





## Reading Sample

This chapter provides the foundation of knowledge you'll need to create OData services. You'll learn about the methods to create OData services in SAP Gateway: service development and service generation. Then you'll explore the main steps using the Service Builder, as well as other tools and scenarios.

-  **"Introduction to OData Service Creation"**
-  **Contents**
-  **Index**
-  **The Authors**

Bönnen, Diehl, Drees, Fischer, Strothmann

### SAP Gateway and OData

810 pages | 12/2023 | \$89.95 | ISBN 978-1-4932-2468-5

 [www.sap-press.com/5759](https://www.sap-press.com/5759)

## Chapter 5

### Introduction to OData Service Creation

*This chapter explains the end-to-end cycle, and the specific tools, for creating SAP Gateway services, both for service development and for service generation.*

As you'll recall from Chapter 2, OData services are what implement the OData protocol and expose an endpoint that allows access to data. The number of OData services shipped with SAP Gateway is limited and will likely remain rather low because, by nature, OData services are granular and mostly tailored to individual use cases. More commonly, services are shipped as part of products such as SAP Fiori, SAP S/4HANA, or SAP mobile solutions. A large amount of development time can go into building the right OData service, so understanding this process is essential.

**Out-of-the-box  
OData services**

The central interface that is used to define and implement services within SAP Gateway is the Service Builder (Transaction SEGW). After you've created a service in the Service Builder, the service can be used directly in any interface. The Service Builder is a one-stop shop with respect to SAP Gateway service development and is supplemented by additional support tools. In certain cases, it even allows you to perform selected steps in third-party tools and then import the results (e.g., usage of an OData modeler for the model definition). Besides the Service Builder, the ABAP Development Tools (ADT) for Eclipse (also known as ABAP in Eclipse) can play an important role in the ABAP programming model for SAP Fiori and the ABAP RESTful application programming model. These models are used to build core data services (CDS) views (see Chapter 8 and Chapter 9) and to build new ABAP RESTful application programming model-specific development artifacts such as behavior definitions.

The main objective of this chapter is to provide an overview of the process of service creation, which we then discuss in more detail in Chapters 6 through Chapter 9. To achieve this overview, in Section 5.1, we'll briefly look at the two methods used to create OData services in SAP Gateway (service development and service generation) and continue in Section 5.2 to explain the main steps in the process of service creation. In Section 5.3, we look at the main tool involved in service creation: the Service Builder. We then complement this first look at the Service Builder with a quick look at some

of SAP Gateway’s other tools that support service creation and maintenance. This section will give you an idea of the tools that are available to assist with tasks during the service creation process.

In Section 5.4, we then dig more deeply into service creation and describe in detail the three main steps in service creation: data model definition, service implementation, and service maintenance. Also, we look at additional topics related to service creation, such as redefining services and reusing existing SAP Gateway services in extension scenarios to create custom OData services based on OData services that are predelivered by SAP. Finally, we introduce you to the development paradigm used for service development—the OData channel—in Section 5.5.

5.1 Methods for Creating an OData Service

You can create OData services with SAP Gateway in one of two ways:

Development versus generation

- **Service development**  
The classic option is the code-based development of SAP Gateway services. This ABAP-based option is extremely flexible and allows you to develop highly efficient and specialized services, but it also requires some significant technical know-how.
- **Service generation**  
The second option is the generation of SAP Gateway services. There are four main methods of service generation:
  - *Mapping to a data source*: Allows you to generate a service by mapping the create, read, update, delete, and query (CRUD-Q) methods of an entity set to a data source. This mapping is supported for the following data sources:
    - Remote function call (RFC)/Business Object Repository (BOR) function modules
    - Search help (only READ and QUERY method)
    - CDS views (only READ and QUERY method)
  - *Redefinition*: Allows you to define a service based on an existing data source or an existing SAP Gateway service.
  - *Referenced data sources (RDS)*: Allows you to define a service based on a CDS view.
  - *Creating CDS views with Eclipse*: Generates an OData service without the Service Builder by creating CDS views using Eclipse and setting the `OData.publish:true` option.

Of these two approaches, service generation is the quicker approach and requires a lot less effort. On the other hand, if not based on CDS views, the service might be more limited, and thus this approach is primarily recommended for developing simple, straightforward services. Service generation doesn’t offer much optimization potential because, without custom coding, you’re restricted to what the service generators offer. In most real-world situations, you might opt for service development because the advantages are well worth the effort. Still, if you have search helps, Generic Interaction Layer (GenIL) or service provider interface (SPI) objects, analytical queries such as SAP Business Warehouse (SAP BW) Easy Queries or OData, or a suitable RFC function module or Business Application Programming Interface (BAPI) and are aiming for a quick result, then service generation might be an option for you.

However, with the advent of SAP S/4HANA, OData services based on CDS views, as shown in Figure 5.1 ❶, became the preferred approach for OData service development. Because this kind of service can also support smart templates for user interface (UI) development, a lot of scenarios in SAP S/4HANA don’t require SAPUI5 coding but the development of appropriate CDS views and Business Object Processing Framework (BOPF) objects. With BOPF objects, you can also develop transactional applications that can even support the draft handling, which allows users to keep unsaved changes if an editing activity is interrupted and to resume editing later.

Even when using OData services that are generated from CDS views, the execution of the OData service can be enhanced by adding additional business logic in the data provider extension class. (We’ll return to these specific options in Chapter 7 and Chapter 8, when we discuss service generation in greater detail.)

In systems based on SAP NetWeaver 7.50 or higher, you can still develop OData services using service development and the mapping of data sources, as shown in Figure 5.1 ❷. In this way, you can leverage existing resources such as ABAP classes and RFC function modules when using SAP Business Suite EHP 8 or higher or when using on-premise SAP S/4HANA.

Service creation process

Recommendation for SAP S/4HANA 2022 or Later

With SAP S/4HANA 2022 and later, we recommend using the ABAP RESTful application programming model instead of Transaction SEGW and BOPF objects. For more details, see Chapter 9. For BOPF-based OData services, a migration tool has been delivered with SAP S/4HANA 2023.



