

Browse the Book

In this chapter, you'll learn about SAP S/4HANA's application programming model and its associated technologies.



"ABAP Programming Model for SAP Fiori"



Table of Contents



Index



The Authors

Stefan Haas, Bince Mathew

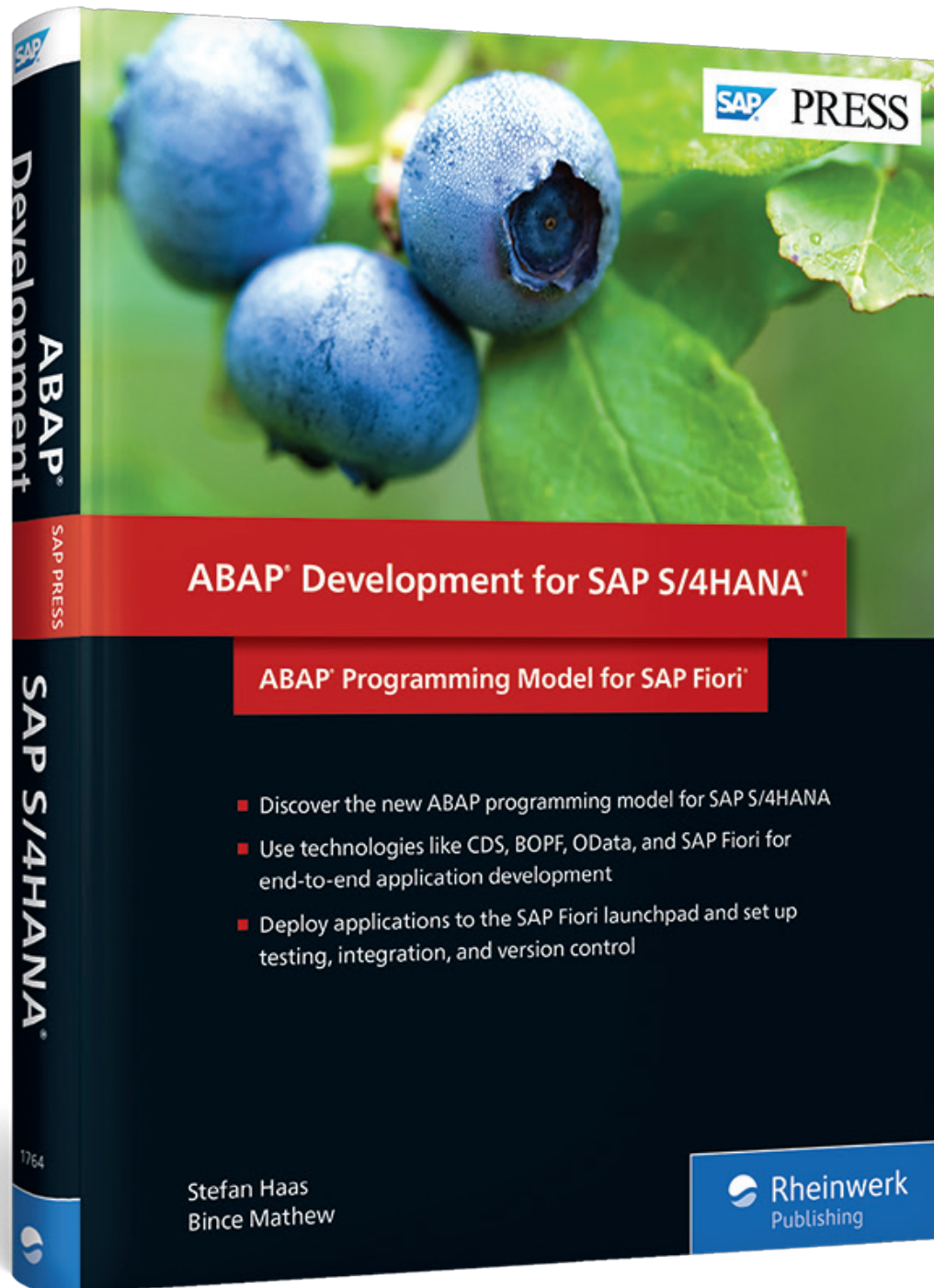
ABAP Development for SAP S/4HANA: ABAP Programming Model for SAP Fiori

461 Pages, 2018, \$79.95

ISBN 978-1-4932-1764-9



www.sap-press.com/4766



Chapter 3

ABAP Programing Model for SAP Fiori

This chapter introduces the ABAP programing model for SAP Fiori, which has been available since SAP S/4HANA 1610, SPS 03, or SAP NetWeaver Application Server for ABAP 7.52, SP 02. It's the standard development model for new SAP S/4HANA applications and reflects the SAP S/4HANA core architecture described in Chapter 1.

In general, the model supports the development of SAP HANA optimized OData services for SAP Fiori applications based on Core Data Services (CDS) views and covers analytical, transactional, and search application scenarios. Two main scenarios can be distinguished when developing applications using the programing model: read-only applications and transactional applications. Read-only applications only require an underlying CDS data model and application-specific analytics or search annotations. The CDS data model and its annotations are then exposed as an OData service using the Service Adaptation Description Language (SADL) technology.

Transactional applications, in addition to read-only applications, require the generation of a Business Object Processing Framework (BOPF) business object for handling create, update, and delete operations as well as additional business logic implemented via BOPF actions, validations, and determinations. In the following, we'll go through the different technologies associated with the ABAP programing model.

3.1 Core Data Services

Core Data Services (CDS) are the foundation of all SAP S/4HANA application types. They are deployed on top of legacy or new SAP ERP tables and enable the development of semantically rich data models that foster code pushdown to the SAP HANA database. They are developed on the ABAP stack and therefore use the standard lifecycle management of ABAP development objects; for instance, they are transported between systems using the standard ABAP Change and Transport System (CTS).

On activation of a CDS view, two artifacts are generated: a corresponding Data Dictionary (DDIC) view in the ABAP Dictionary (@AbapCatalog.sqlViewName) and a SAP HANA view on the database. Only the CDS view definition is transported and has the object repository entry R3TR DDLS <CDS_DEFINITION_VIEW_NAME>.

To cover different application scenarios, CDS views, which are defined using the data definition language (DDL) of CDS, can be enhanced using different types of annotations.

■ Analytical annotations

To use a CDS view as a data cube or query within analytical application scenarios via the Analytical Engine, it must be annotated using @Analytics annotations.

■ UI annotations

CDS views can be annotated with user interface (UI) annotations (@UI) to define where certain entities, fields, and data will be placed within a SAP Fiori elements template application, which reduces the required JavaScript SAPUI5 frontend code drastically. UI annotations can be moved to a metadata extension file with object repository type R3TR DDLX in order not to clutter the core CDS view with loads of UI annotations.

■ Search annotations

CDS views can be configured for search scenarios using @Search annotations, for instance, as an Enterprise Search (ESH) model for the SAP Fiori launchpad search or for SAP Fiori in-app search by defining the SAP HANA text search scope and fuzziness.

■ Transactional annotations

To enable transactional processing capabilities (create, write, delete) in addition to analytical or search capabilities, a BOPF object can be generated for a CDS entity by providing transactional @ObjectModel annotations.

Figure 3.1 shows an overview of the ABAP programming model development flow and involved artifacts. As you can see, CDS is in the center of the development flow and, for instance, the basis of the OData service, which is, in turn, consumed by the SAP Fiori app or the BOPF business object for transactional processing.

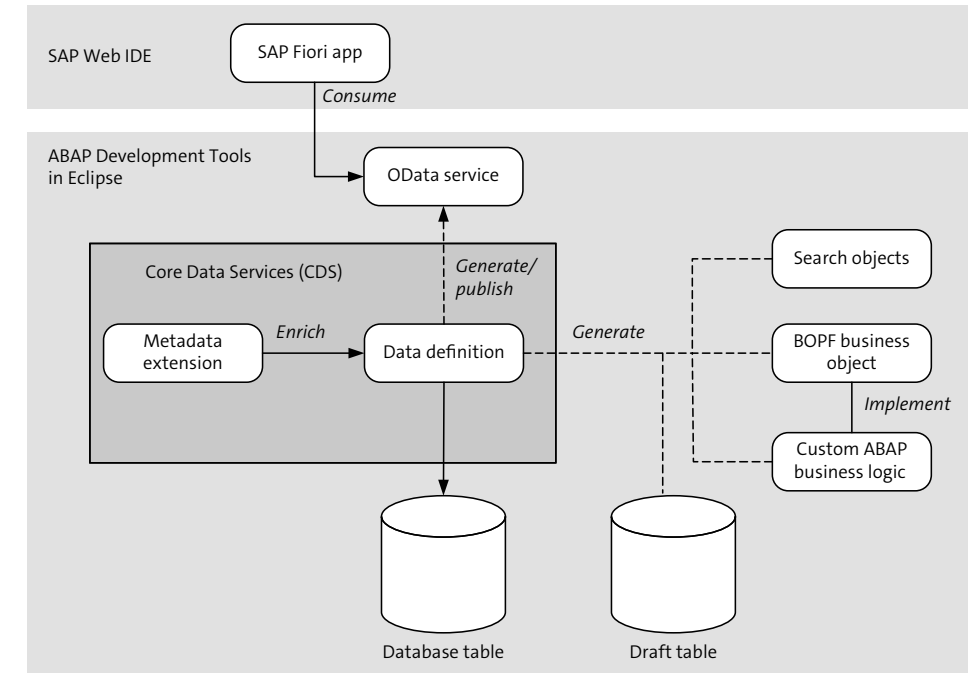


Figure 3.1 Overview of the ABAP Programming Model Development Flow and Its Artifacts

3.2 SAP Gateway

SAP Gateway plays a crucial role in providing an easy-to-use non-ABAP-based access to business data stored in backend SAP NetWeaver systems. Access to business data is granted via REST-based OData services using HTTP as the underlying data transfer protocol.

