requirements for working with ABAP projects. Using SSO, the user does not need to enter a user ID and password, but can access the specific system directly after the system has checked the validity of the logon ticket. To enable SSO for an ABAP system ...

- Install the *SAP NetWeaver Single Sign-On 1.0 SP03* or higher (either "*Secure Login Client*" or "*Enterprise Single Sign-On*") for the corresponding platform (either for Windows 32- or 64-Bit).

- Configure the Secure Network Communication between ABAP Development Tools client and the ABAP back-end system. For more information, see http://scn.sap.com/community/netweaver-sso.

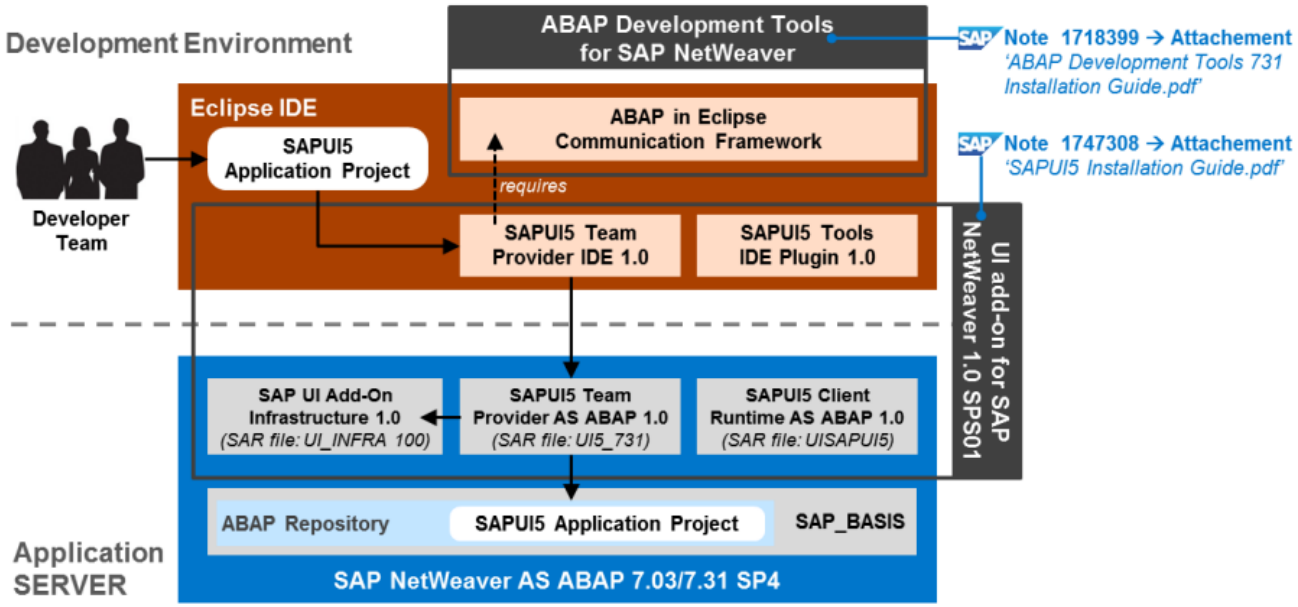


Figure 6: Installation of SAPUI5 ABAP Team Provider to connect to an ABAP backend system on SAP NetWeaver 7.31

Note: In order to deploy SAPUI5 applications built with the Eclipse-based tools to an ABAP environment, there are 2 options depending on the SAP NetWeaver release:

1/ SAP NetWeaver 7.03/7.31: Direct integration via the SAP ABAP repository team provider allows easy and fast deployment. The SAP ABAP repository team provider functionality, which is available in the Eclipse IDE, is available in SAP Business Suite systems with SAP NetWeaver 7.31 containing the UI add-on for SAP NetWeaver in the software components *UI_INFRA* and *UI5_731*. For more information, see the User Interface add-on for SAP NetWeaver Developer Guide.

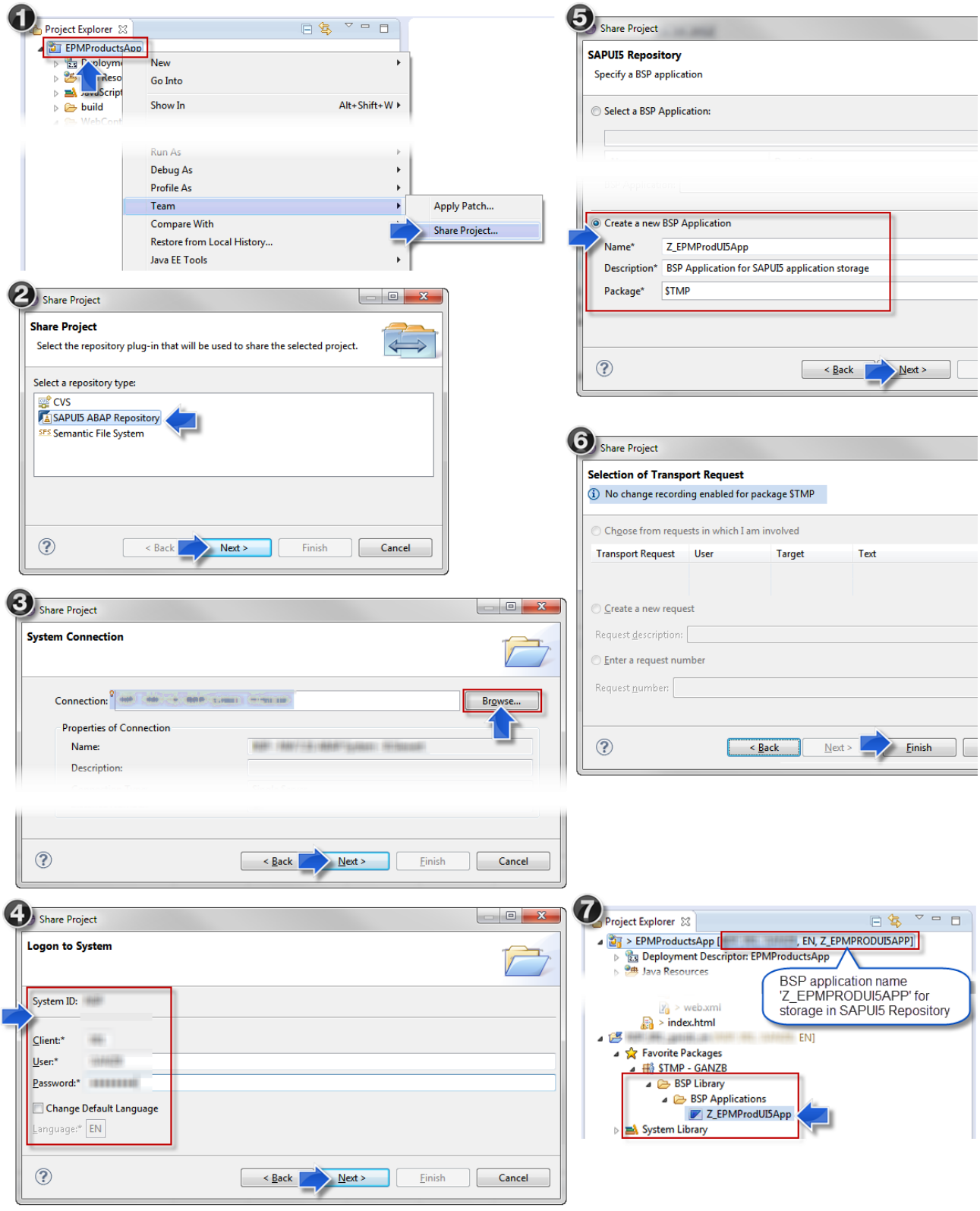
2/ SAP NetWeaver 7.00, 7.01, 7.02 and 7.03/7.31 < SPS04: In these releases, it is *not possible* to directly deploy or upload applications from the local Eclipse installation to the ABAP server. Deployment must take place manually. For more information, see the User Interface add-on for SAP NetWeaver Developer Guide.

For more information on how to use the SAPUI5 ABAP repository and team provider see the tutorial *Synchronizing with the SAPUI5 Repository* inside the UI development toolkit for HTML5 – developer guide.

On Screenshot 28 you can see the wizard steps needed to share a SAPUI5 application project with your team by deploying to the ABAP system. To connect your project with the backend you have to do a right mouse click on the project name (in the project explorer, see Screenshot 28-1) and select the menu path *Team / Share Project*. On the next screen you can select a repository plug-in – in our use case it's the *SAPUI5 ABAP Repository*. Choose *Next*. Configure the connection to the ABAP system by using the *Browse…* button. You can only select system connections that are configured in the SAP GUI Launchpad providing your authentication credentials (Screenshot 28-3 and 4). Then you have to specify the name, a description and the development package of the BSP Application (in our example Z_EPMPRODUI5APP, see Screenshot 28-5), which will be created as the corresponding repository object (if a transportable package is selected, in the next step a transport request has to be selected or created).