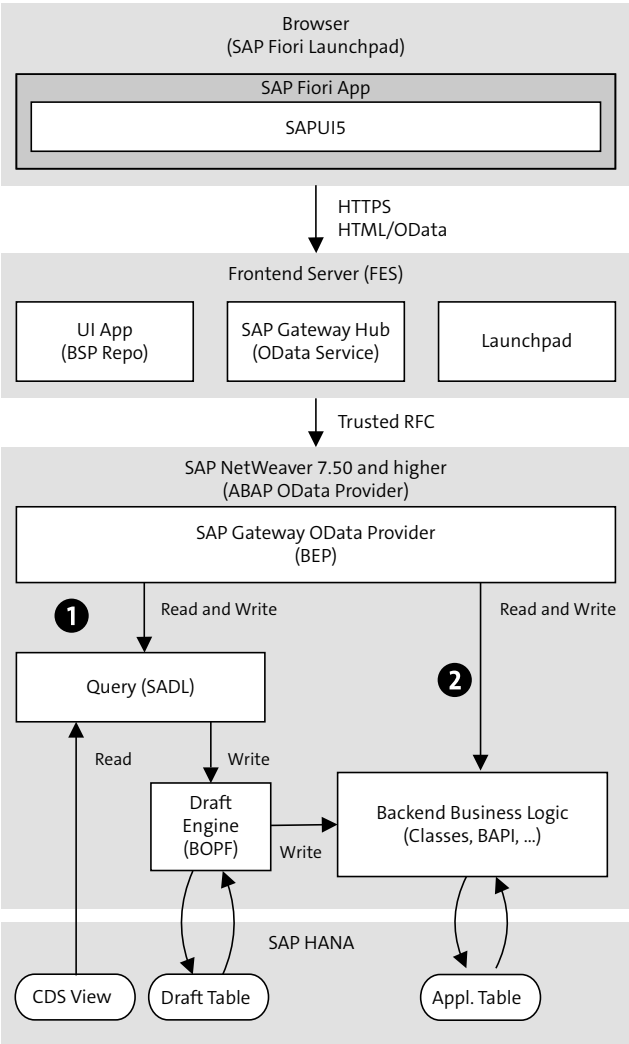


Figure 5.1 SAP Gateway OData Service Provisioning for SAP Fiori: The Transformation to SAP S/4HANA

Whether you’re using service development or service generation, you create an OData service by following the SAP Gateway service creation process, as discussed next.

5.2 Service Creation Process Overview

In this section, we’ll introduce the general steps in OData service creation and explain how the two methods for creating an OData service (service development and service generation) fit into this process. This explanation

of the service creation process is somewhat simplified in an effort to explain it with distinct and sequential steps (a waterfall approach). In reality, some of these steps can also be performed out of order (an incremental approach). We’ll go into a bit more detail about this distinction at the end of this section, after presenting the simplified process.

This process consists of three main phases: data model definition, service implementation, and service maintenance. Depending on whether you opt for service development or for service generation, the individual phases of the service creation process can have different flavors. These flavors result in different paths that can be taken during the actual process.

Before you can start with this process, you must complete the process of *service definition* as a prerequisite. This process involves identifying what service to create and specifying its details. Ideally, you’ve done all this together with your client developers so that you know exactly what data they require and how this works with the artifacts in the SAP S/4HANA (or SAP Business Suite) backend system that will be the basis for your SAP Gateway service. After you have the service definition, you can start with the three development phases of the service creation process.

In the first phase, *data model definition*, you define the model your service is based on. That is, you define the required artifacts such as entity types, entity sets, associations, and other components that your service will use (refer to Chapter 2 for explanations of these components). After data model definition, you must generate the repository objects and register them in the SAP backend system so that you can proceed with the next main phase, *service implementation*.

Data model definition phase

In the service implementation phase, the operations that are supported by the service are implemented. At this point, the different tracks for service development and service generation come into play:

Service implementation phase

- For service development, operations that are supported by the service are implemented using ABAP coding.
- For service generation, there are four paths, depending on the type of generation chosen:
  - If you use data source mapping, service implementation takes place by mapping the OData model to the methods of an RFC function module, search help, or CDS view.
  - If you use redefinition, there is no service implementation step. You only must perform the model definition step because the implementation of the service is generated based on the customizing that has been performed in the model definition step.

- If you reference a data source, there is again no service implementation step. Instead, you include one or more existing entity sets and associations of a CDS view into a data model.
- If you use the Eclipse-based ADT to create a CDS view by setting the `odata.publish:true` option, there is no service implementation step. Based on the CDS view definition, the implementation of the service is generated.

Service maintenance phase

The third phase of the service creation process, *service maintenance*, publishes the service so that it becomes visible in the service catalog of the SAP Gateway system. In effect, the created OData service can then be consumed.

The three phases—data model definition, service implementation, and service maintenance—are shown in Figure 5.2. Steps that are only performed in service development are marked with one color, and steps that are only executed in service generation are marked with a different color. Steps that must be performed in both the development and generation of OData services in SAP Gateway are marked with both colors.

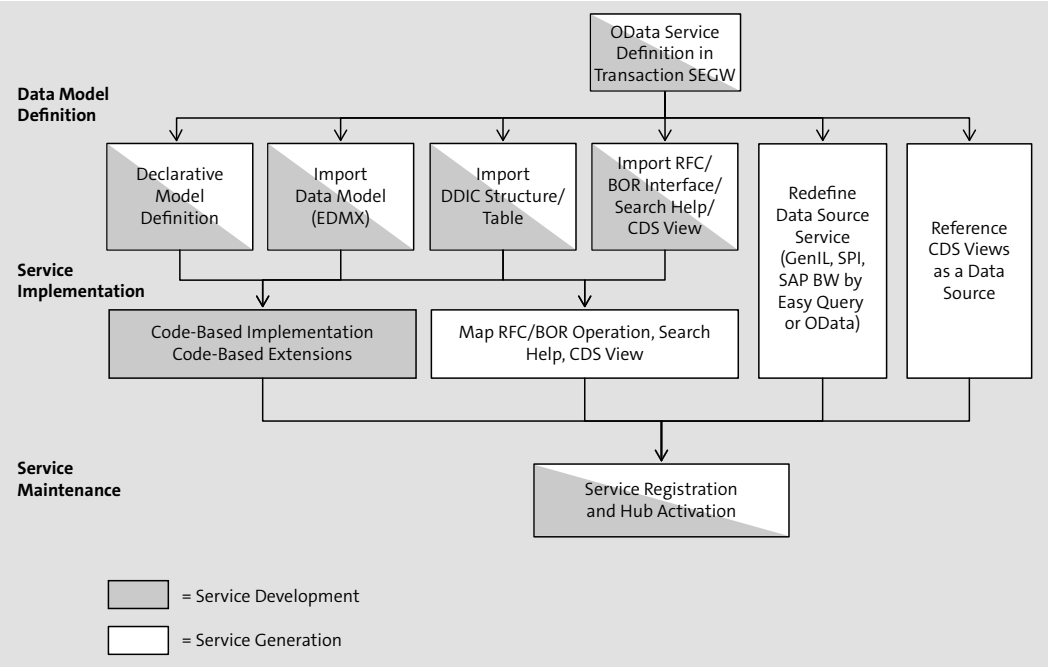


Figure 5.2 Service Creation Process

Although we clearly delineate the two methods of service creation (service generation and service development), it's actually possible to mix these in a way that suits you best. For example, you can create an OData service where one entity set is implemented using the RFC/BOR Generator (service

generation), while a second entity set is implemented using code-based implementation (service development). You can also generate a read-only OData service based on a CDS view and extend this service via a code-based implementation so that it also supports updating business data.

As mentioned earlier, we've presented the service creation process in a highly structured and clearly sequential way. This waterfall approach allows you to easily understand what the different phases are for. In real-world projects, after you've understood how it works, you can adjust the sequence to what fits you best (within certain boundaries). The one exception to this rule is the service maintenance phase—this is almost always a one-time activity. As soon as a service is registered and activated (published), you don't have to touch these settings anymore, even if the implementation and/or model definition changes.

**Exception**

Service publication is a one-time activity as long as you don't perform major changes. Registering the service for additional SAP backend systems, for example, is such an activity in which you must go back to the service maintenance phase. Again, though, changes in the implementation of an already published service or in the data model can be used in the already published service without any further activities.