As of SAP NetWeaver version 7.40, the software component SAP_GWFND is installed as part of the SAP NetWeaver standard and includes the full scope of functionality required for hub and backend enablement. In general, from an architectural perspective, there are two deployment approaches: embedded deployment or hub deployment. The hub deployment can be further split up into development on the hub or development in the backend system. The usual setup for an SAP S/4HANA on-premise system is shown in Figure 3.2. Usually, development takes place in the ABAP backend system, and a hub system will be introduced to handle the additional load of

client OData requests to the SAP Gateway component. This setup requires an additional SAP NetWeaver system functioning as the SAP Gateway hub system or front-end server and a trusted remote function call (RFC) connection between the hub and backend system to forward requests from the frontend to the backend system, which contains the business logic and the data. The RFC-enabled function module passing data from the frontend to the backend is `/IWBEP/FM_MGW_HANDLE_REQUEST`.

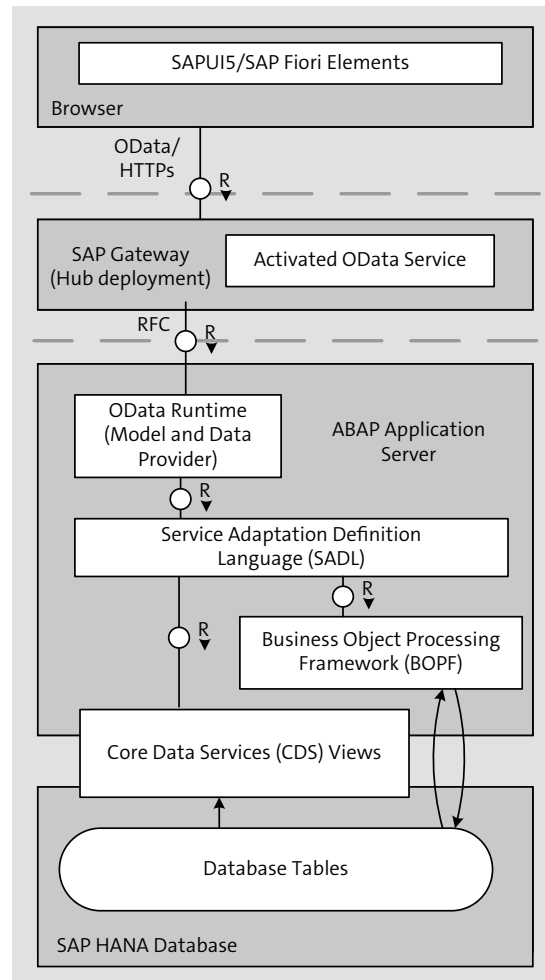


Figure 3.2 High-Level Overview of a Typical SAP S/4HANA System Setup with SAP Gateway Provided as a Separate Hub System

The SAP Fiori app running in the browser sends OData HTTP GET, POST, DELETE, or PUT requests to the SAP Gateway system, which exposes all registered and activated OData services. The SAP Gateway passes the incoming OData requests to the backend via a trusted RFC connection. The OData runtime in the backend then delegates the actual data selection to the SADL framework layer. If the request is a read-only GET request, SADL will delegate the request to its query engine, which will generate an SQL SELECT statement to select the requested OData entity's data from the database tables via its corresponding CDS view. In case of a write access, for instance, POST for create or PUT for update, the request will be delegated to the transactional BOPF runtime, which will handle the database updates according to the provided transactional annotations specified in the CDS view.

3.3 OData

OData is a REST-based data protocol used for transferring business data as well as metadata between the ABAP backend system and client applications via the SAP Gateway hub system. In SAP S/4HANA, client applications of OData services usually are SAP Fiori SAPUI5 applications running in the local browsers of end-user devices such as desktop PCs or tablets. Together with the SAP Gateway, OData provides access to the SAP backend business data in an easy-to-understand and well-defined way using HTTP as its data transfer protocol.

3.3.1 Overview

An OData service organizes data in the form of entities that have a set of properties interconnected via associations. These elements resemble the elements of CDS data models, so CDS data models are the perfect candidates for exposure as OData services.

The structure of an OData service can be explored by looking at its service document and its service metadata document. The service document contains a list of entities or resources that can be accessed using this service and can be requested via `/sap/opu/odata/sap/<OData_service_name>/`. Additionally, you can see whether the service allows creating, changing, or deleting entities by looking at the `sap:creatable`, `sap:updatable`, and `sap:deletable` attributes of the `<app:collection>` tag, which also contains a relative reference to the entity set via its `href` attribute (see Figure 3.3).

```
<app:service xml:lang="en"
xml:base="https://ldciuyt.wdf.sap.corp:44300/sap/opu/odata/sap/Z_PurchasingDocumentDDL_CDS/"
xmlns:app="http://www.w3.org/2007/app" xmlns:atom="http://www.w3.org/2005/Atom"
xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata"
xmlns:sap="http://www.sap.com/Protocols/SAPData">
  <app:workspace>
    <atom:title type="text">Data</atom:title>
    <app:collection sap:creatable="false" sap:updatable="false" sap:deletable="false" sap:content-
version="1" href="Z_PurchasingDocumentDDL">
      <atom:title type="text">Z_PurchasingDocumentDDL</atom:title>
      <sap:member-title>Purchase Document</sap:member-title>
    </app:collection>
    <app:collection sap:creatable="false" sap:updatable="false" sap:deletable="false" sap:content-
version="1" href="Z_PurchasingDocumentItemDDL">
      <atom:title type="text">Z_PurchasingDocumentItemDDL</atom:title>
      <sap:member-title>Purchase Document Item</sap:member-title>
    </app:collection>
  </app:workspace>
  <atom:link rel="self"
href="https://ldciuyt.wdf.sap.corp:44300/sap/opu/odata/sap/Z_PurchasingDocumentDDL_CDS/">
  <atom:link rel="latest-version"
href="https://ldciuyt.wdf.sap.corp:44300/sap/opu/odata/sap/Z_PurchasingDocumentDDL_CDS/">
</app:service>
```

Figure 3.3 Service Document of a Simple Purchase Document and Purchase Document Item OData Service

The service metadata document is a lot more detailed than the service document and shows all metadata of the service. It can be requested using the \$metadata option: /sap/opu/odata/sap/<OData_service_name>/\$metadata. It displays all entities of the service, including their properties and associations.

OData uses the REST HTTP commands POST, GET, PUT, and DELETE for creating, reading, updating, and deleting (CRUD) entities. Additionally, OData defines a simple but powerful query language for restricting the result set provided by the SAP Gateway. Table 3.1 lists the most common OData query options.

Operation	Query Option
Filtering	\$filter, e.g., \$filter = PurchaseDocument eq '0005'
Projecting or selecting properties	\$select, e.g., \$select = PurchaseDocument, PurchasingOrganization,...
Sorting	\$orderby, e.g., \$orderby = CreationDate desc
Client-side paging	\$top and \$skip, e.g., \$top = 10&\$skip = 0

Table 3.1 Most Important OData Query Options

Operation	Query Option
Counting	\$count
Formatting	\$format, e.g., \$format = JSON

Table 3.1 Most Important OData Query Options (Cont.)

3.3.2 OData Service Creation

As you already know, you can directly select data from a CDS view using Open SQL in ABAP, as you would select data from a conventional database table. However, SAP Fiori apps require the exposure of data as an OData service using HTTP as the data transfer protocol.

Currently, a CDS data model can be exposed as an OData service in three different ways: Auto-Exposure, Reference Data Source (RDS), or Mapped Data Source (MDS).

Auto-Exposure

The simplest, but also the most restricted, way of exposing a CDS model as an OData service is by using the @OData.publish:true header annotation (Listing 3.1).

```
@AbapCatalog.sqlViewName: 'SQL_VIEW_NAME'
...
@OData.publish: true
define view CDS_VIEW_NAME as select from ...
```

Listing 3.1 @OData.publish:true Header Annotation to Expose a Simple Coherent CDS Data Model as an OData Service

When activating the view in the ABAP development tools in Eclipse, several artifacts will be generated in the system (Figure 3.4):

- The actual SAP Gateway Business Suite Enablement - Service object named <CDS_VIEW_NAME>_CDS (object type: R3TR IWSV)
- An SAP Gateway model named <CDS_VIEW_NAME>_MDL (object type: R3TR IWMO)
- An annotation model named <CDS_VIEW_NAME>_CDS_VAN (object type: R3TR IWVB)

By default, the generated OData service won't be activated. This must be done manually by accessing Transaction /IWFND/MAINT_SERVICE on the SAP Gateway hub system.

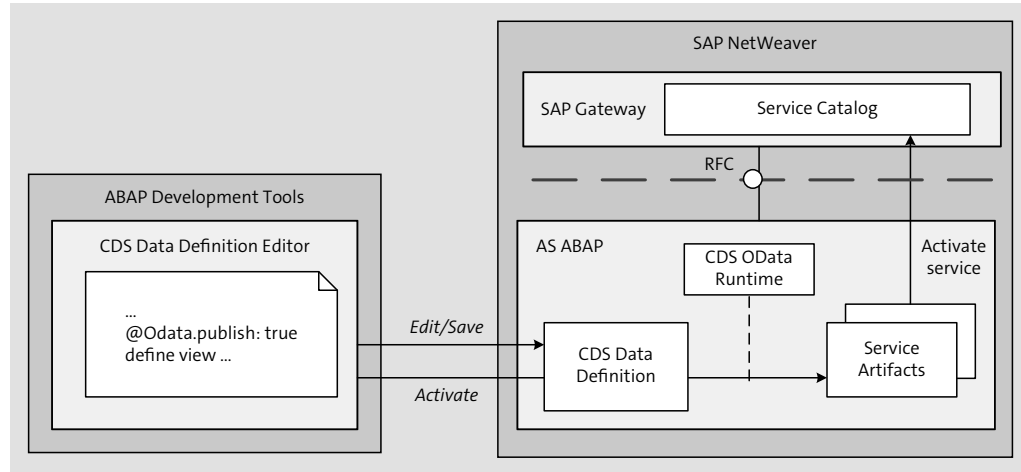


Figure 3.4 Components and Activities Involved When Exposing a CDS Data Model as an OData Service Using the Auto-Exposure Method

The Auto-Exposure OData service creation method can only be used for simple coherent CDS models that consist of a root node and first-level-associated views. It's not possible to add additional non-first-level-associated or unrelated CDS views as OData entities to the service; for instance, if you have a coherent model of a purchase document root view that has an association to a purchase document item child view, and the item view in turn has an association to a material view, the material entity can't be part of the service because it's not a first-level association of the root node (Figure 3.5). If this is required, one of the other approaches must be used.