In Screenshot 28-7 you see, that your shared SAPUI5 application is now connected to the SAPUI5 BSP application artifact of the SAPUI5 Repository. A description containing the ABAP system, client, user, language, and SAPUI5 BSP application name is displayed next to the project name.

Note: If you have shared a SAPUI5 application project, the SAPUI5 runtime library version of the server will be compared against the ones which have been installed into Eclipse and a warning might be shown. You can find more details about this check section '*What's a SAPUI5 Runtime Libraries Server Version Check?*' below.

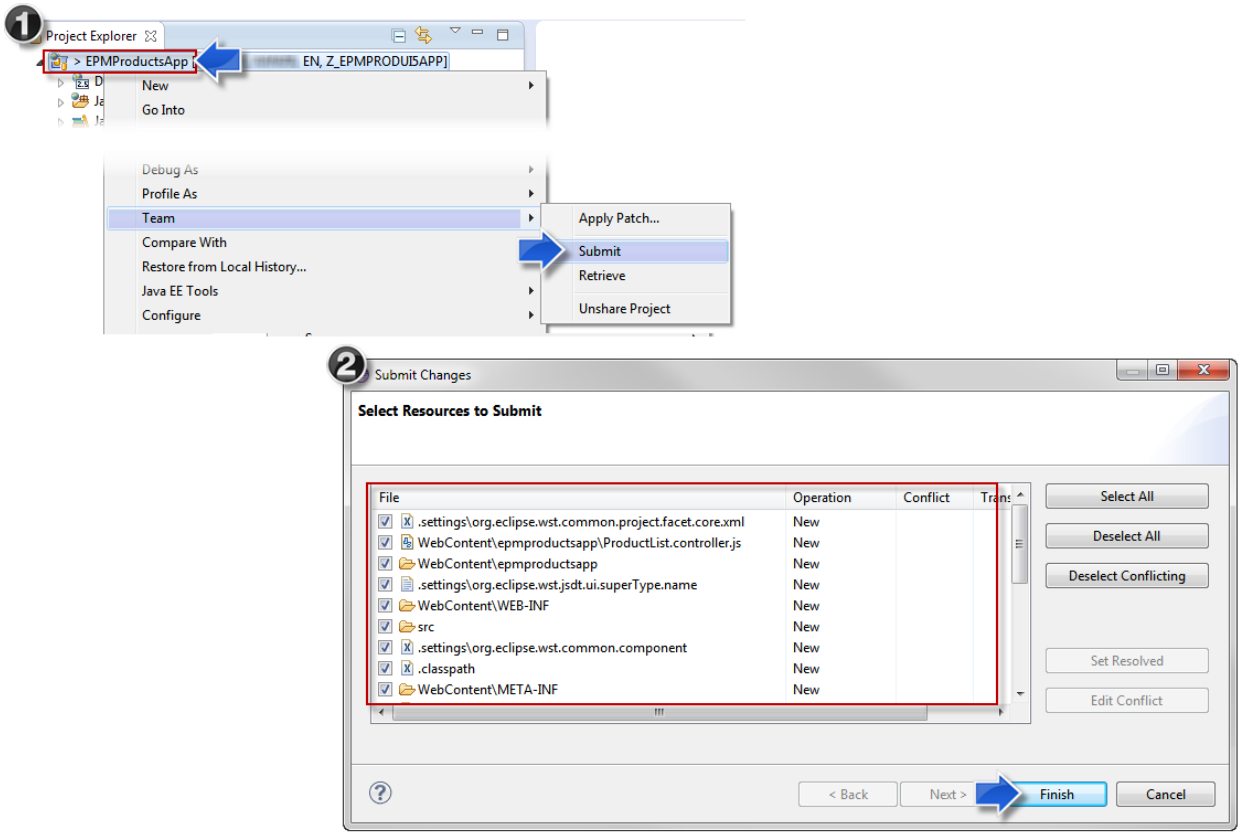


Screenshot 28: Sharing a SAPUI5 application in the SAPUI5 Repository as BSP application

All that has happened is that you have identified to the SAP system that the contents of your SAPUI5 project are, ***at some point in the future***, going to be shared. However, no file sharing has yet taken place! The "Share Project" step is a one-off preparatory step that does not now need to be repeated.

In the project explorer select the SAPUI5 application node again and open context menu item *Team / Submit* (Screenshot 29-1). If a logon is required, enter your password in the logon popup and choose *OK*. In the last step you have to select all resources, which will be submitted to the ABAP backend system (all resources are

selected by default). You will get a list of files that have been modified (added, updated, or deleted) in the client. The dialog shows the files for which submit conflicts exist, for example, if another user has submitted a newer file version in the meantime. Submit conflicts must be resolved before submission. If a file is already locked in a transport request, the corresponding request is shown in the dialog. If the BSP application (created to store your SAPUI5 application resources on the ABAP system) belongs to a transportable ABAP package, you have to choose a transport request. Note that transport requests are not automatically released when the files are submitted; you still have to release them using transaction SE09 in the underlying ABAP system.



Screenshot 29: Submitting a SAPUI5 application to the SAPUI5 Repository on an ABAP system

Note: Enabling Virus Scan During Upload

When uploading files to the SAPUI5 Repository, you can perform a virus scan. As of SAP NetWeaver 7.00 with UI add-on, SAP delivers the following virus scan profile for ABAP within the UI add-on for SAP NetWeaver: `/UI5/UI5_INFRA_APP/REP_DT_PUT` .This profile is used by the SAPUI5 Repository API to store files in the SAPUI5 Repository based on BSP Repository. For example: Upload of a local file using SAPUI5 Repository API `/UI5/CL_UI5_REP_DT`, method `/UI5/IF_UI5_REP_DT~PUT_FILE` from 7.00 on, or the SAPUI5 Team Repository Provider in SAP NetWeaver 7.31.

The profile is deactivated when delivered. To activate it, first create at least one basis profile and save it as the default profile. You can then activate one of the delivered profiles. By default, it links to a reference profile, which is the default profile. For more information, see SAP Help Portal:

- ABAP-Specific Configuration of the Virus Scan Interface (7.00)
- ABAP-Specific Configuration of the Virus Scan Interface (7.31)

Note: Tracking Coding Changes and Text Changes in the SAPUI5 Repository

Code changes can be tracked by using the usual ABAP version control of the corresponding resource file. A new version is created when a new transport is written. Text changes can be tracked by using the "Table History" transaction (SCU3), the relevant tables for SAPUI5 texts are /UI5/TREP_TEXT and /UI5/TREP_TEXT_T for the translated text. Table logging has to be activated in the system for this functionality. For more information, see SAP Help Portal.

What's a Server Version Check of SAPUI5 Runtime Libraries?

When you are developing SAPUI5 applications with the SAPUI5 Tools the code completion and application preview features are based on the SAPUI5 runtime libraries, which has been installed in your Eclipse installation. However after you have deployed the application to the ABAP server and execute it there, it will use the SAPUI5 runtime libraries, which have been installed on the ABAP server.

It is recommended to always have the same library version installed in Eclipse as well as on the ABAP server. If this is not the case (e.g. if the local version is higher than the one on the server) the following could happen:

- During development you could use features which are not available on the server, yet.
- When testing in Eclipse the application might behave differently, e.g. because in the newer runtime version some issues have been fixed, which still occur on the server.

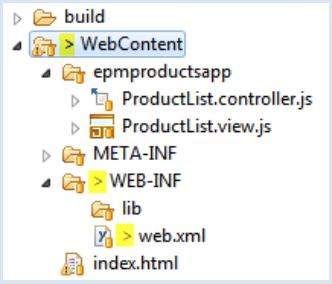
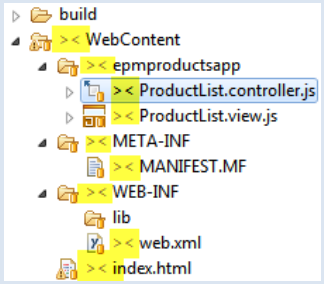
Please check the compatibility rules which apply for the SAPUI5 runtime libraries.

When you share a SAPUI5 Application project with the SAPUI5 ABAP Repository or start the submit wizard, a version comparison of the local libraries against the server will be performed. In case they are different, a warning popup will be shown which tells you the current versions. In case you want to continue with that, the popup can be suppressed for that project and this version constellation.

To prevent the above problems from occurring, when using different runtime library versions locally and on the server, the following measures are recommended:

- Check the JavaScript Documentation of used controls and their methods for **@since** tags. They indicate which version has introduced a new feature which you are going to use.
- When testing in Eclipse you should configure it to use the runtime libraries located on the server instead of the local ones as described here.
- Always test your application on the server after submitting changes.