For all other phases, you'll typically follow an incremental approach: You build a service (or part of it), execute and test it, and then go back and refine that same service until it fits all of your needs. During the creation of an OData service, you may change the model and/or the service implementation multiple times.

Furthermore, an approach often used in real-world projects is to perform the service implementation and the service maintenance in a different order. Performing the service maintenance with a service implementation stub before the actual service implementation allows you to browse meta-data (service document and service metadata document), even if the service itself doesn't yet have any functionality. You've basically started with a service stub and can then fill this stub in an incremental way.

Figure 5.3 shows the incremental service creation process. Based on the diagram shown in Figure 5.2, we've added incremental steps to the original process.

These incremental steps are displayed by the solid arrows, which represent potential transitions among the three phases of data model definition, service implementation, and service maintenance, which are contained in

Incremental service creation process

horizontal boxes. The dotted line stands for the one-time activity of service publication as part of the service maintenance phase.

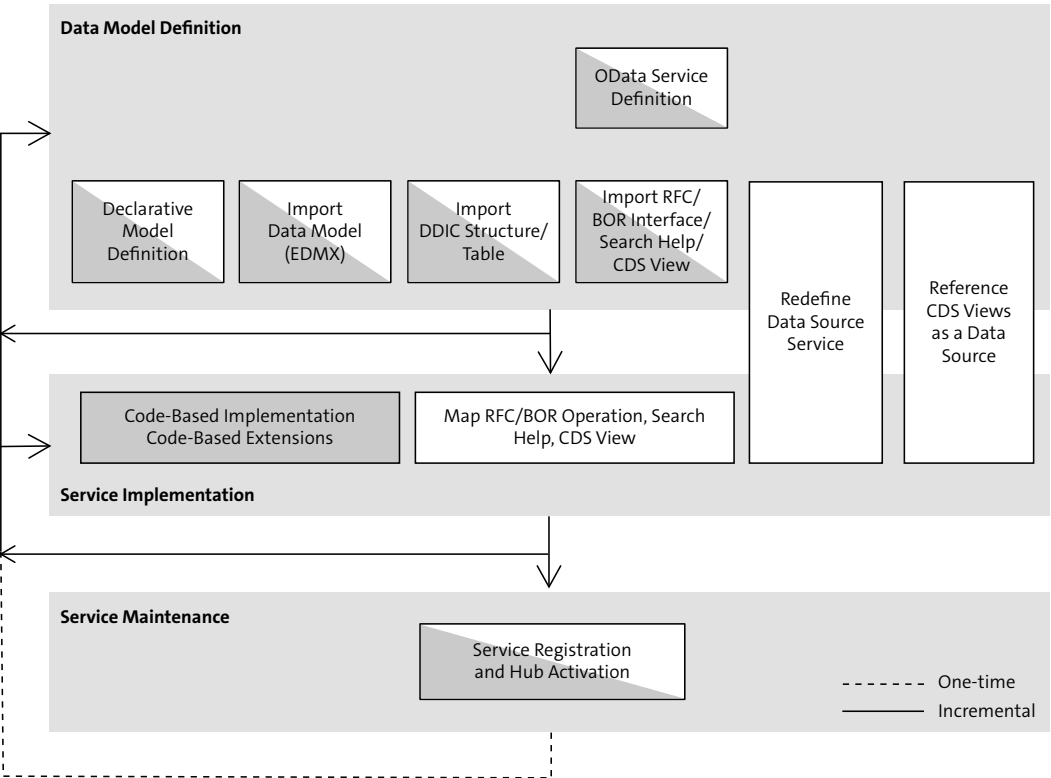


Figure 5.3 Incremental Service Creation

5.3 SAP Gateway Toolset

SAP Gateway provides a set of tools to address all your needs, from development to testing to operations. For now, we'll skip tools targeted at operating SAP Gateway and focus specifically on tools related to service creation. In this section, we'll take a closer look at Service Builder—the central, one-stop development tool for SAP Gateway services—and the additional, well-integrated tools that support you during the SAP Gateway service creation process.

5.3.1 Service Builder

The Service Builder contains all relevant functions for modeling and developing OData services in SAP Gateway. This includes both code-based development of services and the generation of OData services. Also, it provides

Supports development lifecycle of an OData service

direct access to additional development-related functions such as service registration/activation and service validation. The Service Builder supports the entire development lifecycle of an OData service in SAP Gateway, and you can start it using Transaction SEGW, as shown in Figure 5.4.

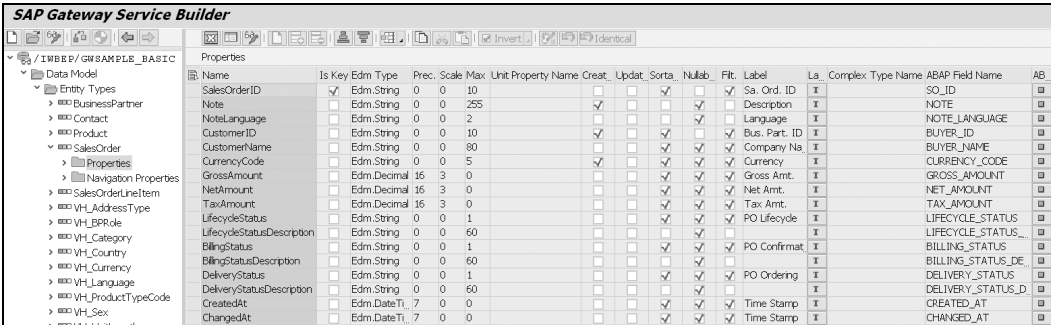


Figure 5.4 Service Builder

Overall, the Service Builder addresses the needs of both experienced and less experienced developers, as well as non-developers. Whereas experienced developers can develop their own source code with maximum flexibility in their service implementation, they still can use the built-in OData modeler and other tools to simplify the development process. Less experienced developers will appreciate the ability to use tools that generate OData services without having to write a single line of code.

The Service Builder allows for the central display and creation of definitions for an OData service. These definitions include runtime artifacts (i.e., model provider class [MPC], data provider class [DPC], model, and service); OData artifacts (i.e., entity set, entity type, and properties); and data sources and models.

The modeling environment follows a project-based approach, and all relevant data is consolidated into these projects. Development using the Service Builder is therefore organized in projects, and creating a project is the starting point of every service development using the Service Builder. Projects are used to bundle all the artifacts needed for service development in one central place, thereby providing a means to organize the development process. The Service Builder allows developers to open several projects at the same time, as shown in Figure 5.5 (in this example, ZPRODUCT and ZSALESORDER).

Project-based development

**BEP Component**

From a technical system perspective, the Service Builder is used in a system where the Business Enablement Provisioning (BEP) component is installed,

which is typically an SAP backend system (refer to Chapter 4 for a discussion of the different deployment options for SAP Gateway). The BEP component is delivered as the IW\_BEP add-on until SAP NetWeaver 7.31. Starting with SAP NetWeaver 7.40 SP 02, the BEP component is included in SAP NetWeaver itself as part of the SAP\_GWFND component. As a result, you can develop OData services using the Service Builder without additional effort in all systems running SAP NetWeaver 7.40 SP 02 or later, including all SAP S/4HANA systems as well as SAP Business Suite systems with EHP 7 and later.