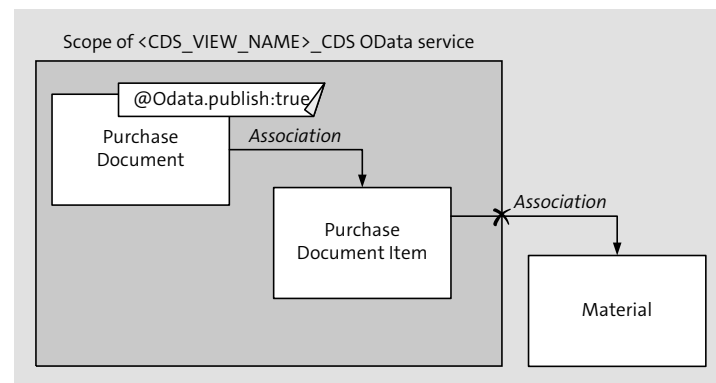


Figure 3.5 Scope of the OData Auto-Exposure Service Creation

Referenced Data Source

For complex CDS view models that need to be exposed as OData entities within a common OData service, the Reference Data Source (RDS) approach is the right choice. A CDS model can be considered as complex in this case when it has several levels of associations or unrelated root views. The RDS approach, however, requires the creation of an SAP Gateway Service Builder project. This can be done by carrying out the following steps:

1. Open Transaction SEGW.
2. Choose the **Create Project** button in the toolbar (Figure 3.6).

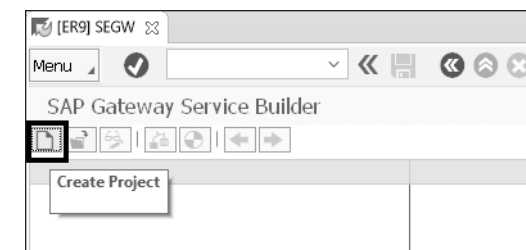


Figure 3.6 Creating a New SAP Gateway Service Builder Project

3. In the **Create Project** dialog that appears (Figure 3.7), provide a description and name for the SAP Gateway project that will become a transportable ABAP repository entry (R3TR IWPR Z_PURCHASING_DEMO). In this entry, IWPR stands for *Gateway Business Enablement – Service Builder Project*. Choose **Continue**.

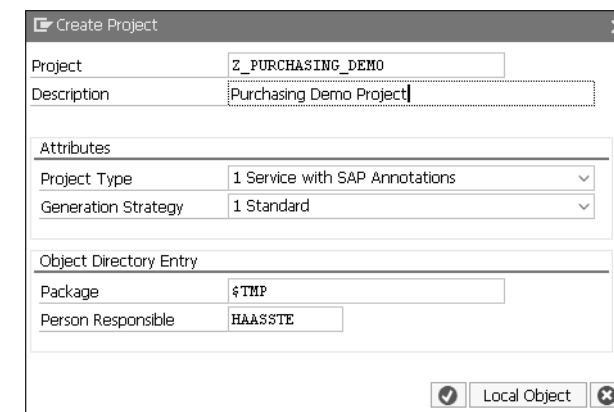


Figure 3.7 SAP Gateway Service Builder: Create Project Dialog

After the project has been created, you can start creating the actual OData service. First, you must create a data model. For the RDS scenario, we'll create a data model by referencing a CDS model. Therefore, choose **Reference • Data Source** from the context menu of the **Data Model** folder (Figure 3.8).

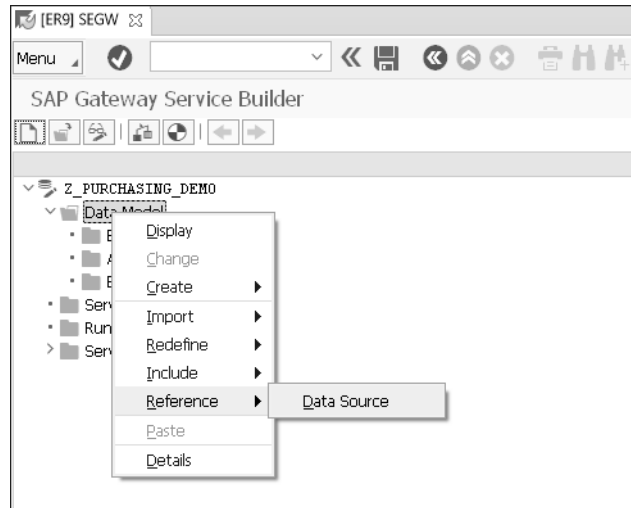


Figure 3.8 Creating a Data Model by Referencing a CDS Data Source

This will trigger the **Reference Data Source Wizard** (Figure 3.9). You must enter a CDS view you want to expose as an OData entity within your OData service, usually the root view or one of the root views of your CDS model.



Figure 3.9 Entering the CDS Root View of the Referenced CDS Model

In the next step of the wizard (Figure 3.10), you can select the associated views of the previously selected CDS view and make them part of your OData service model definition.

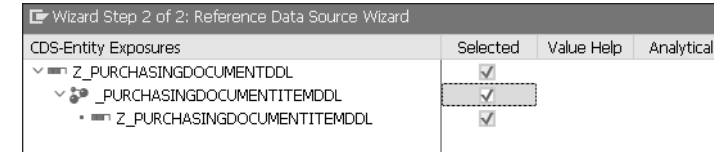


Figure 3.10 Selecting Associations of the CDS Root View

When you finish the wizard, you'll see a new folder called **Data Source References** Folder. If you open its subfolder, **Exposures via SADL** and select the **CDS-Entity Exposures** node, you'll see the previously selected CDS entities that will be exposed as OData entities (Figure 3.11).

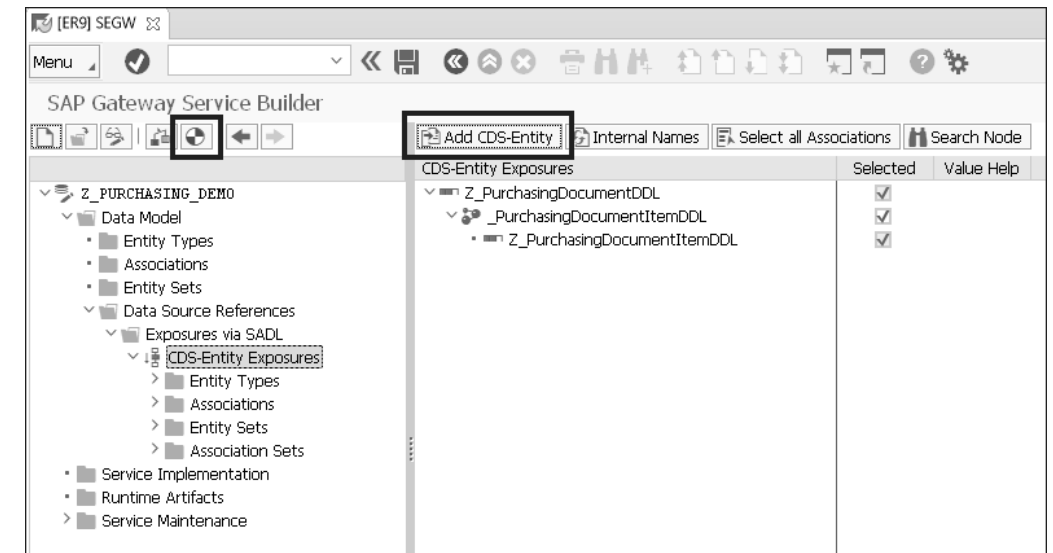


Figure 3.11 The Exposed CDS Entities of the Service Builder Project

You can also add any additional CDS entities using the **Add CDS-Entity** button, which makes the RDS scenario a lot more flexible than Auto-Exposure. Finally, the OData service can be generated using the **Generate Runtime Objects** toolbar button, which will generate the runtime artifacts of the service and register them in the backend system.

The first time the service is generated, the **Model And Service Definition** dialog will appear so you can edit the default names of the objects to be created (Figure 3.12). As

you can see, the RDS scenario will also generate a model provider class (MPC) and a data provider class (DPC), including the respective extension subclasses with the suffixes MPC_EXT and DPC_EXT. The MPC provides the runtime representation of our referenced CDS data model in the form of the service metadata document. The DPC provides the actual entity data by delegating the incoming requests to the generic SADL engine, which will transparently select data from CDS views and map them to OData entities.

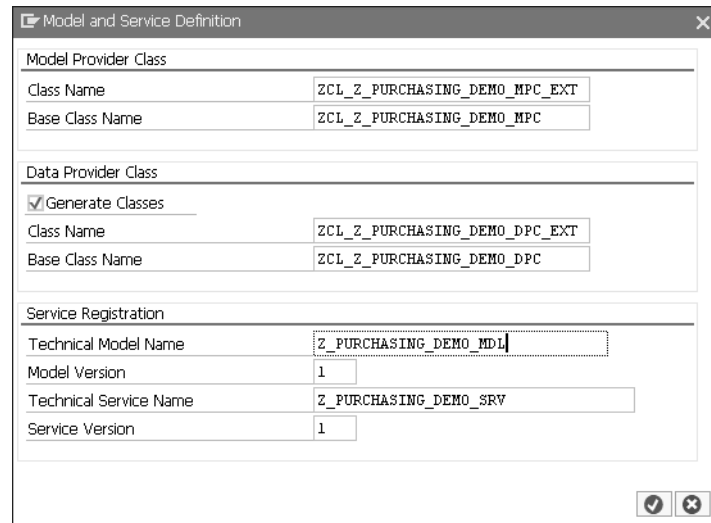


Figure 3.12 Model and Service Definition Dialog

With the RDS scenario, you have a lot of flexibility over the service creation process and implementation. You can even extend the MPC_EXT class to provide additional annotations or implement additional logic in the DPC_EXT class. However, adding non-CDS entities to the service isn't recommended.