The following table illustrates how differing versions of local and remote files are visualized in the SAPUI5 project tree:

Sync State	Symbol	Exampel A	Example B
Local file changed	>	web.xml changed locally 	several files changed locally and remotely 
Remote file changed	<		
Both files changed	><		

Testing the SAPUI5 Application Locally

Already now this application can be called in the browser, but to see at least a small text, the generated view has to be enhanced with one (marked) statement:

```
sap.ui.jsview("epmproductsapp.ProductList", {

      getControllerName : function() {
         return "epmproductsapp.ProductList";
      },

      createContent : function(oController) {
         return new sap.ui.commons.TextView({text:"Hello World"});
      }

});
```

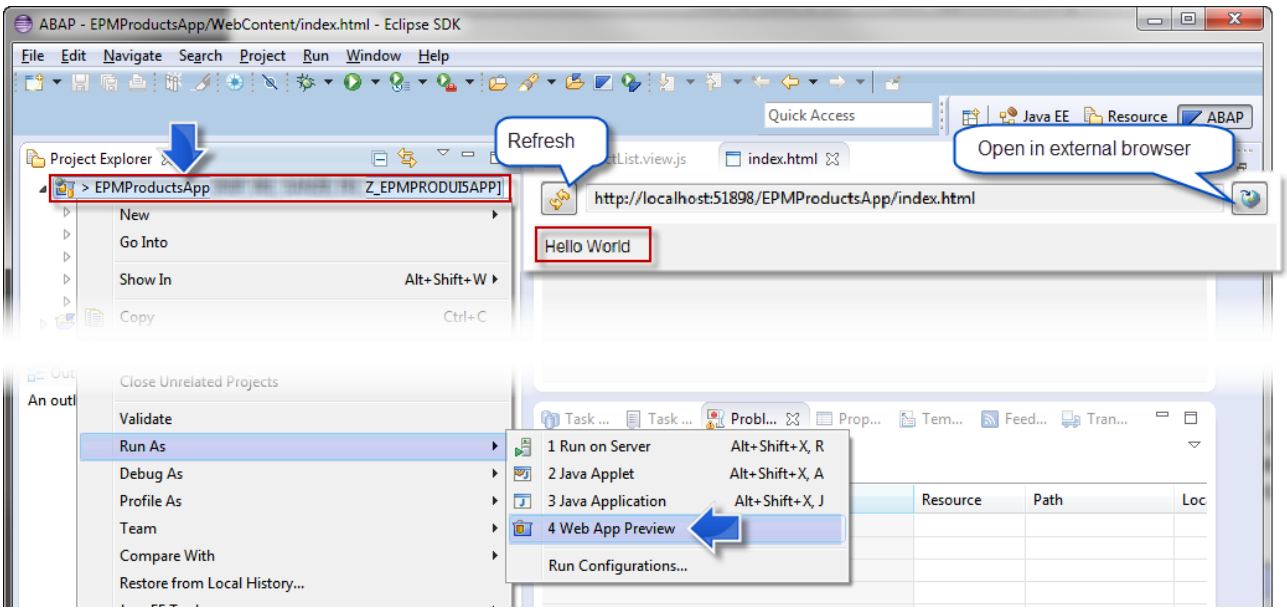
Add new **TextView** control to the view and set text property

Source Code 3: JavaScript view ProductList.view.js in SAPUI5 application EPMProductsApp/Web Content/epmproductsapp

Note: Try to use the code completion there for the *TextView* control (shortcut Ctrl + Space).

Note: After adding the new line of code to the view, the source code has to be submitted (*Team / Submit*) to the SAPUI5 ABAP Repository.

To test the new application with the Web Application Preview on an embedded Jetty server, right-click the HTML file or the project node and choose *Run As → Web App Preview*. Everything is configured automatically. After changing a file of your SAPUI5 application project, you can refresh the SAPUI5 preview by choosing the *Refresh* button on the left in the preview editor.



Screenshot 30: Testing new SAPUI5 application with Web application preview inside the Eclipse IDE

To check the files of your SAPUI5 application project in an external browser (IE/FF), choose *Open in external browser* on the right in the preview editor. This opens the external browser marked as the default browser on your PC. You can also copy the URL from the text field of the editor to an external browser (this is useful for external browsers other than the default browser).

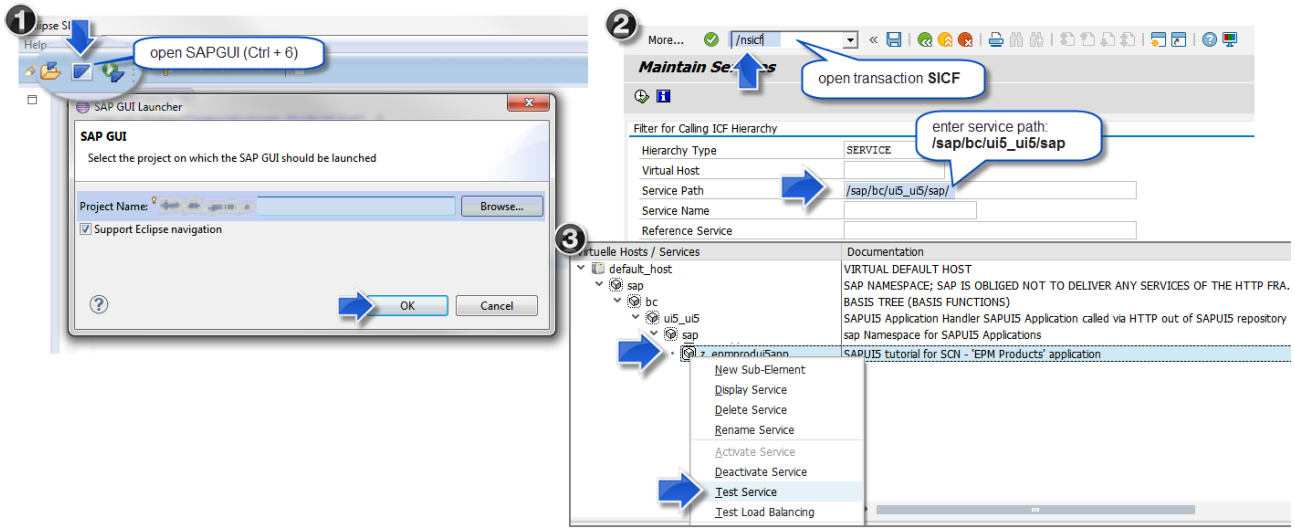
Testing the SAPUI5 Application on the ABAP Server

All files have been submitted to the SAPUI5 Repository and are available under the corresponding SAPUI5 BSP application. The SAPUI5 application project can now be run from a Web browser calling the underlying ABAP system. You actually have two options to get the SAPUI5 application URL: first, by context menu item '*Test Service'* in transaction SICF (requires to launch SAP GUI inside an Eclipse perspective view); second, by directly entering the browser URL when the SAPUI5 application URL schema is known.

Note: In the SAPUI5 Tools version used for this document it was not yet possible to run a deployed SAPUI5 application in the browser via a corresponding SAPUI5 tools function. An upcoming SAPUI5 tools version will provide a new "*Run as - Run on Server"* context menu function to test a remote SAPUI5 application (deployed on an ABAP system) in a Web browser.

For the first option open SAP GUI in Eclipse via shortcut Ctrl+6 or via the corresponding toolbar button (see Screenshot 31-1). Select your ABAP project and if necessary enter your password. In the SAP GUI view enter transaction code '*nsicf*' to start transaction SICF. Enter a service path of `/sap/bc/ui5_ui5/sap/` and apply the filter. Select node item *z_epmproduui5app.* This node corresponds to the BSP application Z_EPMPRODUI5APP (as SAPUI5 repository counterpart to SAPUI5 project *EPMProductsApp*). With context menu item 'Test Service' the deployed SAPUI5 application gets started in a Web browser.

For the second option open a Web browser and manually enter the SAPUI5 application URL based on the given schema:

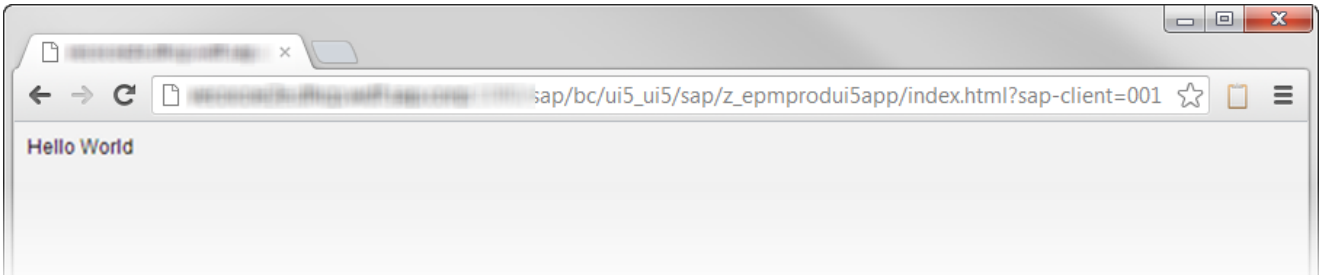


Screenshot 31: Testing SAPUI5 application on ABAP server via transaction SICF

URL: `https://<hostname>:<port number>/sap/bc/ui5_ui5/<namespace>/<application name>/index.html`

our URL: `https://<hostname>:<port number>/sap/bc/ui5_ui5/z_epmprodui5app/index.html`

You should see a Hello World on the typical grey background of the *sap_goldreflection* theme.