Because the Service Builder is part of the BEP component that is typically (but not necessarily) installed on the SAP backend system, you define the service model (i.e., MPC) as well as the service logic (i.e., DPC) on the same system where the BEP component is deployed. This location is important to remember when referencing other ABAP Repository objects, such as Data Dictionary (DDIC) elements (e.g., structures or data elements) that are required when calling, for example, an RFC or BAPI.

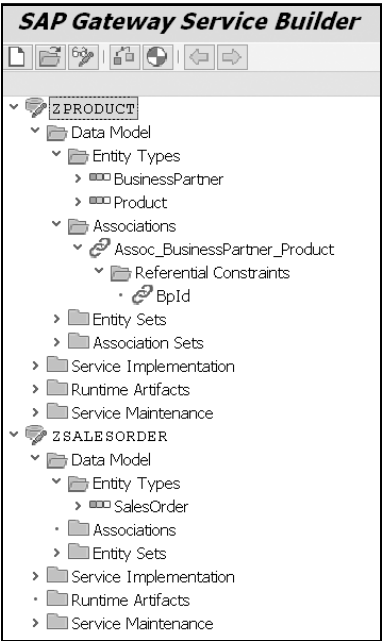


Figure 5.5 Project-Based Development

Comprehensive support for building OData services

The objective of the Service Builder is to provide comprehensive support for building OData services in a declarative way or by reusing existing business objects in the SAP backend system. However, restrictions exist as to what can be declared or generated. Advanced OData features may need to be implemented manually, and certain operations aren't available in a

redefined business object. The result of what you do in the Service Builder will always be ABAP classes, which are based on the OData channel programming model of SAP Gateway (covered in Section 5.5). You can always drill down to understand what is going on during service execution or tweak the code.

5.3.2 Supporting Tools during the Service Creation Process

As mentioned earlier, the main tool during the service creation process is the Service Builder. At the same time, SAP Gateway provides additional tools that are especially useful during the development of SAP Gateway services. These tools allow, for example, early testing of services or the tracing of what is happening when calling a service. As such, in this section, we'll briefly introduce you to some of these functionalities. For a more comprehensive description of the development support and administration toolset of SAP Gateway, see Chapter 16.

SAP Gateway Client

The SAP Gateway client can be used for both testing and troubleshooting and is a representational state transfer (REST) client built into SAP Gateway. This client can be started from within SAP GUI using Transaction /IWFND/GW\_CLIENT. After you've created a service, you can also use this tool for its first test, as shown in Figure 5.6.

Testing and troubleshooting

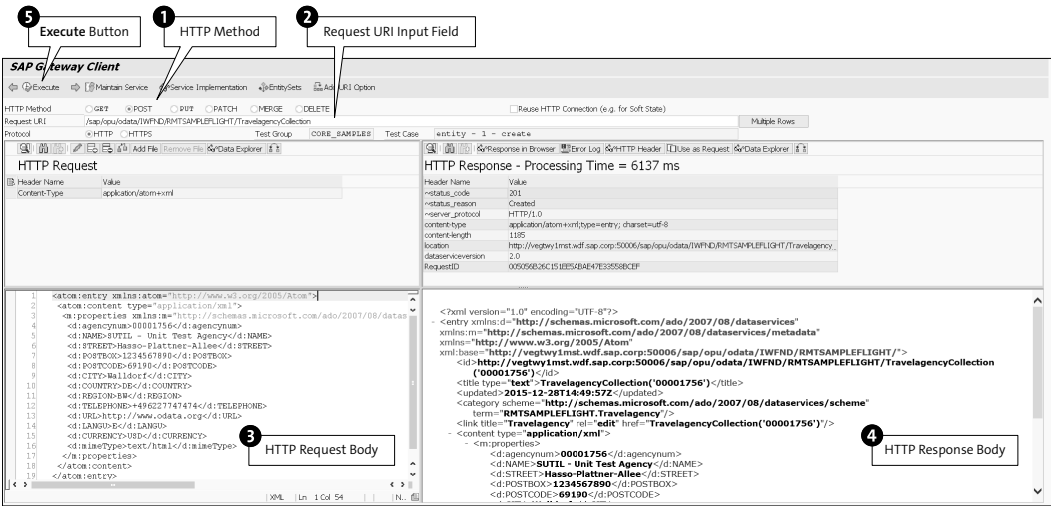


Figure 5.6 SAP Gateway Client: Create Request

First, select an HTTP method such as GET, POST, PUT, PATCH, MERGE, or DELETE ❶. Then, enter the uniform resource indicator (URI) of your request into the



**Request URI** input field ❷. You can also set a certain HTTP header if needed. The body of an HTTP request can be entered either manually or uploaded from a file ❸. In addition, you can use the **Use as Request** function to create, for example, an update request based on the response ❹ of a read request that has been issued against the URI before. Finally, perform the HTTP request by clicking **Execute** ❺.

**Test cases** A particularly useful feature of the SAP Gateway client is that test cases can be stored in a database. The test case shown in Figure 5.6 is one of more than 300 sample test cases that are delivered in the **CORE\_SAMPLES** test group for the **TEA\_TEST\_APPLICATION** and **RMTSAMPLEFLIGHT** standard test services. Note that the test cases of the **CORE\_SAMPLES** test group must be manually created from within the SAP Gateway client by selecting **SAP Gateway Client** ❶ and **Create CORE\_SAMPLES** ❷ from the menu, as shown in Figure 5.7.

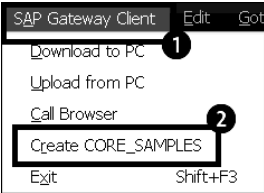


Figure 5.7 Creating Core Samples from within the SAP Gateway Client

If you’ve saved a request as a test case, you can add or change the expected HTTP return code. A request can return multiple HTTP return codes that are valid (e.g., 200, 401, 402, and 403). Therefore, multiple statuses, including status ranges separated by a dash, can be entered (e.g., 201 401-403). In addition, you can use payload validation so that the payload of an HTTP response can be compared with the expected result set and not only with the expected HTTP return code.

One or more test cases can then be run using the SAP Gateway client. The results are displayed in a table indicated by a traffic light icon together with the expected and actual HTTP return code.

Error Log

The error log is the second tool the developer will find especially useful when it comes to troubleshooting. The error log can be called using Transaction **/IWFND/ERROR\_LOG** in the SAP Gateway server system. There is also an SAP backend system error log with a similar UI available that can be used to analyze errors that occurred in the SAP backend system via Transaction **/IWBEP/ERROR\_LOG**.

The error log is tightly integrated with the SAP Gateway client, so you can re-run a request sent by a consumer that led to errors by selecting **Replay** • **SAP Gateway Client**, as shown in Figure 5.8.