Mapped Data Source

The Mapped Data Source (MDS) scenario is only used if one of the previous scenarios isn't applicable, for instance, if the service will contain non-CDS OData entities. Creating an OData service using the MDS scenario requires more manual work than the RDS and Auto-Exposure approaches. As with the RDS approach, you must first create a SAP Gateway Service Builder project using Transaction SEGW. Then, you must define a data model either manually or, in the case of CDS views, by importing the

ABAP SQL view name of the CDS view as a DDIC structure. You can do this by choosing **Import • DDIC Structure** from the context menu of the **Data Model** folder (Figure 3.13). Then, under the **Service Implementation** folder, select the entity, and choose **Map to Data Source**.

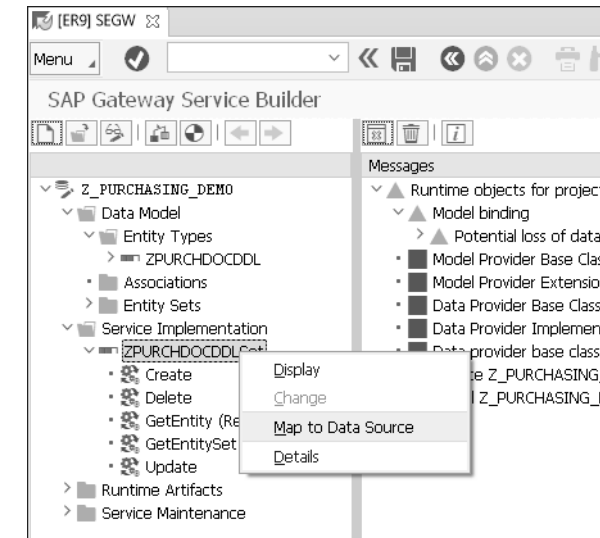


Figure 3.13 Mapping the OData Entity to a Data Source

Select a CDS view to which you want to map the OData entity (Figure 3.14). This mapping will enable model-based data selection via the SADL engine, and you don't have to code the OData entity CRUD methods manually.

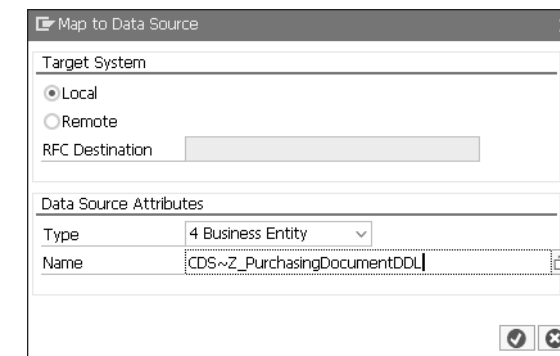


Figure 3.14 Mapping an OData Entity to a CDS Business Entity

Finally, you must manually map the fields of the OData entity to the fields of the CDS entity by dragging fields from the **SADL Model** pane on the right side of the screen to the mapping table in the middle (Figure 3.15).

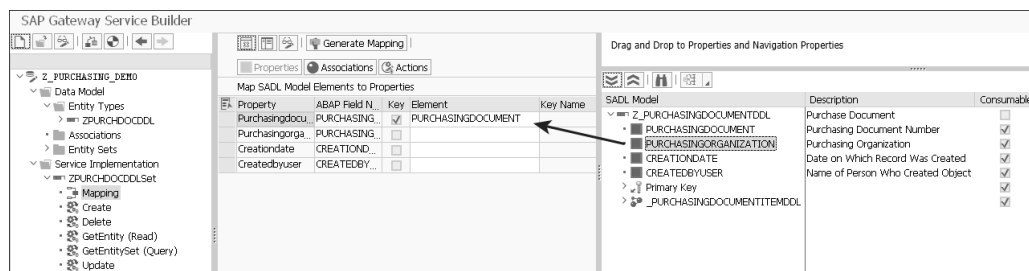


Figure 3.15 Mapping Fields of the CDS Entity to Fields of the OData Entity

In addition to the mapped data source entities, you can also manually create entities and implement their CRUD methods in the `DPC_EXT` class generated on the first generation of the service. As you can see, the MDS approach requires a bit more manual work than the previous options but provides you with full flexibility over the service creation. It supports generic data access to SADL CDS models as well as the manual ABAP-based implementation of OData entities.

Activating a Service on the SAP Gateway Hub System

One step we omitted for all three scenarios was the activation of the service in the Service Catalog of the SAP Gateway hub system, which we'll show now. Transaction `/IWFND/MAINT_SERVICE` is used to maintain all registered services, as well as to register and activate new services. It's the central entry point for dealing with OData services. To add a new service, you must click the **Add Service** button (Figure 3.16 ❶). This will lead you to another application for adding not yet registered backend services to the Service Catalog of the SAP Gateway system ❷. To find a certain service in a backend system, you must provide the system alias of the backend system in which the service has been created as well as the technical service name; for instance, in the Auto-Exposure case, this is `<CDS_VIEW_NAME>_CDS`. Finally, you can add the service to the Service Catalog by selecting the corresponding line and clicking the **Add Selected Services** button ❸.

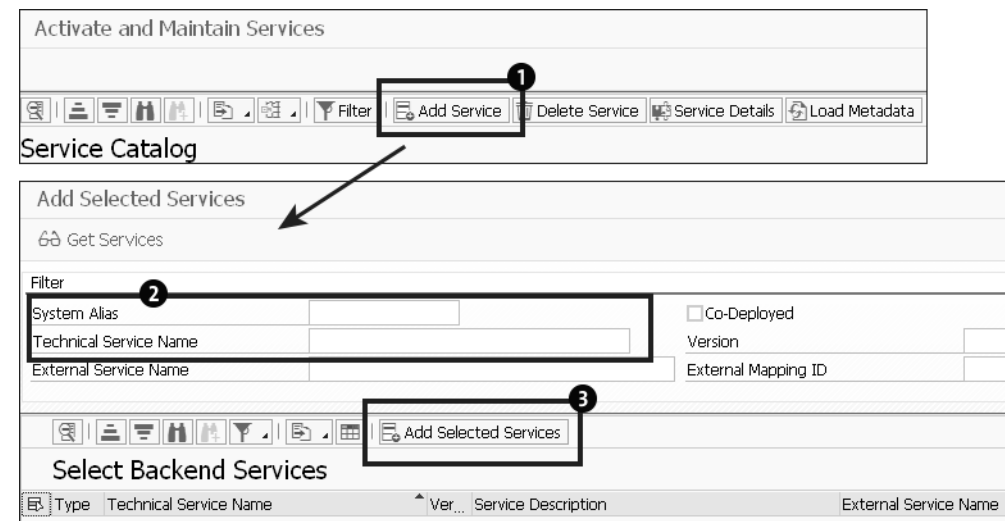


Figure 3.16 Adding a New OData Service to the Service Catalog

3.4 Service Adaptation Description Language

One of the core technologies enabling the ABAP programming model for SAP Fiori is the Service Adaptation Description Language (SADL). It provides two main functionalities in this context:

- *At design time*, SADL supports the model-based creation of an OData service based on a CDS data model.
- *At runtime*, the OData runtime delegates incoming requests for OData entities to the SADL adapter, which dynamically generates SQL `SELECT` statements to select the business data via CDS views or further delegates the requests to the transactional BOPF runtime if write access is required.

With this, SADL supports two of the three phases required for developing an OData service:

- Data model definition (supported by SADL)
- Service implementation (supported by SADL)
- Service maintenance

3.4.1 Data Model Definition Phase

The aim of this OData service development phase is to provide potential clients of the service with information about the service's shape, for instance, the structure of its entities, their relationships, and the supported operations. This information is provided in the form of a service metadata document in Extensible Markup Language (XML). In the data model definition phase, SADL creates a mapping of CDS entities to an OData Entity Data Model (EDM) and populates the MPC of the OData implementation with information derived from the CDS model structure. Additionally, CDS view field data types are mapped to EDM primitive data types; for instance, the ABAP dictionary built-in type DATS is mapped to the EDM.DateTime data type. Moreover, if transactional processing is enabled for CDS entities via the creation of a BOPF business object, BOPF actions will appear as OData function imports in the service metadata document. Table 3.2 provides an overview of the mappings from CDS concepts to OData concepts performed by SADL.

CDS	OData
CDS view (DDL)	EDM entity
CDS associations	EDM association
CDS view field data type	EDM primitive data type
BOPF actions	Function import