Understanding the SAPUI5 Application Code (Basics)

Having seen the running (nearly completely generated) application, let's have a deeper look into the existing code. The application starts with the page *index.html* (see , which contains in the header section at first the SAPUI5 bootstrap with the location of the core JavaScript file, the needed control libraries and the theme. Additionally the SAPUI5 view is instantiated and placed at an anchor. Within the HTML <body> tag there is besides the SAPUI5 CSS class and the ARIA role (accessibility) just the anchor for the view.

```
<!DOCTYPE HTML>
<html>
  <head>
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
      <script src="resources/sap-ui-core.js"
              id="sap-ui-bootstrap"
              data-sap-ui-libs="sap.ui.commons"
              data-sap-ui-theme="sap_goldreflection" >
      </script>
      <!-- add sap.ui.table,sap.ui.ux3 and/or other
       libraries to 'data-sap-ui-libs' if required -->

      <script>
         sap.ui.localResources("epmproductsapp");
         var view = sap.ui.view(
           {id:"idProductList1",
            viewName:"epmproductsapp.ProductList",
            type:sap.ui.core.mvc.ViewType.JS});
```

bootstrap script: SAPUI5 is implemented in JavaScript, so for loading SAPUI5 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.

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 local folder '/epmproducts'). The newly created instance of the *ProductList* view 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.

```
        view.placeAt("content");
      </script>
  </head>
  <body class="sapUiBody" role="application">
     <div id="content"></div>
  </body>
</html>
```

html body: This 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. `<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.

Source Code 4: index.html in SAPUI5 application (in folder WebContent)

The JavaScript view with name *ProductList.view.js* itself contains two functions:

- **`getControllerName()`**: assigns the id of the corresponding controller
- **`createContent()`**: returns the content of the view

The controller contains templates for four standard hook methods, which can be used to modify the view at particular points in time and to free resources, when the view is destroyed (documentation for the methods is available within the generated coding); additionally you can add own methods, which can be called e.g. from the *createContent()* method of the view.

```
sap.ui.jsview("epmproductsapp.ProductList", {

      getControllerName : function() {
         return "epmproductsapp.ProductList";
      },

      createContent : function(oController) {
         return new sap.ui.commons.TextView({text:"Hello World"});
      }

});
```

id of the view – it has to be the file location and its name (separated by dot character)

function returning the id of the controller

function returning the content of the view

add new **TextView** control to the view and set text property

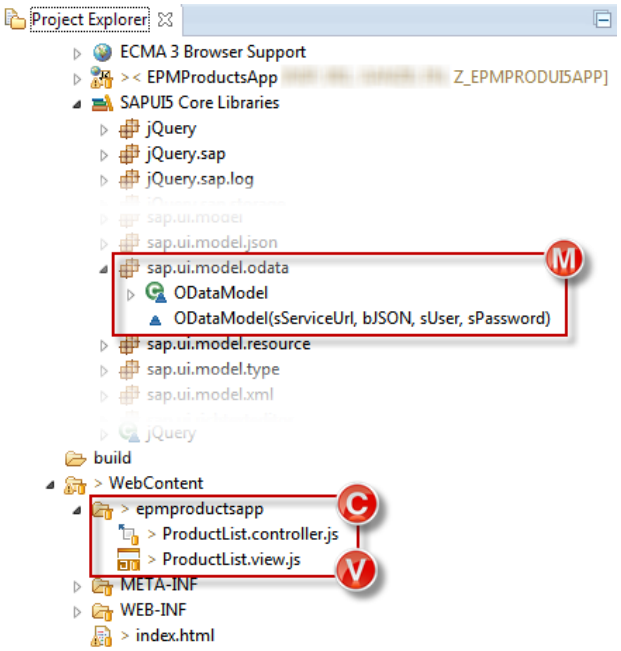
Source Code 5: JavaScript view ProductList.view.js in SAPUI5 application EPMProductsApp/Web Content/epmproductsapp

Consuming an OData Gateway Service for Product Data retrieval in SAPUI5

Since this is not a "Hello World"-Tutorial, but an end-to-end tutorial with OData Gateway consumption our sample application should be enhanced with the local OData Gateway service *z_epm_products* we created before in the first part (see section Implementing and Testing a local OData Service with the SAP NetWeaver Gateway Service Builder above).

The implementation follows the model-view-controller architecture that is supported by the SAPUI5 artifacts *view* (JavaScript file used in this tutorial, XML view or JSON view), *controller* (JavaScript file with suffix *.controller.js*) and *model* (OData model used in this tutorial, XMLModel or JSONModel).

In Figure 7 we explain the technical details on how the MVC principle is applied in our end-to-end sample application.



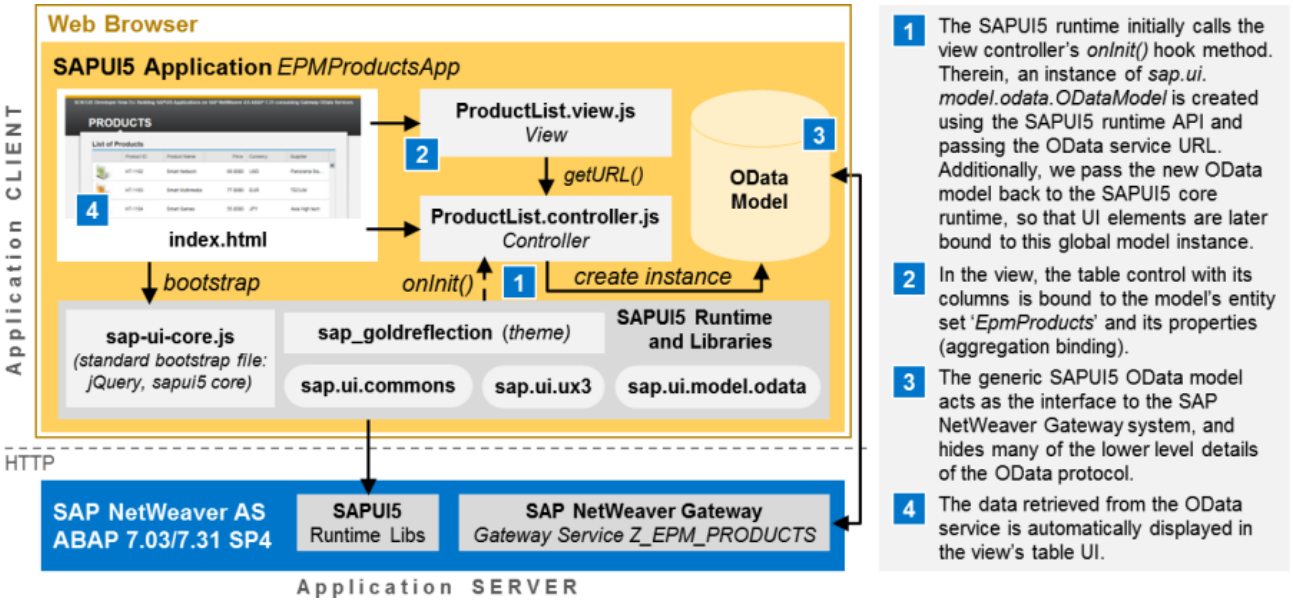


Figure 7: Gateway service consumption in SAPUI5 explained

In ProductList.controller.js: Creating the OData model instance and passing it to the SAPUI5 runtime

The *onInit()* hook method of the view controller's JavaScript file will create an OData model (associated with the implemented Gateway OData service). You must edit the highlighted sections and enter your own values for the userid and password. For sake of simplicity we assign the created OData model instance to the SAPUI5 core runtime as unnamed default model. Afterwards we do not need to explicitly assign the OData model to the view's table or popup controls as the SAPUI5 core runtime uses its unnamed default model instance assigned to itself in *onInit()*.

```
sap.ui.controller("epmproductsapp.ProductList", {
    /**
     * Called when a controller is instantiated and its View controls (if
     * available) are already created. Can be used to modify the View before it
     * is displayed, to bind event handlers and do other one-time initialization.
     */
    onInit : function() {
       // URL of the OData service - IMPORTANT: relative to the server
       var sServiceUrl = "/sap/opu/odata/sap/z_epm_products/";

       // create OData model instance with service URL and JSON format
       var oModel = new sap.ui.model.odata.ODataModel(sServiceUrl, true,
          "Your User", "Your Password");

       // Pass unnamed OData model to the SAPUI5 core runtime as global model for sake of
       // simplicity. Model does not need to be explicitly assigned to a view
       // or to UI controls later but acts as unnamed default model.
       // BUT BEWARE: the SAPUI5 core runtime can only hold one global unnamed model instance.
       // as default model. Every reassignment of another model instance to the SAPUI5 core
       // runtime replaces the default model instance set before. In this case views or UI
       // controls are potentially bound to the wrong model so that the data binding chain
       // from UI controls to model entities and properties gets
       // broken (i.e. controls get empty).
       sap.ui.getCore().setModel(oModel);
    },
...
```

OData model creation: to use **data binding** in a SAPUI5 applications we instantiate the OData model first. The con-structor takes the URL of the model data or the data itself as the first parameter.