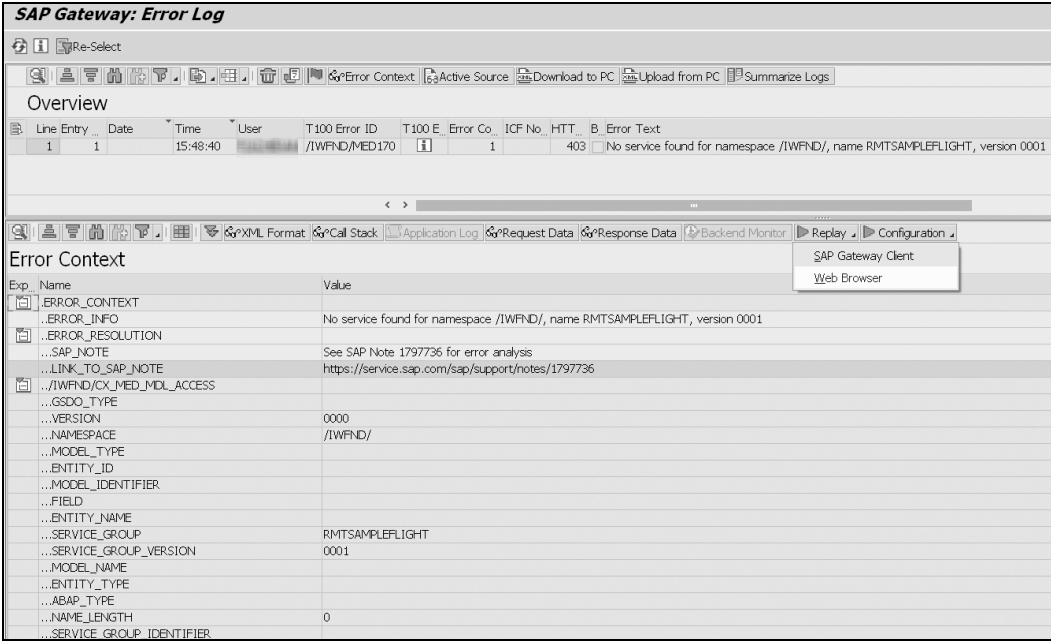


Figure 5.8 Transaction **/IWFND/ERROR\_LOG**

As another way to dig into potential problems, monitoring log entries can be generated for the system log and the application log of SAP Gateway. To access the system log, use Transaction **SM21**; to access the application log, use Transaction **/IWFND/APPS\_LOG**.

Logging and tracing

SAP Gateway Statistics and Payload Trace

When developing an OData service or a client application, a developer will want to know about the performance of the service. SAP performance statistics can be obtained by an OData client by adding **?sap-statistics=true** at the end of the request URL or by adding the HTTP request header **sap-statistics=true**. The SAP Gateway framework provides performance statistics data to the client in the HTTP response header **sap-statistics**. The response time data is also automatically stored by the SAP Gateway framework for every incoming OData request in the SAP Gateway server.

Performance statistics

Based on this data, Transaction **/IWFND/STATS** (SAP Gateway Statistics) provides a detailed statistics view of each service call handled by SAP Gateway. The data is aggregated on a regular basis so that statistical data for each service can be analyzed easily. In a productive system, this transaction



is of great value for system administrators to check the performance of OData services, as shown in Figure 5.9.

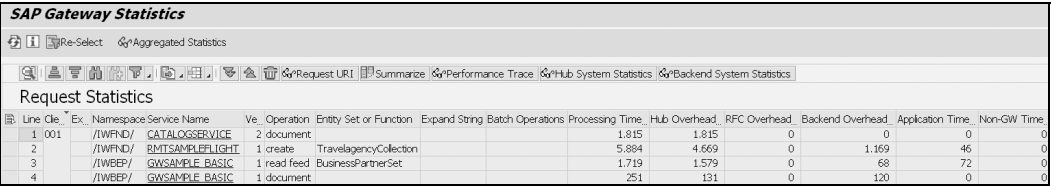


Figure 5.9 Transaction /IWFND/STATS

Via Transaction /IWFND/TRACES, you can trace not only system performance at the service call level for backend and hub systems, but also the payload of a request, as shown in Figure 5.10.

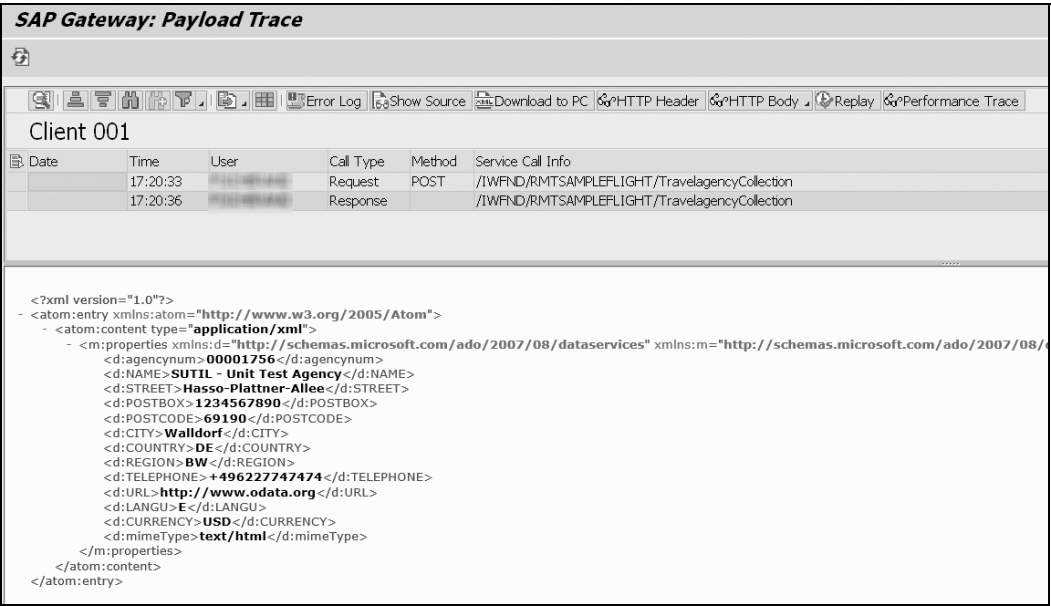


Figure 5.10 Transaction /IWFND/TRACES

Using the payload trace, you can also monitor the payload that is sent by the client and the data that the client receives as a response from the server. The traced data can also be used to replay service calls using the SAP Gateway client.

Payload trace to create test cases

With the replay capability, you can also conveniently create test cases in the SAP Gateway client for your service. To use the performance and payload trace, you must activate those traces.

We'll discuss SAP Gateway statistics and the Performance Trace tool in more detail in Appendix A, Section A.4.

Catalog Service

Each SAP Gateway system provides a catalog service that can be used to retrieve a list of all available services on SAP Gateway, as shown in Figure 5.11. The catalog service is an OData service, and the list of available services can be accessed via the following URL:

http://<server>:<port>/sap/opu/odata/iwfnd/CATALOGSERVICE;v=2/ServiceCollection



Figure 5.11 Service Catalog: Service Document

The catalog service supports OpenSearch. Developers or development tools are thus able to use a free-text search to find services based on the service description. You can search for services where “Product” is contained in the Description using the following URL: http://<server>:<port>/sap/opu/odata/iwfnd/CATALOGSERVICE;v=2/ServiceCollection?search=Product.

OpenSearch

5.3.3 ABAP Development Tools and CDS Views

A CDS view, as the name indicates, is a view that can be defined to retrieve an application-specific projection on the underlying business data. This kind of view may be needed because business data is usually distributed across several database tables.