Table 3.2 Mappings between CDS Data Model Concepts and OData EDM Concepts

Figure 3.17 shows how the service metadata document will look for the simple scenario introduced in Chapter 1. As you remember, the purchase document CDS view (Z_PurchasingDocumentDDL) had an association to the purchase document items view (Z_PurchasingDocumentItemDDL). Both views now appear as OData entity types in the OData service's metadata document, including the information about the association between the two entities and their cardinalities. Which operations an entity supports is indicated by the `sap:creatable`, `sap:updatable`, and `sap:deletable` attributes of the EntitySet element. In the following example, you can see that the service doesn't support any write access at all, as we haven't enabled the transactional runtime for our CDS model by generating a BOPF business object. Thus, the service only supports read access via HTTP GET entity requests. As already mentioned, the metadata file of an OData service can be requested using the `$metadata` path, `/sap/opu/odata/sap/<OData_service_name>/$metadata`.

```
<EntityType sap:content-version="1" sap:label="Purchasing Document" Name="Z_PurchasingDocumentDDLType">
  <Key>
    <PropertyRef Name="PurchasingDocument"/>
  </Key>
  <Property sap:label="Purchasing document" Name="PurchasingDocument" sap:quickinfo="Purchasing Document Number" sap:display-format="UpperCase" MaxLength="10" Nullable="false" Type="Edm.String"/>
  <Property sap:label="Purch. Organization" Name="PurchasingOrganization" sap:quickinfo="Purchasing Organization" sap:display-format="UpperCase" MaxLength="4" Type="Edm.String"/>
  <Property sap:label="Created On" Name="CreationDate" sap:quickinfo="Date on Which Record Was Created" sap:display-format="Date" Type="Edm.DateTime" Precision="0"/>
  <Property sap:label="Created By" Name="CreatedByUser" sap:quickinfo="Name of Person Who Created Object" sap:display-format="UpperCase" MaxLength="12" Type="Edm.String"/>
  <NavigationProperty Name="to_PurchasingDocumentItemDDL" ToRole="ToRole_assoc_3EDEE251A83223DE4BCA615BA622E7D2" FromRole="FromRole_assoc_3EDEE251A83223DE4BCA615BA622E7D2" Relationship="Z_PURCHASINGDOCUMENTDDL_CDS.assoc_3EDEE251A83223DE4BCA615BA622E7D2"/>
</EntityType>
<EntityType sap:content-version="1" sap:label="Purchase Document Item" Name="Z_PurchasingDocumentItemDDLType">
  <Key>
    <PropertyRef Name="PurchasingDocument"/>
    <PropertyRef Name="PurchasPurchasingDocumentItem"/>
  </Key>
  <Property sap:label="Purchasing document" Name="PurchasingDocument" sap:quickinfo="Purchasing Document Number" sap:display-format="UpperCase" MaxLength="10" Nullable="false" Type="Edm.String"/>
  <Property sap:label="Item" Name="PurchasPurchasingDocumentItem" sap:quickinfo="Item Number of Purchasing Document" sap:display-format="NonNegative" MaxLength="5" Nullable="false" Type="Edm.String"/>
</EntityType>
<Association sap:content-version="1" Name="assoc_3EDEE251A83223DE4BCA615BA622E7D2">
  <End Type="Z_PURCHASINGDOCUMENTDDL_CDS.Z_PurchasingDocumentDDLType"
    Role="FromRole_assoc_3EDEE251A83223DE4BCA615BA622E7D2" Multiplicity="1"/>
  <End Type="Z_PURCHASINGDOCUMENTDDL_CDS.Z_PurchasingDocumentItemDDLType"
    Role="ToRole_assoc_3EDEE251A83223DE4BCA615BA622E7D2" Multiplicity="*"/>
</Association>
<EntityContainer Name="Z_PURCHASINGDOCUMENTDDL_CDS_Entities" sap:supported-formats="atom json xlsx" m:IsDefaultEntityContainer="true">
  <EntitySet sap:content-version="1" Name="Z_PurchasingDocumentDDL" sap:deletable="false" sap:updatable="false"
    sap:creatable="false" EntityType="Z_PURCHASINGDOCUMENTDDL_CDS.Z_PurchasingDocumentDDLType"/>
  <EntitySet sap:content-version="1" Name="Z_PurchasingDocumentItemDDL" sap:deletable="false" sap:updatable="false"
    sap:creatable="false" EntityType="Z_PURCHASINGDOCUMENTDDL_CDS.Z_PurchasingDocumentItemDDLType"/>
  <AssociationSet sap:content-version="1" Name="assoc_3EDEE251A83223DE4BCA615BA622E7D2" sap:deletable="false"
    sap:updatable="false" sap:creatable="false"
    Association="Z_PURCHASINGDOCUMENTDDL_CDS.assoc_3EDEE251A83223DE4BCA615BA622E7D2">
    <End Role="FromRole_assoc_3EDEE251A83223DE4BCA615BA622E7D2" EntitySet="Z_PurchasingDocumentDDL"/>
    <End Role="ToRole_assoc_3EDEE251A83223DE4BCA615BA622E7D2" EntitySet="Z_PurchasingDocumentItemDDL"/>
  </AssociationSet>
</EntityContainer>
```

Figure 3.17 XML Service Metadata Document of the Simple CDS Scenario from Chapter 1 Exposed as an OData Service

3.4.2 Service Implementation Phase

In the previous data model definition phase, SADL maps the CDS data model to an OData EDM and creates a static service definition. At runtime, SADL also takes care of processing the OData requests performed by clients. The OData runtime delegates OData requests to the generic SADL query engine, which generates SQL SELECTS on CDS views based on the request parameters (`$select`, `$top`, `$filter`, etc.) and the requested OData entity.

An incoming OData request for the standard purchase order basic view entity might look like Listing 3.2.

```
GET I_PurchaseOrderEntity?$select=
PurchaseOrder,PurchaseOrderType,PurchasingOrganization,PurchasingGroup,Purchas
eOrderDate&$top=10&$orderby=PurchaseOrderDate asc&$filter=
```

```
( PurchaseOrderDate ge datetime'2018-01-01T00:00:00' and PurchaseOrderDate le datetime'2018-12-31T00:00:00')
```

Listing 3.2 OData GET Request for the I_PurchaseOrderEntity OData Entity

The SADL framework will then transparently transform this request into an SQL SELECT statement on the I_PurchaseOrder CDS view (Listing 3.3).

```
SELECT
PurchaseOrder,
PurchaseOrderType,
PurchasingOrganization,
PurchasingGroup,
PurchaseOrderDate
FROM
I_PurchaseOrder
WHERE
PurchaseOrderDate GE '20180601'
      AND PurchaseOrderDate LE '20180601'
ORDERBY
      PurchaseOrderDate ASCENDING
UP TO 10 ROWS
```

Listing 3.3 Generated SQL SELECT for the OData GET Request

When you compare the OData request and the generated SQL statement, you can see that OData queries can easily be pushed down to a relational database as OData already resembles SQL, and all OData parameters can be mapped to a certain part of the SQL SELECT statement. This property makes OData the perfect candidate for exposing business entities over HTTP because easy transformation of HTTP requests to SQL SELECT statements promotes the Code-to-Data paradigm of SAP HANA and prevents unnecessary data processing on the ABAP server. Table 3.3 shows the mappings between OData parameters and SQL query parts.

OData Parameter	SQL
\$select	SELECT
\$top	UP TO n ROWS

Table 3.3 Mapping between OData Parameters and Corresponding Parts of an SQL SELECT Statement

OData Parameter	SQL
\$orderby	ORDERBY
\$filter	WHERE

Table 3.3 Mapping between OData Parameters and Corresponding Parts of an SQL SELECT Statement (Cont.)

3.5 Business Object Processing Framework

The Business Object Processing Framework (BOPF) is an ABAP framework that constitutes the transactional runtime of an application in the ABAP Programming Model for SAP Fiori. As soon as an application requires write actions such as CUD, a BOPF business object must be generated from the CDS data model, which will take over the transactional part of the app. Currently, four transactional scenarios are supported, as discussed in the following subsections.

3.5.1 Read-Only Application with Quick Actions

In this case, you don't want to enable full create, write, and delete access but only enable a quick action, for instance, for changing the status of the business object. A BOPF business object for this scenario can be generated by annotating the root view of the CDS model with annotations on the header level as shown in Listing 3.4.

```
@ObjectModel: {
  transactionalProcessingEnabled: true,
  compositionRoot:               true,
  writeActivePersistence:        '<SQL View>',
  createEnabled:                 false,
  updateEnabled:                 false,
  deleteEnabled:                 false
}
```

Listing 3.4 Required Annotations for Generating a BOPF Business Object to Implement Quick Actions

(Re)activation of the view will lead to the generation of a BOPF business object, and Quick Actions can be implemented in ABAP for the respective BOPF node. We also

must provide an active persistence using the `writeActivePersistence` annotation. However, in this case, because we don't allow standard (CRUD) access, it will only be used for field mappings between CDS and BOPF.

Note

When exposing a CDS view as an OData service using Auto-Exposure or the RDS scenario, SADL takes the `createEnabled`, `updateEnabled`, and `deleteEnabled` CDS annotations into account and maps them to OData entity set attributes. For instance, if the CDS view doesn't enable write access or doesn't have a linked BOPF business object at all, the respective OData entity set will have the attributes `sap:creatable="false"`, `sap:updatable="false"`, and `sap:deletable="false"`.

3.5.2 New Application without Draft

If we also want to enable standard generic CRUD access for our application, we must set the `createEnabled`, `updateEnabled`, and `deleteEnabled` annotations to `true` (Listing 3.5). Furthermore, we must provide the database table in which the data must be persisted in using the `writeActivePersistence` annotation.

```
@ObjectModel: {
  transactionalProcessingEnabled: true,
  compositionRoot: true,
  writeActivePersistence: '<ActivePersistenceName>',
  createEnabled: true,
  updateEnabled: true,
  deleteEnabled: true
}
```

Listing 3.5 Required Annotations for Full CRUD BOPF Support

In this case, BOPF will generically handle CRUD operations. If we want to enhance the standard generic transaction logic, we can provide BOPF actions, validations, or determinations for the respective node of the BOPF business object. However, if we want to reuse existing legacy ABAP CRUD implementations containing complex business logic, the next scenario is better suited.

3.5.3 Draft-Enabled Application

In this scenario, data isn't immediately written to the standard active database tables but is stored in a draft table. This scenario is very well suited for legacy applications that already consist of database tables and ABAP CRUD code, including complex business logic. We can enable the old database tables for SAP HANA optimized read access by providing CDS views on top of the legacy database tables and still reuse the legacy ABAP code when activating a draft. Activating in this context means transferring the data from the intermediary draft table to the actual table. To do this, the BOPF business object root node will provide an adapter (`/bobf/if_frw_draft~copy_draft_to_active_entity`) that must be implemented by the developer and maps the draft representation of the data to the legacy signature of the ABAP code to write the data into the legacy database tables. The required annotations for generating the draft infrastructure are shown in Listing 3.6.

```
@ObjectModel: {
  transactionalProcessingEnabled: true,
  compositionRoot: true,
  draftEnabled: true;
  writeDraftPersistence: '<DraftPersistenceName>',
  createEnabled: true,
  updateEnabled: true,
  deleteEnabled: true
}
```

Listing 3.6 Required Annotations for Generating the BOPF Draft Infrastructure for a Legacy Application with an Existing Persistence

3.5.4 New Application with Draft

In this scenario, we provide new tables for draft and active entities (Listing 3.7). As in the previous scenario, data will be stored in intermediary draft tables and only written to the active persistence on explicit activation of the entity. However, in this scenario, BOPF will completely take over the transactional processing. We can only enhance the data processing by adding BOPF actions, validations, or determinations to the respective nodes of the BOPF object. This scenario is only suited for new applications. If legacy tables and ABAP CRUD code will be reused, this scenario isn't the right choice.

```

@ObjectModel: {
  transactionalProcessingEnabled: true,
  compositionRoot:                true,
  draftEnabled:                    true;
  writeDraftPersistence: '<DraftPersistenceName>',
  writeActivePersistence: '<ActivePersistenceName>'
  createEnabled:  true,
  updateEnabled:  true,
  deleteEnabled:  true
}

```

Listing 3.7 Required Annotations for Generating the BOPF Draft Infrastructure for a New Application

3.6 SAP Fiori

The open access to backend business data via OData and SAP Gateway enables the use of modern non-ABAP UI technologies for displaying and interacting with business data. The UI of applications developed using the ABAP programming model for SAP Fiori is therefore completely based on the SAPUI5 framework, which implements the SAP Fiori design. The UI part of applications is usually implemented using the cloud-based SAP Web IDE and is either based on SAP Fiori elements applications or freestyle applications. SAP Fiori elements applications, for instance, the list report and object page templates or floorplans, adapt their layouts based on the OData service's metadata and UI annotations defined in CDS views or their respective metadata extensions. Therefore, SAP Fiori elements templates significantly reduce the necessary frontend SAPUI5 JavaScript code to a minimum and increase developer productivity significantly by still providing flexibility using predefined extension points in the frontend.