OData model assignment to SAPUI5 core runtime (as unnamed default model) for data binding in the view's UI elements.

Source Code 6: OData model creation and assignment to SAPUI5 core as global default model in file ProductList.controller.js

BEWARE: For simplicity reasons we assign the OData model instance to the SAPUI5 core runtime here. Note that the SAPUI5 core runtime can only hold a single unnamed model instance (as default model) and that the reassignment of another unnamed model instance may break the data binding from UI controls to the model so that controls get empty.

In ProductList.view.js: Creating the view's table UI and binding it to OData entity set and properties

The *createContent()* method of the view will instantiate a table control, define the columns of the table, map the columns to the OData entity field names and bind the result data of the entity set to table rows. Also have a closer look at the comment boxes within the sample coding to better understand the implementation details and at Figure 7 illustrating the Gateway service consumption in SAPUI5.

```
sap.ui.jsview("epmproductsapp.ProductList", {

    getControllerName : function() {
        return "epmproductsapp.ProductList";
    },

    createContent : function(oController) {
        // return new sap.ui.commons.TextView({text:"Hello World"});
        // load table module, alternatively add sap.ui.table to
        // the central bootstrap
        jQuery.sap.require("sap.ui.table.Table");

        // create table control with properties
        var oTable = new sap.ui.table.Table({
            width : "100%",
            rowHeight : 50,
            title : "List of Products",
            selectionMode : sap.ui.table.SelectionMode.None
        });

        // define the columns, which should be displayed
        oTable.addColumn(new sap.ui.table.Column({
            width : "80px",
            flexible : false,
            template : new sap.ui.commons.Image({
                height : "45px",
                src : "{ProductPicUrl}"
            })
        }));

        oTable.addColumn(new sap.ui.table.Column({
            label : new sap.ui.commons.Label({
                text : "Product ID"
            }),
            template : new sap.ui.commons.TextView({
                text : "{ProductID}"
            }),
            sortProperty : "ProductID"
        }

        // alternatively (instead of curly braces syntax for property binding):
        // sap.ui.commons.TextView().bindText("ProductID")
        // alternatively:
        // sap.ui.commons.TextView().bindProperty("text","ProductID")
        ));

        oTable.addColumn(new sap.ui.table.Column({
            label : new sap.ui.commons.Label({
                text : "Product Name"
            }),
            template : new sap.ui.commons.TextView({
                text : "{Name}"
            }),
            sortProperty : "Name"
        }));
        oTable.addColumn(new sap.ui.table.Column({
            label : new sap.ui.commons.Label({
                text : "Price",
```

Property binding: add new **Column** control to the table and bind the table control property 'src' to the model property 'ProductPicUrl' with *curly braces syntax*

```
                textAlign : sap.ui.core.TextAlign.End
            }),
            template : new sap.ui.commons.TextView({
                text : "{Price}",
                textAlign : sap.ui.core.TextAlign.End
            }),
            sortProperty : "Price"
        }));
        oTable.addColumn(new sap.ui.table.Column({
            label : new sap.ui.commons.Label({
                text : "Currency"
            }),
            template : new sap.ui.commons.TextView({
                text : "{CurrencyCode}"
            }),
            sortProperty : "CurrencyCode"
        }));
        oTable.addColumn(new sap.ui.table.Column({
            label : new sap.ui.commons.Label({
                text : "Supplier"
            }),
            template : new sap.ui.commons.TextView({
                text : "{SupplierName}"
            }),
            sortProperty : "SupplierName"
        }));

        // In this sample we do not need to link the table control to the model.
        // SAPUI5 automatically binds the table to the global (unnamed default)
        // model assigned to sap.ui.core.Core befor in the view controller's
        // onInit() hook method (see comments there).
        // oTable.setModel(oModel);

        // bind table rows to the OData entity set
        oTable.bindRows("/EpmProducts");
        return oTable;

        // alternatively:
        // oTable.bindAggregation(sName, oBindingInfo)
        // oTable.bindAggregation( "rows", "/EpmProducts" );
    }
});
```

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 "/EpmProducts" points to the entity set with name "EpmProducts" defined in our Z_EPM_PRODUCTS OData Gateway service.

Source Code 7: Table UI creation with property and aggregation binding to the OData model in ProductList.view.js

Note: Avoid hostnames and ports within the JavaScript coding (stay relative at least to the server). This makes the coding independent from the system landscape (e.g. if Gateway is installed on a different server than the HTML/JavaScript coding is coming from and a reverse proxy is used).

Note: For simplicity reasons static label texts were used. These texts should be translatable and therefore either have to come from the backend (preferred) or from resource bundles. For more details see document Guide for SAPUI5 Application Developers - Translation in the SAPUI5 Demo Kit.

How is Data Binding applied in SAPUI5?

In SAPUI5, data binding is used to bind SAPUI5 controls to a data source that holds the application data, so that the controls are updated automatically whenever the application data is changed. With two-way-binding the application data is updated whenever the value of a bound control changes, e.g. through user input. Data binding supports binding of simple controls like *TextField* and list type controls like *DataTable* and *DropdownBox*.

SAPUI5 data binding comes with built-in support for three different model implementations: JSON model, XML model and OData model. Custom models can also be implemented. To use data binding in a SAPUI5 application you first need to instantiate the appropriate model and then assign it to the SAPUI5 core runtime object (like in this tutorial), to a view or

to specific controls like a table or a UI area. Afterwards control properties can be bound to model properties by either using a SAPUI5 specific curly braces syntax or by calling the *bindProperty()* method (*control property binding*).

Aggregation binding is used to bind aggregated controls to a collection of model entries, like binding multiple rows of a table to an entity set (e.g. products) of a model. To bind a control aggregation to the model you have two options. Firstly and in most cases, you create a so-called *template control* (i.e. a single control or even a tree of controls) which is automatically cloned and added to the parent control for each bound entry of the model's entity set. This is usually the right choice for structured data, where you have lists of entries with the same properties (e.g. use the *ListItem* control as template for the aggregation binding of a *ComboBox* control). Secondly and in advanced cases, you provide a *factory function* as the more powerful approach to create aggregated controls from model data. The factory function is called for every item in the list of bound model entries to aggregate entry-specific controls.

For control properties you can supply a *formatter function* which will be called with the value of the model property. The return value of the formatter function is used as the value of the bound control. When using aggregation binding, you can provide initial sorting and filtering.

Data binding supports the definition of types which can be handed over when binding properties. Bound properties with a defined type will automatically be formatted when displayed in the UI, input values in UI controls are parsed and converted back to the defined type in the model.

To catch invalid user input, you can register special event handlers for formatting, parse or validation errors and for validation success to the SAPUI5 Core.

- See the complete documentation on how data binding works and how to implement it in an application: Introduction to Data Binding in the SAPUI5 Demo Kit.

Handling Image Control Events

This application can already be tested now and it will present a list of EPM products, but to complete this first sample it is worthwhile to enhance it with **event handling**. Therefore the current view implementation will be extended with a click on the image, which will open a popup with a larger version of the image and a longer product description. Within the sample code below the popup is using the same *sap.ui.commons.Dialog* instance and just its content will be replaced. Since the data are already available the model instance can be reused. It has to be linked to the popup control and the binding context has to be set appropriately (to the binding context of the clicked image. Source Code 8 contains the modified part of the *createContent()* method of the *ProductList* view:

```
sap.ui.jsview("epmproductsapp.ProductList", {

    getControllerName : function() {
      return "epmproductsapp.ProductList";
    },

    createContent : function(oController) {

      // load table module, alternatively add sap.ui.table to the central
      // bootstrap
      jQuery.sap.require("sap.ui.table.Table");

      // create table control with properties
      var oTable = new sap.ui.table.Table({
          width : "100%",
          rowHeight : 50,
          title : "List of Products",
          selectionMode : sap.ui.table.SelectionMode.None
      });

      // create the popup for click on image
      var oDialog = new sap.ui.commons.Dialog({
          minWidth : "300px",
          minHeight : "240px"
      });

      // define the columns, which should be displayed
      oTable.addColumn(new sap.ui.table.Column({
```