CDS views: one concept, two flavors

CDS provide a specification for an SQL-based data definition language (DDL). With SAP HANA CDS and ABAP CDS, two flavors of this specification are available. Whereas SAP HANA CDS views only need to run on top of SAP

HANA, ABAP CDS views must support multiple databases. This distinction is similar to the ABAP Open SQL syntax, which is the least common denominator of all the different SQL dialects supported by SAP NetWeaver AS for ABAP.

**CDS view entities** As of SAP S/4HANA 2020, a new type of CDS view was introduced. These new CDS views are called *CDS view entities* and are recommended for use in new developments. Learn more in the blog post found at <http://s-prs.co/v575907>.

**Additional Resources**  
You'll find a comprehensive and detailed comparison between ABAP CDS views and SAP HANA CDS views at <http://s-prs.co/v575908>.

Listing 5.1 and Listing 5.2 show an example CDS view and CDS view entity, respectively, adapted to the Z namespace. We started with the ABAP CDS view found in the online documentation at <http://s-prs.co/v575909>.

```
@EndUserText.label: 'cust_book_entity'
@AbapCatalog.sqlViewName: 'ZCUSTOMER_VW'
define view zcust_book_entity
as select from scustom
join sbook on scustom.id = sbook.customid
{
  key scustom.id,
  scustom.name,
  sbook.bookid
}
```

Listing 5.1 Example of an ABAP CDS View

```
@EndUserText.label: 'cust_book_view_entity'
define view entity zcust_book_view_entity
as select from scustom
join sbook on scustom.id = sbook.customid
{
  key scustom.id,
  scustom.name,
  sbook.bookid
}
```

Listing 5.2 Example of an ABAP CDS View Entity

Both the CDS view `zcust_book_entity` and the CDS view entity `zcust_book_view_entity` create a join on the two database tables `scustom` and `sbook`, which are part of the SFLIGHT demo data model. As a result, we can access the data via the ABAP Open SQL statement shown in Listing 5.3.

```
SELECT id name bookid
FROM zcust_book_view_entity
INTO TABLE @DATA(result_data)
WHERE ... .

SELECT id name bookid
FROM zcust_book_entity
INTO TABLE @DATA(result_data_2)
WHERE ... .
```

Listing 5.3 Using ABAP CDS Views in ABAP Code

A CDS view can be defined using the Eclipse-based ADT with the ABAP CDS **ADT** statement `DEFINE VIEW`. This statement will create two objects in the ABAP DDIC—namely, an SQL view and the CDS entity, as shown in Figure 5.12.

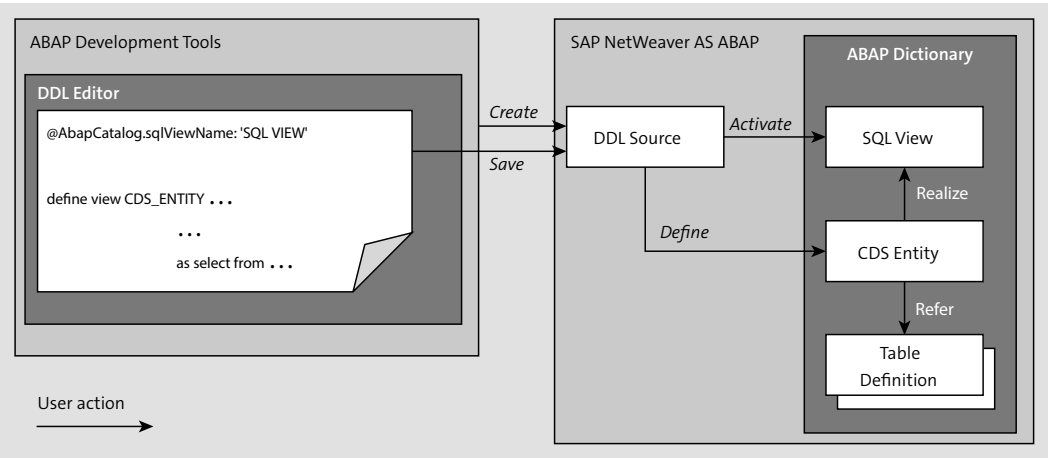


Figure 5.12 ABAP CDS View Building Architecture

**Note**  
The SQL view and the CDS entity are created in the same namespace in the ABAP DDIC. As a result, these names must be different. As shown in Listing 5.1, the SQL view is denoted as `CUSTOMER_VW`, whereas the CDS entity is denoted as `cust_book_entity`.

5.4 Steps in the Service Creation Process

In Section 5.2, we introduced the SAP Gateway service creation process, which consists of three phases: data model definition, service implementation, and service maintenance. You can take different tracks for creating your services, depending on whether you go for service development or service generation. Now let’s take a closer, more technical look at the different tracks and the individual steps in these tracks. Due to the various options for creating SAP Gateway services, you may find it useful to refer to the diagram shown earlier in Figure 5.2 throughout this section.

5.4.1 Data Model Definition in the Service Builder

The first phase of the service creation process is the data model definition phase. The goal of this phase is to use the Service Builder to create a data model that contains all information about the OData model of a service, such as entity types, complex types, properties, and associations. So, when developing an SAP Gateway service (service development) or when generating an SAP Gateway service by mapping a data source (one specific type of service generation), the first main process step is to create a data model.

Note

When using the second method of service generation, which is to redefine an existing service, the data model isn’t defined but rather *redefined* based on the existing business objects. For information about that kind of data model building, see Section 5.4.5.

You can define a data model in several ways with the Service Builder, each of which addresses a specific use case.

Four options for defining an OData model

The first option is the manual creation of the various components of an OData model, which is called a *declarative model definition*. Entity types, associations, and association sets in this approach are created manually.

The second option is the import of data models in the entity data model XML (EDMX) format, the service metadata document of an existing OData service.

The third and fourth options, which are much more convenient for an ABAP developer, are to create entity types by reusing data models that already exist in the SAP backend system. This reuse can be achieved by importing DDIC structures/tables or by generating new entity types based on an RFC/BOR interface or a search help.

Next, we’ll discuss all these options in a bit more detail.

Declarative Data Model

A declarative data model is created manually using the Service Builder. This method is mainly used to create entity types based on manually created properties, which can be based on existing DDIC types.

Entity types

Import Data Model via EDMX

Using the import model option, the developer can import, into the Service Builder, either a complete OData model stored in an EDMX file or a meta-data document of an existing OData service. This import includes the definition of entity types, entity sets, associations, and other components. You can import data model files that have been defined by graphical OData modeling tools or by the service metadata files of an existing OData service. If you import a service metadata document or an EDMX file for an existing project into the Service Builder, the Service Builder provides the option to reimport the data model files. A dialog box will appear that shows which artifacts will be added to and which will be deleted from the data model.