Additionally, transparent to the developer, the SAP Fiori elements applications implement the necessary CRUD request handling for read-only, draft-enabled, and non-draft-enabled applications, depending on which BOPF scenario has been annotated, generated, and implemented in the backend. Freestyle applications, in contrast to SAP Fiori elements, provide the frontend developer with full flexibility over the UI design and logic but require a lot of effort in the development phase. The UI layout and its controls must be declared manually by the developer, and the necessary SAPUI5 JavaScript logic must be implemented. Therefore, it's not recommended to

use complex draft-enabled BOPF scenarios with freestyle applications because the orchestration of the necessary requests might become too complicated. Moreover, another risk when developing freestyle applications is the violation of SAP Fiori design guidelines. Consequently, only use freestyle applications when the required UI design isn't realizable using one of the SAP Fiori elements floorplans.

3.7 Summary

In this chapter, we gave you an overview of the ABAP programming model for SAP Fiori and its associated technologies. We started with CDS, which are at the center of any application development for SAP S/4HANA. They are the foundation for all application types and can be configured for different application types using domain-specific UIs, analytics, search annotations, or transactional annotations. Next, we looked at SAP Gateway and OData as the technologies enabling easy network-based access to business data stored in SAP NetWeaver backend systems. Then we explored SADL and how it supports providing model-based read and transactional access to CDS entities via OData. The BOPF constitutes the transactional runtime of the ABAP programming model for SAP Fiori and enables ABAP-based CUD operations for data models based on CDS views. It provides several scenarios for different application requirements, which we briefly listed and described.

Contents

Preface	15
---------------	----

PART I SAP S/4HANA and SAP Fiori

1 SAP S/4HANA Architecture	23
---	-----------

1.1 Core Architecture	23
1.1.1 Overview	24
1.1.2 OData	25
1.1.3 SAP HANA	26
1.1.4 SAP NetWeaver ABAP	27
1.1.5 SAP Fiori Front-end Server Based on SAP NetWeaver Gateway	27
1.1.6 SAPUI5	28
1.1.7 SAP Fiori Launchpad	28
1.2 Backend	29
1.2.1 Virtual Data Model	29
1.2.2 The Transactional Architecture (BOPF)	37
1.2.3 Analytics	40
1.3 User Experience	41
1.3.1 SAP Fiori	42
1.3.2 SAP Fiori Launchpad	42
1.4 SAP S/4HANA Editions	42
1.4.1 On-Premise Architecture	42
1.4.2 Cloud Architecture	43
1.4.3 Release Strategies	45
1.5 SAP Cloud Platform	46
1.6 Summary	47

2	SAP Fiori and the Design-Led Development Process	49
2.1	What Is SAP Fiori?	49
2.1.1	SAP Fiori 1.0	50
2.1.2	SAP Fiori 2.0	51
2.1.3	Design Principles	52
2.1.4	Responsiveness and Adaptiveness	54
2.1.5	SAP Fiori Launchpad	54
2.2	Design-Led Development Process	62
2.2.1	Discover	63
2.2.2	Design	64
2.2.3	Develop	64
2.3	Different SAP Fiori App Types	64
2.3.1	Freestyle	64
2.3.2	SAP Fiori Elements	65
2.4	Prototyping Tools	77
2.4.1	Axure	78
2.4.2	SAP Build	78
2.4.3	Building a Prototype Using the SAP Build Tool	80
2.5	Summary	94
3	ABAP Programing Model for SAP Fiori	95
3.1	Core Data Services	95
3.2	SAP Gateway	97
3.3	OData	99
3.3.1	Overview	99
3.3.2	OData Service Creation	101
3.4	Service Adaptation Description Language	109
3.4.1	Data Model Definition Phase	110
3.4.2	Service Implementation Phase	111

3.5	Business Object Processing Framework	113
3.5.1	Read-Only Application with Quick Actions	113
3.5.2	New Application without Draft	114
3.5.3	Draft-Enabled Application	115
3.5.4	New Application with Draft	115
3.6	SAP Fiori	116
3.7	Summary	117
PART II Developing Applications for SAP S/4HANA		
4	Developing an SAP Fiori Elements List Report and Object Page	121
4.1	Core Data Services Views	121
4.1.1	ABAP Development Tools in Eclipse	122
4.1.2	Introduction to the Data Model	125
4.1.3	Creating Basic Interface Core Data Services Views	130
4.1.4	Adding Calculated Fields	136
4.1.5	Adding Data Control Language Files	146
4.2	Transactional Processing with the Business Object Processing Framework	150
4.2.1	Generating a BOPF Business Object from a CDS Model	150
4.2.2	Testing Business Object Nodes	155
4.2.3	Creating a Quick Action Using Business Object Processing Framework	156
4.3	Virtual Elements in Core Data Services	160
4.3.1	Adding a Virtual Element to a Core Data Services View	160
4.3.2	Implementing an ABAP Code Exit to Populate the Virtual Element	160
4.4	Creating an OData Service	161
4.4.1	Auto-Exposing the OData Service	161
4.4.2	Activating the OData Service on the SAP Gateway Hub System	162
4.4.3	Testing the OData Service Using the SAP Gateway Client	164

4.5	Adding User Interface Annotations	164
4.5.1	Creating a Metadata Extension File	165
4.5.2	User Interface-Relevant Annotations for the List Report	166
4.5.3	User Interface Annotations for the Object Page	172
4.6	Generating a List Report Template in SAP Web IDE Full-Stack	177
4.7	Extending the User Interface	181
4.7.1	Implementing User Interface Extensions via Breakout	181
4.7.2	Adding a QUnit Unit Test	184
4.8	List Report Application versus Worklist Application	187
4.9	Summary	187
5	Developing an Overview Page	189
5.1	Core Data Services Views	189
5.1.1	Creating a Simple Core Data Services View	190
5.1.2	Adding a Data Control File	199
5.2	Adding User Interface Annotations	199
5.2.1	Creating Annotations for an Analytical Card	200
5.2.2	Creating Annotations for a List Card	202
5.3	Creating an OData Service	206
5.3.1	Creating the OData Project	206
5.3.2	Adding Overview Page Card CDS Views into the Project	206
5.4	Exposing CDS Views as an OData Service	208
5.4.1	Importing the OData Service	208
5.4.2	Registering the OData Service	209
5.5	Generating an Overview Page Template Project in SAP Web IDE	210
5.5.1	Generating the Basic Overview Page Layout	210
5.5.2	Adding the List Analytical Card	215
5.5.3	Adding the Standard List Card	218
5.5.4	Adding the Bar List Card	220
5.5.5	Adding the Table Card	222
5.5.6	Overview Page Output	224
5.6	Summary	225

6	Developing an Analytical List Page	227
6.1	Introduction	227
6.2	Building the Required CDS Views	227
6.2.1	Building Dimension Views	229
6.2.2	Building Cube Views	229
6.2.3	Building the Main Query View	231
6.3	Configuring the Title Area	232
6.4	Configuring the Filter Area	236
6.5	Configuring the Content Area	239
6.5.1	Configuring the Default Chart	239
6.5.2	Configuring the Table	241
6.6	Combining All the UI Annotations in the Metadata Extension View	243
6.7	Generating an Analytical List Page from SAP Web IDE	247
6.7.1	Adding Key Performance Indicators to the Project	251
6.7.2	Adding Visual Filters to the Project	253
6.8	Summary	259
7	Developing a Freestyle Application	261
7.1	Smart Controls	261
7.1.1	Smart Field	262
7.1.2	Smart Link	263
7.1.3	Smart Form	264
7.1.4	Smart Table	266
7.1.5	Smart Filter Bar	268
7.2	Application Development with the SAP Web IDE Full-Stack	270
7.2.1	Setting Up an OData Service	270
7.2.2	Object Creation Page Using Smart Fields and Forms	276
7.2.3	List Report Page Using Smart Table and Filter Bar	283
7.2.4	Add an OPA5 Integration Test	285
7.3	Summary	295

8	Deploying Applications to the SAP Fiori Launchpad	297
8.1	Uploading a User Interface to the ABAP Frontend Server	297
8.1.1	Deploying Applications from SAP Web IDE	297
8.1.2	Uploading Applications Directly into the Frontend Server	300
8.2	SAP Fiori Launchpad Admin Page	304
8.2.1	Catalogs	305
8.2.2	Groups	306
8.2.3	Roles	306
8.3	Creating the Technical Catalog and Business Catalog	307
8.4	Creating the Application Tiles	309
8.5	Creating Groups for Application Tiles	320
8.6	Creating and Assigning a Transaction PFCG Role to Users	322
8.7	Setting Up Intent-Based Cross-Application Navigation from OVP to LRP	329
8.8	Summary	331