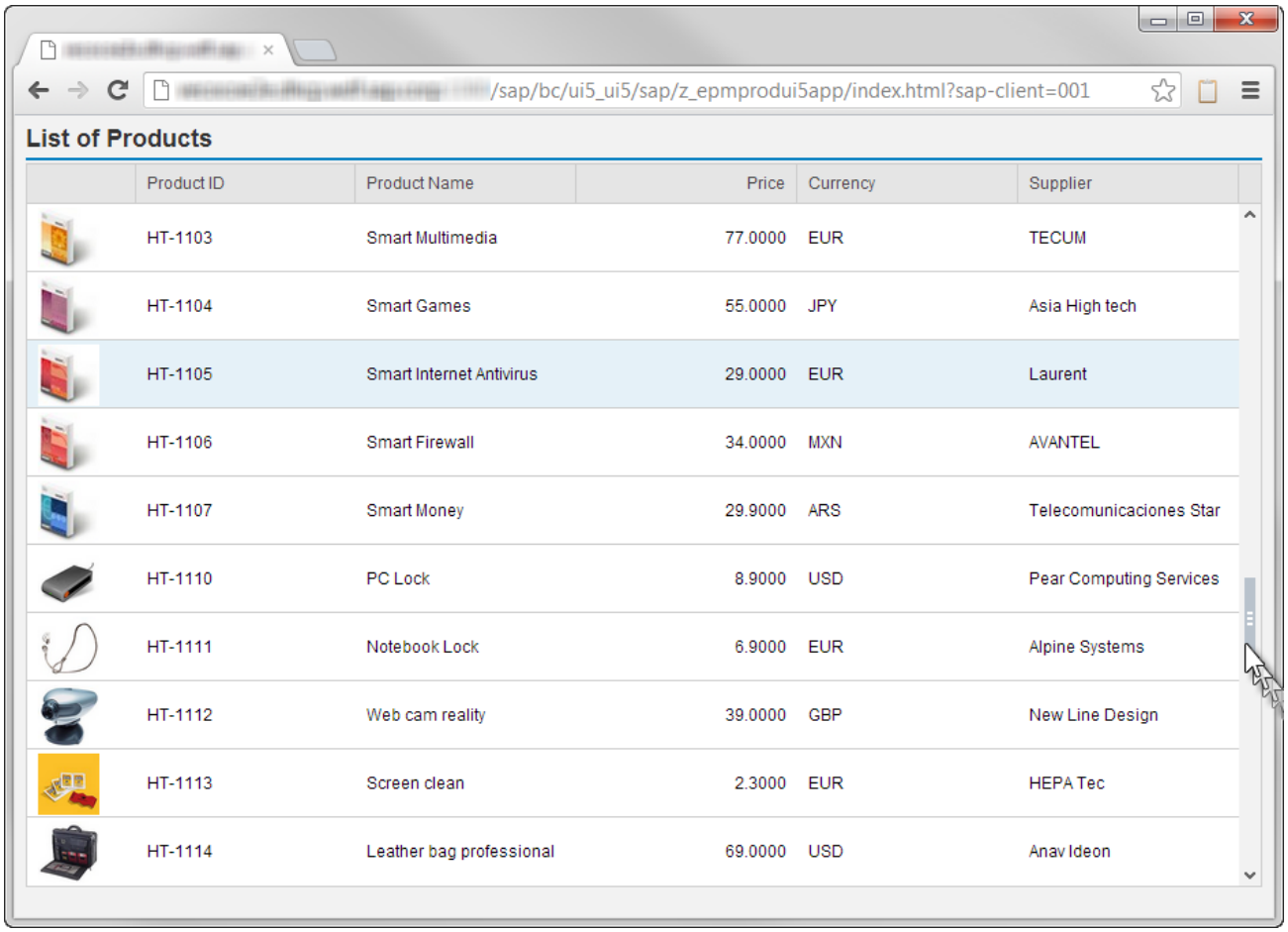
```
        width : "80px",
        flexible : false,
        template : new sap.ui.commons.Image({
      height : "45px",
      src : "{ProductPicUrl}",

      // register function for press event on the image
      press : function(oEvent) {

          // initialize popup content
          oDialog.destroyContent();

          // set the binding context from image
          // NOTE: No oDialog.setModel() needed here, as SAPUI5
          // will bind controls to its default model. The default
          // model instance was assigned in the view controller's
          // onInit() hook method
          oDialog.setBindingContext(oEvent.oSource
                  .getBindingContext());

          // create popup content
          oDialog.bindProperty("title", "ProductID");
          oDialog.addContent(new sap.ui.commons.layout.BorderLayout({
             height : "220px",
             top : {
                 contentAlign : "center",
                 size : "155px",
                 content : new sap.ui.commons.Image({
                    height : "150px",
                    src : "{ProductPicUrl}"
                 })
             },
             center : {
                 contentAlign : "center",
                 content : new sap.ui.commons.TextView({
                    text : "{Description}"
                 })
             }
          }));

          // open popup
          if (!oDialog.isOpen()) {
             oDialog.open();
          }
      }
        })
    }));

    oTable.addColumn(new sap.ui.table.Column({
        label : new sap.ui.commons.Label({
      text : "Product ID"
        }),
        template : new sap.ui.commons.TextView({
      text : "{ProductID}"
        }),
        sortProperty : "ProductID"
    }

    // alternatively:
    ...
```

Binding Context: to display the correct product details for the selected product we need to assign the binding context from the event source (clicked product in table UI) to the dialog control.

Source Code 8: Handling image control events in controller ProductList.controller.js

Starting and Testing the complete SAPUI5 Application

Again these changes have to be submitted to the SAPUI5 ABAP Repository after saving it. Now the application can be started in the browser with the same URL (*http://<hostname>:<port>/sap/bc/ui5_ui5/sap/epmproductsapp/index.html*) as above. The result is shown in Screenshot 32.

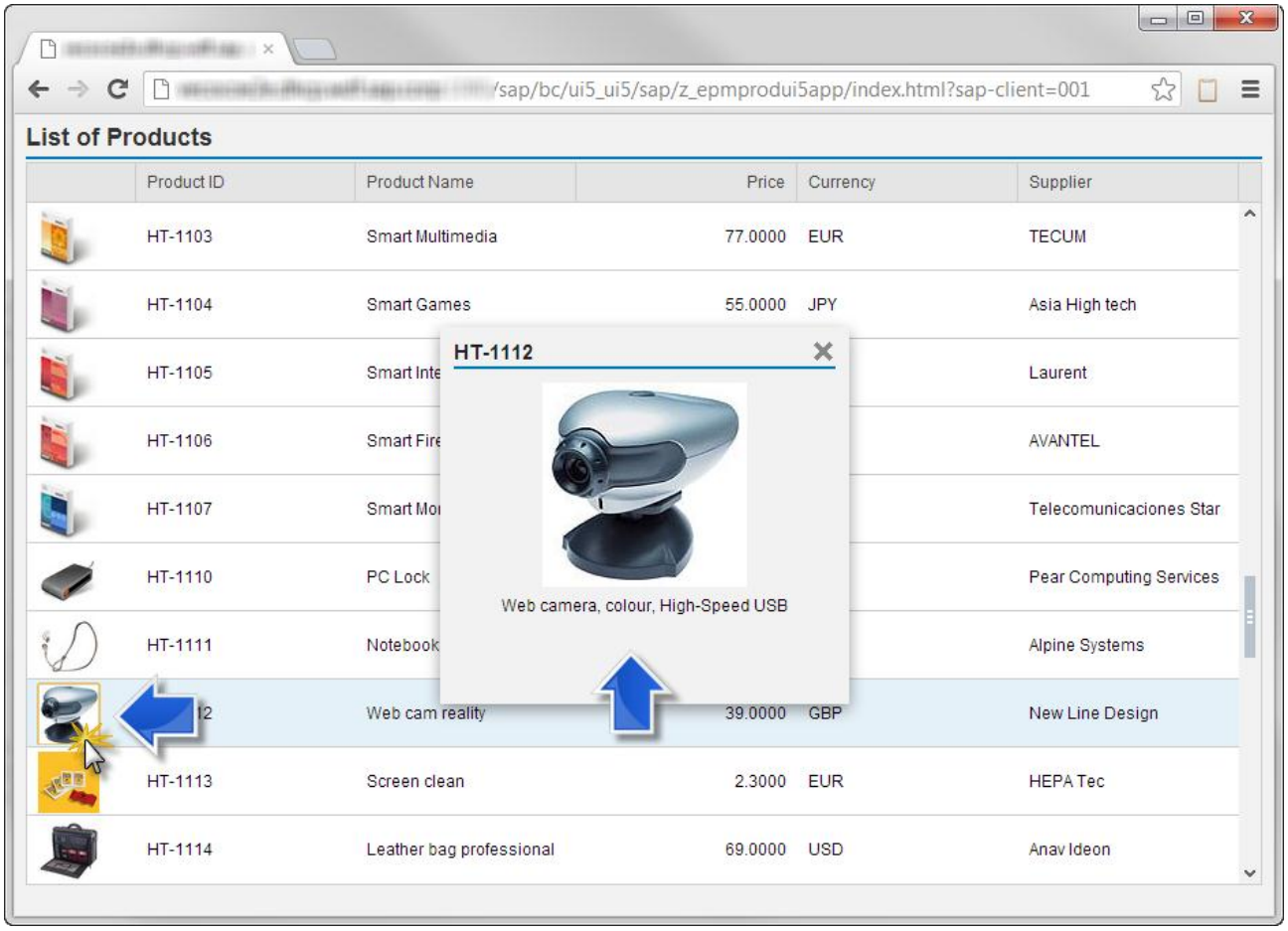


Screenshot 32: SAPUI5 sample application started in Web browser with the URL from the ABAP system

Besides the visible result list of products, this application has already some not directly visible features:

- **Table paging**: the table supports paging (by scrollbar and keyboard) and while doing this it requests missing entries asynchronously from the OData service.
- **Sortable and resizable columns:** Additionally the columns are resizable and can be sorted. The latter one has to be implemented in the OData service, since it may not have all entries on the client available – see Source Code 1 in section *Application Backend / Implementing and Testing own OData Service*.