Import Data Model via the Data Dictionary

To reduce the time required to create entity types and complex types in your data model and to leverage existing data structures in your SAP backend system, you can import the following DDIC types into the Service Builder:

DDIC type support

- Views
- Database tables
- Structures

Beautification

When creating an entity type from a DDIC type, the name of the entity type and the names of the properties of the entity type suggested by the Service Builder are derived from the original names of the DDIC type and its fields by removing the underscores and generating a name with camel case notation instead. For example, when using a structure such as BAPI\_EPM\_PRODUCT\_HEADER, the Service Builder will propose the name BapiEpmProductHeader for the entity type. The same naming convention for proposals is used for the property names of the generated entity type—so that instead of the original field name SUPPLIER\_NAME, the field name of the generated entity type becomes SupplierName.

The name of the entity set and its properties should be easy to understand because they are visible to the consumer, and the names of the properties of an entity set are derived from the property names of the underlying entity type.

During the process of importing a DDIC structure or even afterward, a developer can start a process called *beautification*. Through this process, you can reduce the number of properties of an entity type by simply removing single properties from it. In addition, you can maintain the names of the properties of an entity type.

Reducing the number of properties to those that are absolutely necessary and maintaining the names that are visible to the outside world are important for creating services that are easy to consume. Publishing existing DDIC structures as is to the outside world is usually not beneficial.

Beautification is discussed in more detail in Chapter 7, Section 7.4.1.

Import Data Model via RFC/BOR

Function  
module and BAPI  
parameters

The Service Builder also enables you to create entity types from function module parameters and BAPI parameters. A wizard is provided to guide you through the process. Using the interface of an RFC function module or a BOR interface is beneficial if you are using them to access the data in an SAP backend system. Both code-based implementation and using the RFC/BOR Generator are possible with this approach.

Import Data Model via Search Help

Finally, the Service Builder also allows you to create entity types from the search help. Again, a wizard is provided to guide you through the process. This wizard even performs the mapping of the READ and QUERY methods in the same step so that a separate service implementation step is no longer necessary.

5.4.2 Service Registration in the SAP Backend System

After the data model is defined, it must then be registered. Service registration in the SAP backend system manifests the data model definition phase’s results. Thus, the runtime objects required for an SAP Gateway service are generated using the Service Builder. For the convenience of the developer, the Service Builder also performs the necessary tasks to register the service in the SAP backend system (SAP S/4HANA or SAP Business Suite).

Service Registration versus Service Maintenance

As described earlier in Section 5.2, the service maintenance phase of service creation involves activating and registering the service on the SAP Gateway server. Don’t confuse this task with service registration in the SAP backend system, which is a process that occurs after the data model definition. In this section, we’re focusing on service registration in the SAP backend system. In Section 5.4.4, we’ll discuss service maintenance.

The difference between service registration and service maintenance can be summarized in the following way:

- *Service registration* is an activity during service development that results in the creation of artifacts needed for development.
- *Service maintenance* is an activity during the deployment/operation of an SAP Gateway service that activates the service for consumption.

Based on the data model that has been created, the Service Builder generates a corresponding MPC and DPC, as well as extension classes. The MPC contains the coding that programmatically declares the data model being used by your service. The implementation of the service operations is performed in the DPC. The extension classes that have been generated by the Service Builder must be used to redefine methods of the generated base classes by custom code because the base classes are always regenerated when the model has been changed. (For more information on MPC and DPC, see Section 5.5.)

Stub class creation

To be used as a service, some configuration steps must be performed, which are supported by the Service Builder, as shown in Figure 5.13.

Service registration

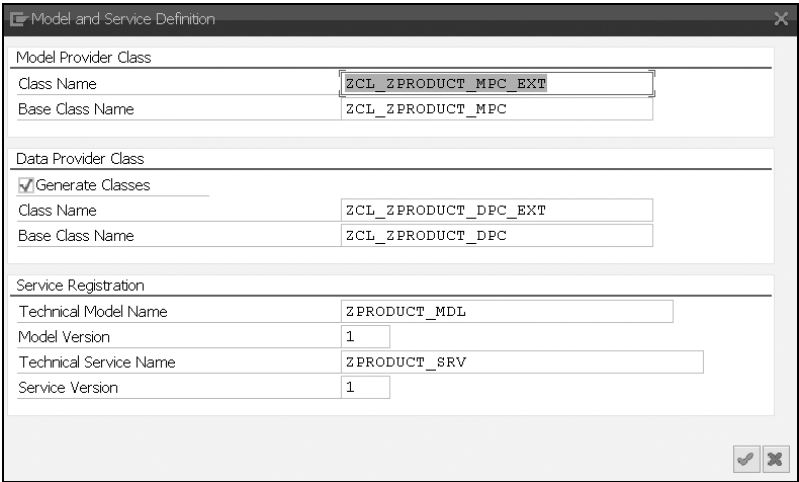


Figure 5.13 Model and Service Definition Using the Service Builder



When generating a project for the first time, the developer must specify the names of the MPC and its extension class and the DPC and its extension class. In addition, the developer must specify the **Technical Model Name** and the **Technical Service Name**. The latter becomes the external service name that is later used for publishing the service on the SAP Gateway.

**MPC and DPC** The MPC and the DPC are thus combined into an SAP Gateway service by means of configuration, not coding. These configuration steps are facilitated for you by the Service Builder when the project is generated for the first time. The model and service definition process is shown in Figure 5.14. In addition to the MPC and DPC (covered in detail in Section 5.5.1 and Section 5.5.2, respectively), two additional repository objects for the model and the service are created as part of the registration process of a service in the SAP backend system.

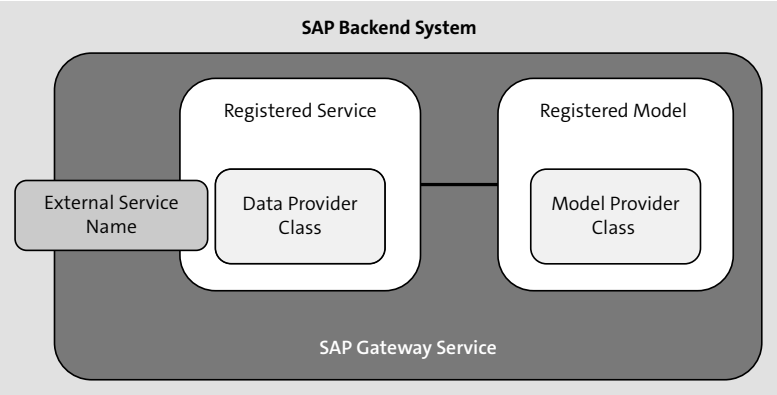


Figure 5.14 Register Service and Model

5.4.3 Service Implementation

During the service implementation phase of the service creation process, operations that should be supported by the SAP Gateway services are implemented via ABAP code or by mapping the methods of a data source on the properties of an OData model. Operations are executed on the defined data model during runtime and encompass CRUD-Q operations when using RFC function modules or BAPIs, or they are limited to READ and QUERY when using the search help or CDS views.

**Data source mapping** Note that the service implementation phase applies only to service development and to one of the service generation options: data source mapping. For service generation using redefinition or by referencing a CDS view as a data source, the service implementation step isn't immediately necessary because the implementation of the service will be generated or handled dynamically based on the customizing that has been performed in the

model definition step. It can, however, be performed at a later stage to add additional functionality such as the support of Create, Update, and Delete to services that have been generated based on CDS views.

**Note**

We provide an introduction to service generation using redefinition in Section 5.4.5 and to service generation by referencing a CDS view as a data source in Section 5.4.6.