PART III Operating Applications

9	Version Control in SAP Web IDE Using Git	335
9.1	Git Introduction	335
9.2	Git Basics	335
9.2.1	Creating Initial Project Repositories Using GitHub	337
9.2.2	Initializing the Local Repository for the Projects in SAP Web IDE	339
9.2.3	Linking the Local Repository with the Remote Repository in GitHub	340
9.2.4	Submitting Code to Repository (Stage, Commit, Push)	341
9.2.5	Cloning the Project into SAP Web IDE	347
9.2.6	Getting Code from the Remote Branch (Fetch, Merge, Pull)	350
9.2.7	Working with Branches	355
9.3	Summary	360

10	Automated Testing	361
10.1	Backend Test Automation	363
10.1.1	Unit Testing (ABAP Unit)	364
10.1.2	Unit Testing (Core Data Services Test Double Framework)	376
10.2	Frontend Test Automation	380
10.2.1	Unit Testing (JUnit)	381
10.3	End-to-End Test Automation Tools	386
10.3.1	Setting Up Nightwatch.js	387
10.3.2	Creating the Create Purchase Document End-to-End Nightwatch.js Test	388
10.3.3	Running the Create Purchase Document End-to-End Nightwatch.js Test	389
10.4	Summary	390
11	Continuous Integration	393
11.1	Introduction	394
11.1.1	Continuous Integration	394
11.1.2	Continuous Delivery	395
11.1.3	Continuous Deployment	396
11.2	Setting Up a Continuous Integration Pipeline for SAPUI5 on the ABAP Server	397
11.2.1	Setting Up a Local Jenkins Automation Server	398
11.2.2	Creating an Initial Jenkinsfile	400
11.2.3	Creating the Continuous Deployment Pipeline	401
11.2.4	SAPUI5 Grunt Plug-ins	403
11.2.5	Implementing the Build Stage	404
11.2.6	Automatically Triggering Builds on Git Push	407
11.2.7	Implementing the Test Stage	408
11.2.8	Implementing the Deploy Stage	411
11.3	Continuous Integration on the ABAP Server	414
11.3.1	Quality Checking Using the ABAP Test Cockpit	415
11.3.2	Scheduling ABAP Unit Tests Using the ABAP Unit Runner	418
11.4	Summary	421

Appendices 423

A Developing Applications on the SAP Cloud Platform 423

B The Authors 453

Index 455

Index

A

ABAP	
<i>development tools in Eclipse</i>	122
<i>dictionary</i>	96
ABAP code exit	366
ABAP core data services	23, 26
ABAP data model	125
ABAP development tools	43, 365
ABAP Development Workbench	122
ABAP Dictionary	128
ABAP in Eclipse	43
ABAP programming model	39, 95
ABAP test cockpit	415
ABAP unit	121, 364
ABAP unit runner	364, 418
ABAP unit test	365
Access control	147
adaptation description language	
<i>data model definition</i>	110
<i>service implementation</i>	111
Analytical card	215
<i>add</i>	215
<i>annotations</i>	200
Analytical engine	40
Analytical list page	40, 227, 247
<i>annotation selection</i>	250
<i>data connection</i>	249
<i>template</i>	248
Analytics	40
Annotation	
@AbapCatalog.enhancementCategory:	
#EXTENSIBLE_ANY	128
@AccessControl.authorizationCheck:	
#CHECK	132
@Analytics.query:true	227
@Consumption.valueHelpDefinition	197
@DefaultAggregation:#NONE	134
@endUserText.label	128
@ObjectModel	150, 153
@ObjectModel.compositionRoot:true	151
@ObjectModel.createEnabled	151

Annotation (Cont.)

@ObjectModel.dataCategory:	
#VALUE_HELP	198
@ObjectModel.deleteEnabled	151
@ObjectModel.foreignKey.association	197
@ObjectModel.modelCategory:	
#BUSINESS_OBJECT	151
@ObjectModel.representativeKey	132
@ObjectModel.semanticKey	132
@ObjectModel.text.element:	
['Description']	132
@ObjectModel.transactionalProcessing-	
Enabled:true	151
@ObjectModel.updateEnabled	151
@ObjectModel.writeActivePersistence	152
@ObjectModel.writeActivePersistence:	
'ZPURCHDOCUMENT'	151
@OData.publish:true	161, 249
@selectionVariant	233
@Semantics.amount.currencyCode	129
@Semantics.imageUrl:true	167
@Semantics.quantity.unitOfMeasure	129
@UI.chart	200, 238, 245
@UI.datapoint.criticalityCalculation	203
@UI.dataPoint.visualization:	
#NUMBER	200
@UI.facet.parentId	174
@UI.facet.type:#COLLECTION	174
@UI.headerInfo	167
@UI.hidden	267
@UI.identification:	201
@UI.lineItem	166–168, 171, 175, 266
@UI.presentationVariant	244
@UI.selectionField	169
@UI.selectionVariant	244
@VDM.viewType:#BASIC	132
#ALLOWED	128
semanticObjectAction:	201
writeActivePersistence	114
Z_C_PurchaseDocumentLRP	165
App launcher	
dynamic	309
static	309

Application development 270
 Application tiles 309, 320
 Apply stash 343
 Artificial intelligence 423
 Authorization field
 ACTVT 147
 ZPURCHORGA 147
 Automated testing 361
 Axure 49, 78

B

Bamboo 398
 Bar list card 220
 Behavior-driven development 289
 Blue Ocean 398
 Branches 355
 Breakout 181
 Build task 404
 Business intelligence consumer services 40
 Business logic 39
 action 39
 determinations 39
 validations 39
 Business object
 Z_I_PurchaseDocumentTP 156
 Business Object Nodes 155
 Business object processing framework 27,
 37, 95, 113, 414
 business object generation 150
 draft-enabled application 115
 new application with draft 114–115
 quick action 156
 read-only application 113
 transactional processing 150

C

Calculated fields 136
 Catalog 305
 business 305, 307
 link to user 322
 technical 305, 307
 Change and transport system 95
 classic online transaction processing 40
 Clean task 404

Cloud Foundry 46, 439
 Commit 341
 Composite interface view 136
 Concourse CI 398
 Constructor injection 371
 Consumption view 35, 191
 Z_C_PurchaseDocumentOVP 191
 Contact quick view 168
 Content area 239
 Continuous delivery 395
 Continuous deployment 393, 396
 pipeline 401
 Continuous integration 393–394
 ABAP server 414
 SAPUI5 397
 Core data services 95
 basic interface 130
 test double framework 121, 376
 virtual elements 160
 Core data services view 365
 create 190
 simple 190
 Core data services views 121, 189
 Cross-application navigation 329
 Cube views 229

D

Data control file 199
 Data control language 199
 Data control language files 146
 Data definition language 26, 96
 analytical annotations 96
 search annotations 96
 transactional annotations 96
 UI annotations 96
 Data dictionary 96
 Data provider class 106
 Default chart 239
 Dependency inversion principle 372
 Deploy stage 411
 Deployment 297
 Design-led development 62
 design 64
 develop 64
 discover 63

Dimension views 229
 Docker 398

E

Eclipse 122
 Eclipse Oxygen 122
 Embedded analytics 40
 End-to-end tests 363
 Enterprise search 26
 Entity relationship model 125
 Extreme programming 394

F

Feature packs 45
 Fetch 342
 Filter area 236
 Filter bar 283
 consumption view 196
 Freestyle app 316
 Freestyle application 261
 development 270

G

Git 335
 automatic trigger 407
 basics 335
 push 407
 GitHub 337
 Group
 link to user 322
 Groups 306, 320
 Grunt 398, 403

H

HTML5 28, 46

I

Indirect input 375
 Indirect output 375
 Info access service 40

Integrated development environment 122
 Integration tests 363
 Interface annotations 164
 Internet of Things 46, 423

J

Java 46
 Jenkins
 automation server 398
 jenkinsfile 400
 multibranch pipeline 402
 Journey 384
 JSON 27

K

Key performance indicators 251
 Kubernetes 46

L

Layer
 #CORE 165
 #CUSTOMER 165
 Lint task 404
 List card
 annotation 202
 bar list card annotations 203
 consumption view 193–194
 interface view 192
 standard list card annotations 202
 List report 166, 187, 283
 create 177
 criticality 168
 header info 167
 image 167
 line item 166
 quick actions 171
 searchable 170
 selection field 169
 template 177
 value helps 171
 Local repository 339–340

M

Machine learning	423
Main query view	231
Merge	342
Metadata extension file	165
Metadata extension view	243
Method	
<i>class_setup</i>	140
<i>overall_price_no_items</i>	140
<i>prepare_test_data_set</i>	140
Mock data	287
Mock server	286
Model provider class	106
Model view controller	42
MySQL	425

N

Neo	46
News tile	309
Nightwatch.js	387, 389
Node.js	46

O

Object creation page	276
Object factory class	370, 372
Object page	172
<i>create</i>	177
<i>facet</i>	174
<i>header info</i>	172
<i>line item</i>	176
<i>quick actions</i>	176
OData	24–25, 27, 40, 99, 161, 206
<i>architecture</i>	25
<i>auto-exposure</i>	101
<i>CDS views</i>	208
<i>client-side paging</i>	100
<i>counting</i>	101
<i>create project</i>	206
<i>CRUD</i>	107
<i>filtering</i>	100
<i>formatting</i>	101
<i>mapped data source</i>	106
<i>overview page</i>	206
<i>projecting</i>	100

OData (Cont.)