If the Source Code 8 enhancements for image control event handling were also implemented, it should be possible to click on the image to open a non-modal popup displaying the image with a larger size and a detailed product description (product id is shown as title) as shown in Screenshot 33.



Screenshot 33: Popup dialog showing product details after clicking an icon in the products table entry

How to comply with the Same-Origin Policy in a Development or Test Scenario

There are different ways of testing the application after modifications:

1. The first one, as already described, is submitting the modified sources with the Team Provider to the SAPUI5 ABAP Repository and test directly with the URL from the application system (see right side of Figure 8).
2. A short preview of the application (without any data!) can be started directly in Eclipse by right mouse click on the project name or the *index.html* page and selecting *Run As / Web App Preview.*
3. A third variant is local testing in Eclipse with a configured Java Server (e.g. Apache Tomcat, see left side of Figure 8).

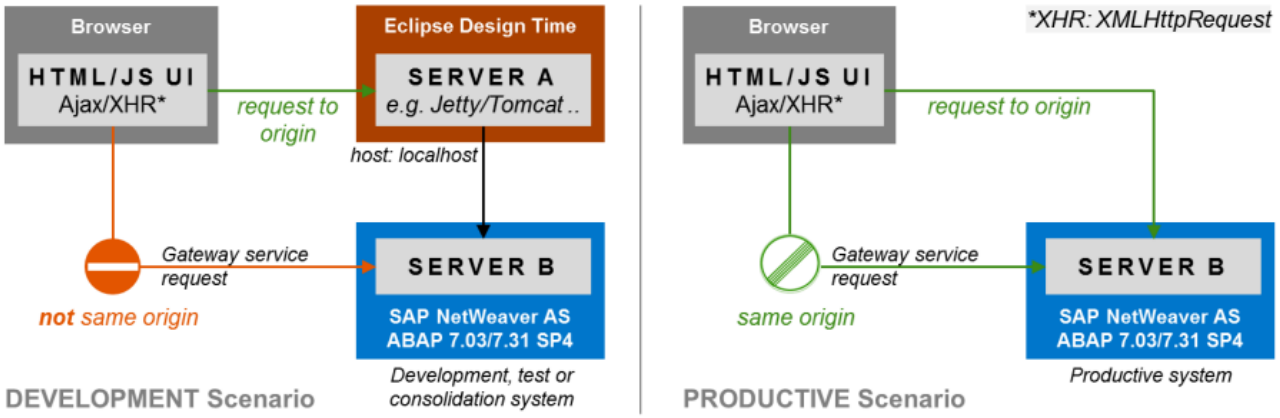


Figure 8: Same-origin policy in development and productive scenarios

But this usually very convenient third method (local testing in Eclipse) conflicts with the same-origin policy of the Web browser, because the HTML/JavaScript is loaded from the locally installed Tomcat, whereas the business data is requested by *XMLHttpRequest* from the application backend (see Figure 8). So a proxy has to be placed in front as an intermediary for requests to both servers. This can be done by using the delivered SAPUI5 *SimpleProxyServlet* (see Figure 9) or by installing an Apache web server and configure it as reverse proxy.

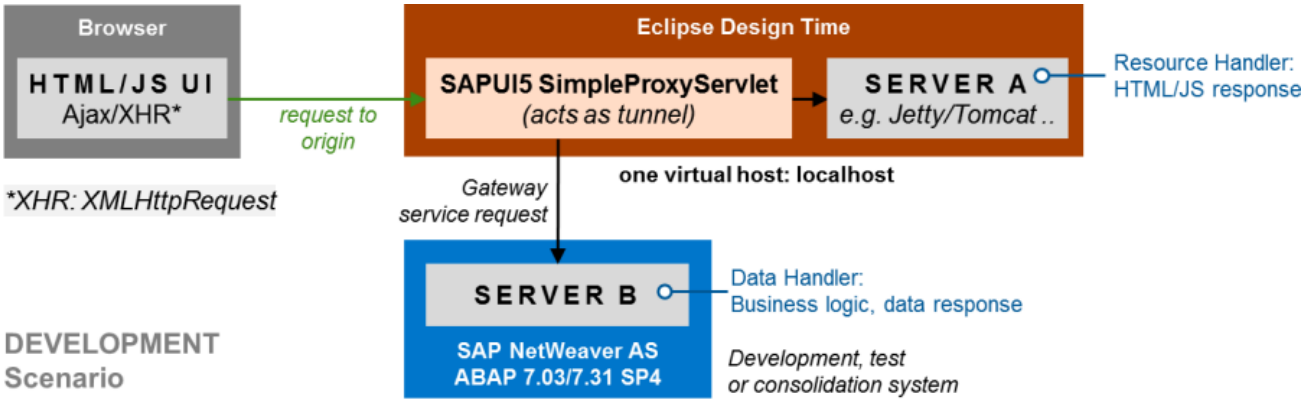


Figure 9: Using the SAPUI5 *SimpleProxyServlet* to comply with same-origin policy in a development scenario

The *SimpleProxyServlet* is restricted to local testing usage only and you configure it in the *web.xml* available in your project under *WebContent / WEB-INF*. There you will find already parts of the configuration, but you have to add the remote location for your data services as a context parameter (highlighted in yellow).

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
                             http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
         version="2.5"
         xmlns="http://java.sun.com/xml/ns/javaee">
  <display-name>EPMProductsApp</display-name>
  <!-- ============================================================ -->
  <!-- UI5 resource servlet used to handle application resources    -->
  <!-- ============================================================ -->
...
  <!-- ============================================================ -->
  <!-- Cache Control Filter to prevent caching of any resource      -->
  <!-- ============================================================ -->
...
  <!-- ============================================================ -->
  <!-- UI5 proxy servlet                                            -->
  <!-- ============================================================ -->
  <servlet>
    <servlet-name>SimpleProxyServlet</servlet-name>
    <servlet-class>com.sap.ui5.proxy.SimpleProxyServlet</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>SimpleProxyServlet</servlet-name>
    <url-pattern>/proxy/*</url-pattern>
  </servlet-mapping>
  <context-param>
    <param-name>com.sap.ui5.proxy.REMOTE_LOCATION</param-name>
    <param-value>http://yourhost:yourport</param-value>
  </context-param>
  <!-- ============================================================ -->
  <!-- Welcome file list                                            -->
  <!-- ============================================================ -->
  <welcome-file-list>
    <welcome-file>index.html</welcome-file>
  </welcome-file-list>
```

```
</web-app>
```

Source Code 9: Proxy configuration in EPMProductsApp/WebContent/WEB-INF/web.xml using the SimpleProxyServlet

Now the service call in the JavaScript code of controller *ProductList.controller.js* has to be adapted to use the proxy instead of a direct connection to the server. Since this code maybe reused several times, it will implemented as function *getUrl(sUrl)* in the view controller:

```
sap.ui.controller("epmproductsapp.ProductList", {

    /**
     * Called when a controller is instantiated and its View controls (if
     * available) are already created. Can be used to modify the View before it
     * is displayed, to bind event handlers and do other one-time
     * initialization.
     */
    // onInit: function() {
       // URL of the OData service - IMPORTANT: relative to the server
       var sServiceUrl = "/sap/opu/odata/sap/z_epm_products/";
       ...
       sap.ui.getCore().setModel(oModel);
    // },

    // onBeforeRendering: function() {
    // },

    // onAfterRendering: function() {
    // },

    // onExit: function() {
    // }

    getUrl : function(sUrl) {
       if (sUrl == "")
           return sUrl;
       if (window.location.hostname == "localhost") {
           return "proxy" + sUrl;
       } else {
           return sUrl;
       }
    }

});
```

Source Code 10: Helper method getUrl() in ProductList view controller to calculate same-origin policy compliant URL

This method has to be used now for the service URL of our OData model created in the view controller's *onInit()* hook, but also for the images (the SAP NetWeaver ESPM OData service delivers just relative URLs for its images).

```
sap.ui.controller("epmproductsapp.ProductList", {
    /**
     * Called when a controller is instantiated and its View controls (if
     * available) are already created. Can be used to modify the View before it
     * is displayed, to bind event handlers and do other one-time
     * initialization.
     */
    onInit : function() {

       // URL of the OData service - IMPORTANT: relative to the server
       var sServiceUrl = this.getUrl("/sap/opu/odata/sap/z_epm_products/");
       ...
    },
});
```

```
sap.ui.jsview("epmproductsapp.ProductList", {

    getControllerName : function() {
       return "epmproductsapp.ProductList";
    },

    createContent : function(oController) {

       ...

       // define the columns, which should be displayed
       oTable.addColumn(new sap.ui.table.Column({
           width : "80px",
           flexible : false,

           template : new sap.ui.commons.Image({
              height : "45px",
              // src:"{ProductPicUrl}",
              src : {
                  path : "ProductPicUrl",
                  formatter : function(value) {
                     return oController.getUrl(value);
                  }
              },

              // register function for press event on the image
              press : function(oEvent) {

                  // initialize popup content
                  oDialog.destroyContent();

                  // link model to popup & set the binding context from image
                  oDialog.setModel(oModel);
                  oDialog.setBindingContext(oEvent.oSource
                         .getBindingContext());

                  // create popup content
                  oDialog.bindProperty("title", "ProductID");
                  oDialog.addContent(new sap.ui.commons.layout.BorderLayout({
                     height : "220px",
                     top : {
                         contentAlign : "center",
                         size : "155px",
                         content : new sap.ui.commons.Image({
                            height : "150px",
                            // src : "{ProductPicUrl}"
                            src : {
                                path : "ProductPicUrl",
                                formatter : function(value) {
                                   return oController.getUrl(value);
                                }
                            }
                         })
                     },
...
```

Source Code 11: Calling the getUrl() helper method to get proxy compliant service and image URLs

Having done this configuration and coding modifications the application can also be tested locally on the Java Server configured in Eclipse before submitting it to the SAPUI5 ABAP Repository.

Note: For more details on how to locally test SAPUI5 applications see document Testing the SAPUI5 Application in Eclipse in the SAPUI5 Demo Kit.

Embedding a SAPUI5 view into a ux3 Shell control

Finally we make our SAPUI5 application UI look more real by embedding it into a ux3 shell control. It is part of the next-generation UX control library of SAPUI5 comprising other controls like *ExactBrowser, FacetFilter, OverlayContainer, ThingInspector, QuickView* and *NavigationBar* (for more details see documentation)

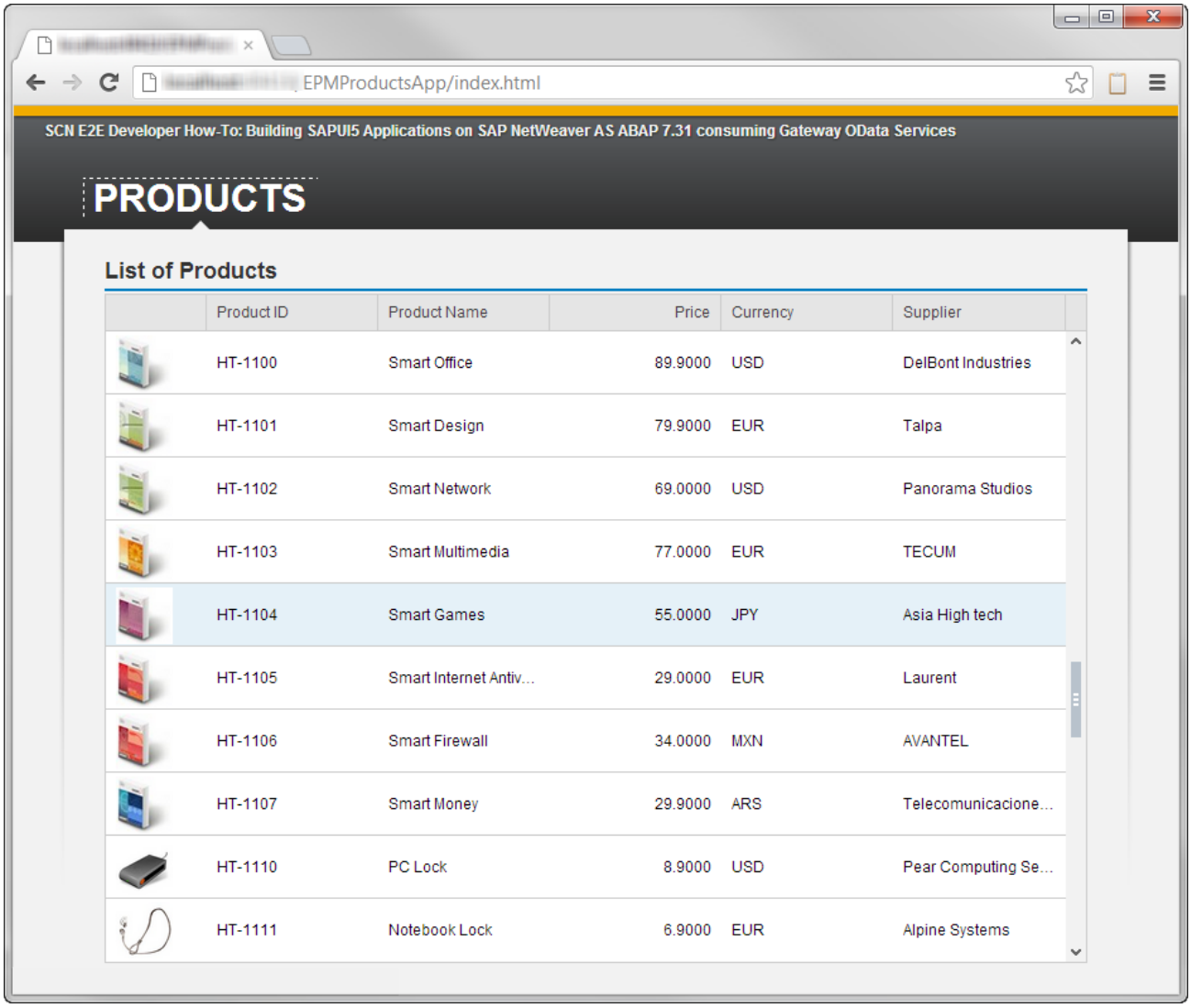
In the index.html page the ux3 shell control will be added and the products list view will be embedded in the shell:

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

<script src="resources/sap-ui-core.js" id="sap-ui-bootstrap"
       data-sap-ui-libs="sap.ui.commons, sap.ui.ux3"
       data-sap-ui-theme="sap_goldreflection">

</script>
<!-- add sap.ui.table,sap.ui.ux3 and/or other libraries to 'data-sap-ui-libs' if required -->

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

    // ************************************************************************
    // Create the ux3 shell
    // ************************************************************************
    var oShell = new sap.ui.ux3.Shell({
     appTitle : "SCN E2E Developer How-To: Building SAPUI5 Applications on SAP NetWeaver AS ABAP
7.31 consuming Gateway OData Services",
     showFeederTool : false,
     showInspectorTool : false,
     showSearchTool : false,
     showLogoutButton : false,
     worksetItems : [ new sap.ui.ux3.NavigationItem({
     text : "Products"
     }) ]
     });

    // ************************************************************************
    // Instantiate view 'epmproductsApp.ProductList.view.js' and add it to ux3 shell
    // ************************************************************************
    var view = sap.ui.view({
       id : "idProductList1",
       viewName : "epmproductsapp.ProductList",
       type : sap.ui.core.mvc.ViewType.JS
    });

    // set the initial content of the Shell
    oShell.setContent(view);

    // place the Shell into the <div> element defined below
    oShell.placeAt("content");
    //view.placeAt("content");
</script>

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

to use the ux3 shell control import the related sap.ui.ux3 library in the

embed product list view into the shell content

Source Code 12: Embedding the product list view into a ux3 shell control within the index.html page



Screenshot 34: Final SAPUI5 application UI with list of products displayed in ux3 shell

You have now successfully completed a basic SAPUI5 application that is hosted from within an SAP NetWeaver AS ABAP system consuming a local Gateway OData service.

Related Content

SAP NetWeaver Gateway

SAP NetWeaver Gateway Community, SCN space

SAP NetWeaver Gateway Product Documentation, SAP Help, e.g. the Gateway Security Guide or Gateway Service Builder

SAP NetWeaver Gateway - How-To Guides, SCN doc

SAP Technical Brief – SAP NetWeaver Gateway, web page, Chris Whealy, SAP AG

The new SAP NetWeaver Gateway Service Builder: Build new OData Services in 3 Quick Steps, SCN blog, Thomas Meigen, SAP AG, Sept 2012

Rebooting your User Interfaces with SAP NetWeaver Gateway, SCN blog from SAP Mentor John Moy, October 2012

Take Advantage of Cross-Platform, Cross-Device Access While Keeping Your Data Secure with SAP NetWeaver Gateway , SAP Insider article, Genady Podgaetsky, André Fischer, SAP AG, July 2012

SAP Note explaining issues when using Remote Gateway with HTML5 apps in ABAP

SAP Web Dispatcher (Remote Proxy Solution from SAP), SAP Online Help, UI add-on for SAP NetWeaver

GW100: SAP NetWeaver Gateway - Building OData Services, SAP Training, 3 days classroom training

UI Development Toolkit for HTML5

UI Development Toolkit for HTML5 Developer Center, SCN space for SAPUI5

https://sapui5.netweaver.ondemand.com/sdk/#content/Overview.html , 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

UI add-on for SAP NetWeaver

UI Add-On for SAP NetWeaver Product Documentation: SAP Help, Master Guide, Installation Guide, Security Guide, Developers Guide, Admin Guide. Contains also References to Gateway and WebDispatcher

UI Add-On for SAP NetWeaver - Central Note, SAP note 1759682, central note for the UI add-on for SAP NetWeaver with up-to-date informtion

Introducing the new UI Add-On for SAP NetWeaver, SCN blog, Filip Misovski, SAP AG, September 2012

Copyright