<i>query</i>	100
<i>referenced data source</i>	103
<i>service creation</i>	101
<i>sorting</i>	100
OData Service	161, 206, 208, 270, 440
<i>auto-expose</i>	161
<i>create</i>	161
<i>custom logic</i>	449
<i>on SAP Gateway hum system</i>	162
<i>registering</i>	209
<i>setup</i>	270
<i>test</i>	164
One-page acceptance	363
One-page acceptance test	285
Online analytical processing	40
Open SQL	101
Oracle	425
Overall item price	136
Overall price criticality	144
Overall purchase document price	136
Overview page	40, 189
<i>analytical card</i>	189
<i>list card (bar)</i>	190
<i>list card (standard)</i>	190
<i>table card</i>	190

P

Page objects	385
Page output	224
Path expression	
<i>_PurchaseDocumentItem.OverallItem-Price</i>	142
PostgreSQL	46
Project repositories	337
Prototyping	77
Pull	341
Purchase document	279, 281, 389
<i>ZPURCHDOCITEM</i>	126
<i>ZPURCHDOCUMENT</i>	126
Push	341

Q

Quality checking	415
QUnit	184, 361, 381, 408

R

RabbitMQ	47
Rebase	342
Redis	46
Remote function call	27, 98
Remote repository	340
Reset	344
REST	25
Roles	306, 322

S

SAP Analysis for Microsoft Excel	40
SAP API Business Hub	36
SAP Ariba	51
SAP Basis	26
SAP Build	49, 78
<i>create prototype</i>	80
<i>design phase</i>	83
<i>discover phase</i>	80
<i>ideate phase</i>	84
<i>prototype phase</i>	85
<i>research phase</i>	82
<i>scope phase</i>	80
<i>synthesize</i>	82
<i>validate phase</i>	90
SAP Business Warehouse	27
SAP Cloud for Customer	51
SAP Cloud Platform	46, 423
<i>application programming model</i>	424
<i>DevOps</i>	46
<i>setup SAP Web IDE</i>	426
SAP Cloud Platform Integration	47
SAP CoPilot	59
<i>business objects</i>	61
<i>chat</i>	61
<i>notes</i>	60
<i>quick actions</i>	61
<i>screenshots</i>	60
SAP Development Tools	123
SAP Dynpro	37
SAP ERP	24
SAP Fiori	23, 27–28, 49–50, 116
<i>adaptive design</i>	54
<i>develop</i>	121

SAP Fiori (Cont.)

<i>elements list report</i>	121
<i>front-end server</i>	27
<i>markup</i>	425
<i>object page</i>	121
<i>responsive design</i>	54
<i>UI</i>	181
<i>UX</i>	37, 41, 52
SAP Fiori apps	64
<i>analytical list page</i>	73
<i>elements</i>	65
<i>freestyle</i>	64
<i>list report</i>	66
<i>object page</i>	68
<i>overview page</i>	71
<i>worklist page</i>	75
SAP Fiori launchpad	24, 28, 54, 297, 329
<i>admin page</i>	304
<i>catalog</i>	305
<i>components</i>	62
<i>groups</i>	306
<i>home page</i>	57
<i>me area</i>	57
<i>notifications</i>	58
<i>roles</i>	306
<i>UX</i>	42
<i>viewport</i>	55
SAP Fiori UI	
<i>extend</i>	181
SAP Gateway	24–25, 27, 97, 99
<i>embedded deployment</i>	28
<i>hub deployment</i>	28
<i>hub system</i>	108, 208
<i>service builder</i>	103, 262
SAP GUI	24, 28, 50
SAP HANA	26
<i>architecture</i>	26
<i>database table</i>	126
<i>enterprise cloud</i>	45
SAP Leonardo	46
SAP Lumira	40
SAP NetWeaver	24, 43, 97
<i>ABAP</i>	27
<i>ABAP backend server</i>	124
<i>application server for ABAP</i>	43
<i>backend</i>	43

- SAP NetWeaver Application Server for
 - ABAP 24
 - SAP Notes 43
 - SAP S/4HANA 23, 361
 - architecture* 23
 - backend* 29
 - backend system* 28
 - basic interface view layer* 134
 - cloud* 42
 - on-premise* 42
 - on-premise architecture* 42
 - release strategies* 45
 - UX* 41
 - virtual data model* 121
 - SAP S/4HANA Cloud 23, 42, 423
 - architecture* 43
 - private cloud* 43
 - public cloud* 43
 - release strategies* 45
 - SDK* 47
 - SAP SuccessFactors Employee Central 51
 - SAP Tax Service 47
 - SAP Web IDE 46, 177, 200, 208, 210, 247, 335
 - business application* 429
 - cloning project* 347
 - deployment* 297
 - export application* 300
 - interface extensions* 181
 - overview page template project* 210
 - SAP Web IDE Full-Stack 270
 - SAP Cloud Platform development* 426
 - SAPUI5 15, 28, 42, 52, 261, 361
 - application* 270
 - grunt* 403
 - Service adaptation definition language 27
 - Service adaptation description language 95, 109
 - Setter injection 371
 - Sinon.js 383
 - Smart controls 261
 - Smart fields 276
 - Smart filter bar 268
 - Smart forms 264, 276
 - definition* 277
 - Smart link 263
 - Smart table 266, 283
 - Software development kit 15
 - Source code management 335
 - Stage 341
 - Standard list card 218
 - Support packages 45
- ## T
- Table 241
 - Table card 222
 - annotations* 205
 - consumption view* 195
 - Table category
 - #TRANSPARENT* 128
 - Target mapping 311
 - Test automation
 - backend* 363
 - frontend* 380
 - Test automation tools
 - end-to-end* 386
 - Test double 374
 - dummy* 374
 - fake* 375
 - mock* 375
 - spy* 375
 - stub* 375
 - Test fixture 364
 - Test stage 408
 - Test-driven development 136, 361
 - Title area 232
 - Total cost of ownership 28
 - Transaction
 - /IWFND/GW_CLIENT* 164
 - /IWFND/MAINT_SERVICE* 101, 108, 162
 - /n/IWFND/MAINT_SERVICE* 208
 - /UI2/SEMObj* 315
 - LPD_CUST* 312
 - PFCG* 28, 148–150, 264, 322–323
 - SACMSEL* 150
 - SE16* 128
 - SE80* 122
 - SEGW* 103, 106, 206, 262
 - SU20* 146
 - SU21* 146
 - SU22* 149
 - Travis CI 398

U

- Unified modeling language 364
- Unit testing 184, 361, 376, 381
- Upload application 300
- User interface
 - Annotations* 199, 435
 - build* 442

V

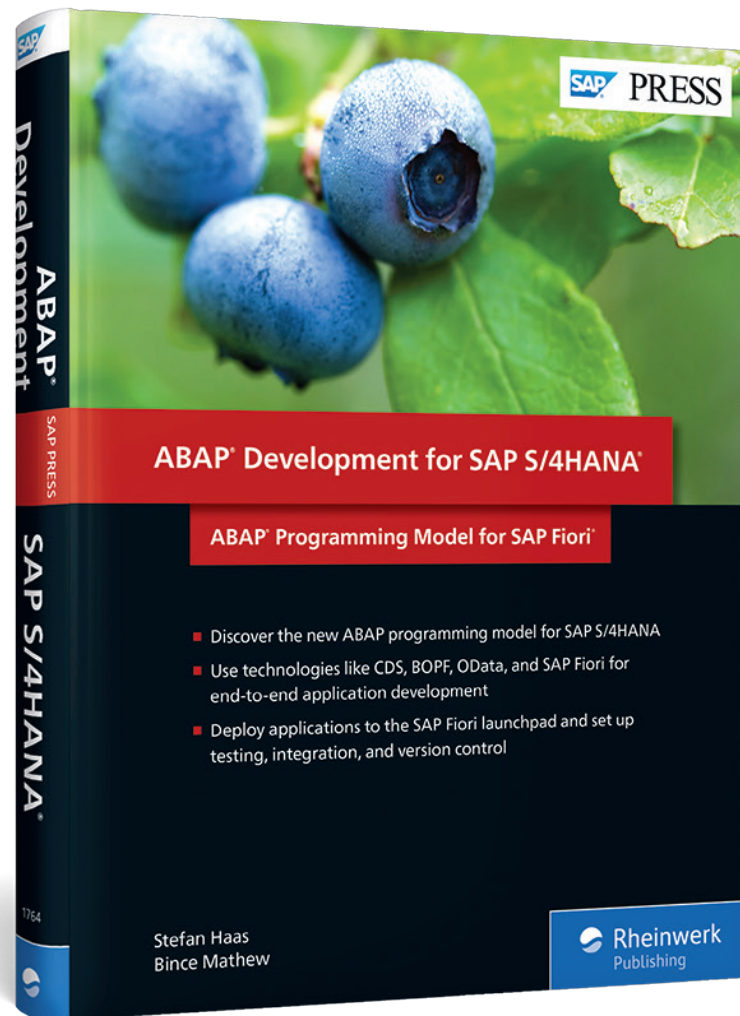
- Version control 335
- Virtual data model 23, 227
- Virtual elements 160, 365
 - add* 160
 - calculate* 161
 - filter* 161
 - populate* 160
 - sort* 161

W

- Web dynpro 28, 37
- Worklist 187

X

- XML 27



Stefan Haas, Bince Mathew

ABAP Development for SAP S/4HANA: ABAP Programming Model for SAP Fiori

461 Pages, 2018, \$79.95

ISBN 978-1-4932-1764-9

 www.sap-press.com/4766



Stefan Haas is a senior developer at SAP working on Enterprise Asset Management applications for SAP S/4HANA within the Digital Supply Chain and Manufacturing lines of business. He is an experienced full stack developer on all layers of SAP S/4HANA ranging from SAP HANA, core data services (CDS), and ABAP over OData and SAP Gateway to SAP Fiori, SAPUI5, and JavaScript. Additionally, he was a fellow at the Machine Learning Engine team of SAP's Predictive Maintenance and Service cloud solution. He holds a master's degree in computer science, with a minor in business, from Ludwig Maximilian University of Munich and is a certified Cloud Foundry developer.



Bince Mathew is a senior developer at SAP with more than nine years of experience in providing solutions on various SAP mobility platforms. He is an expert in areas such as SAP Fiori, SAP S/4HANA, SAP Syclo, SAP Mobile Platform (formerly Sybase Unwired Platform), SAP Afaria, and the Internet of Things. He has worked with industry verticals such as Manufacturing and Retail. He is an SAP Certified Technology Associate for SAP Fiori and a published author with a focus on SAP Fiori developments. He was also given the best speaker award for SAP Fiori at the SAP Inside Track event held by SAP. He is active in the SAP Community Network (SCN) on topics related to SAP mobility.

We hope you have enjoyed this reading sample. You may recommend or pass it on to others, but only in its entirety, including all pages. This reading sample and all its parts are protected by copyright law. All usage and exploitation rights are reserved by the author and the publisher.