Next, we'll provide a brief overview of the service implementation phase for both scenarios where the phase is relevant: service development and service generation via data source mapping.

Implementation for Service Development

Remember that, during the service registration of the data model definition phase, a data provider extension class was created. Also, during the service implementation phase, operations that are to be supported by the SAP Gateway services are being implemented.

To implement the supported SAP Gateway services using ABAP coding, you must manually redefine the respective methods of the data provider extension class, which should remind you of the CRUD-Q operations:

- <ENTITY\_SET\_NAME>\_CREATE\_ENTITY
- <ENTITY\_SET\_NAME>\_GET\_ENTITY
- <ENTITY\_SET\_NAME>\_UPDATE\_ENTITY
- <ENTITY\_SET\_NAME>\_DELETE\_ENTITY
- <ENTITY\_SET\_NAME>\_GET\_ENTITYSET

Access to these methods is quite convenient in the Service Builder. Simply expand the service implementation node, as shown in Figure 5.15.

Expand CRUD-Q methods

Then, you can navigate to the relevant entry of an entity set, expanding all CRUD-Q methods of an entity set. Selecting **Go to ABAP Workbench** allows the developer to switch seamlessly to the Class Builder (Transaction SE24) to implement an operation. If Transaction SEGW is started from within ADT (also known as ABAP in Eclipse), the Eclipse-based editor can be used instead.

In addition, you might need to redefine additional methods in the data provider extension class that aren't specific to an entity set such as the CRUD-Q methods mentioned earlier (e.g., if deep inserts should be supported by the OData service).

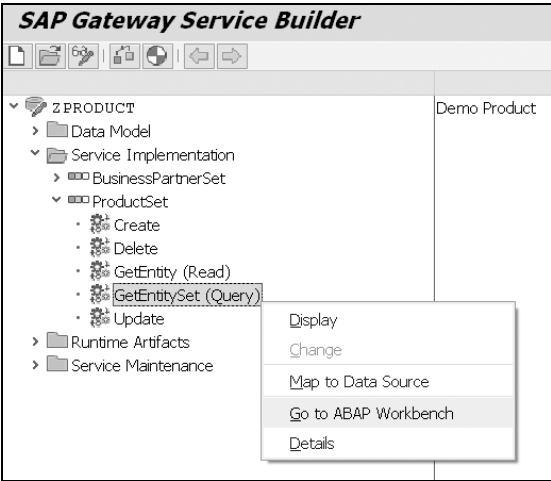


Figure 5.15 Code-Based Implementation

Implementation for Mapping RFC/BOR Interfaces

The process of implementation for mapping RFC/BOR interfaces is different from the process of service development. To start the mapping process, you must select **Map to Data Source** in the context menu of a CRUD-Q method of an entity set in the **Service Implementation** folder, as shown in Figure 5.16. The mapping dialog of the Service Builder then allows you to define relations between the interface parameters of a function module or BAPI and the properties of an entity set.

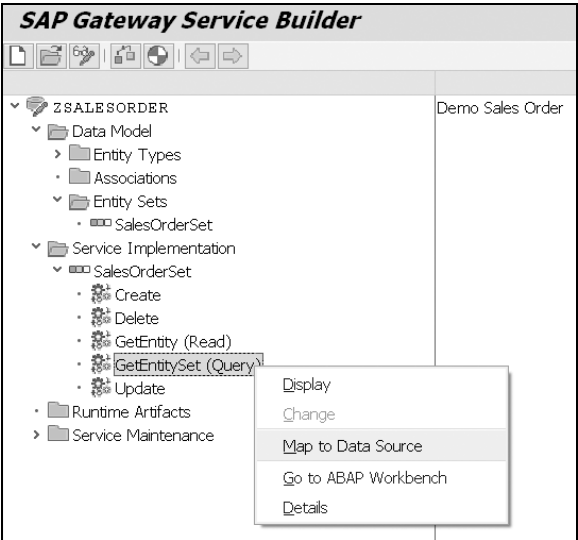


Figure 5.16 Mapping the Methods of an Entity Set to a Data Source

You can map CRUD-Q methods of each entity set separately. The actual service implementation (i.e., the coding in the CRUD-Q methods mentioned earlier) will be generated by the Service Builder based on the mapping you’ve performed. The Service Builder supports the developer by providing mapping proposals if the entity type has been created by importing a BOR interface or an RFC interface. Clicking the **Propose Mapping** button, shown in Figure 5.17, which results in the list of mapping proposals. For example, as shown in Figure 5.17, the Service Builder suggested a mapping between the `SoId` property in the `SalesOrderSet` entity set and the `SO_ID` property of the `SOHEADERDATA` export parameter of the `BAPI_EPM_SO_GET_LIST` BAPI. This mapping can automatically be suggested because the entity type on which the `SalesOrderSet` entity set is based has been created by importing the `SOHEADERDATA` interface parameter.

CRUD-Q

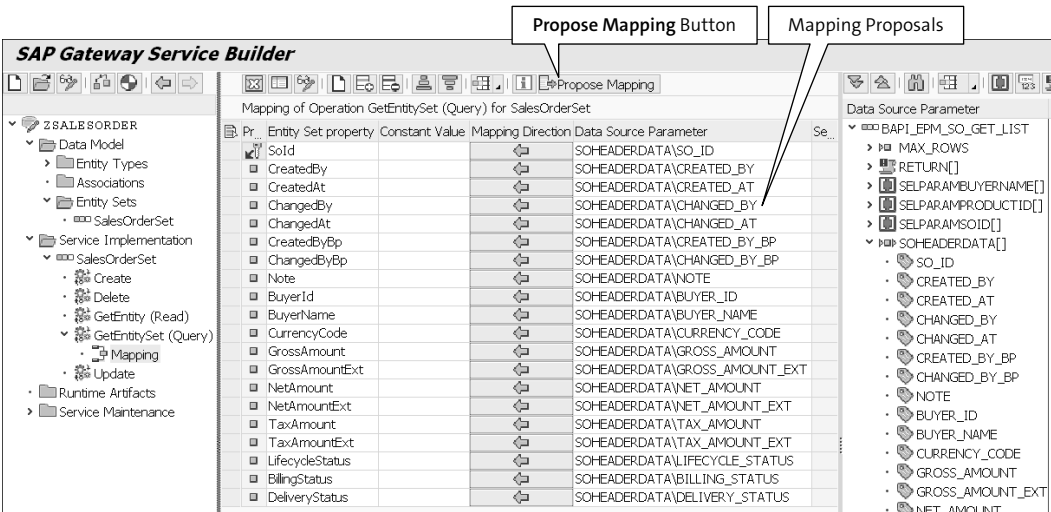


Figure 5.17 Mapping Proposals: RFC Function Module

If additional methods for the entity sets are mapped, the Service Builder checks the already existing mappings and derives proposals for them. If you, for example, started to map the `QUERY` operation (`GET_ENTITYSET`) of your entity set and now want to map the `READ` operation (`GET_ENTITY`), the Service Builder provides a proposal for those properties that have already been mapped in the `GET_ENTITYSET` method.

Implementation for Mapping CDS Views

The implementation process for mapping CDS views is different from that of mapping RFC/BOR interfaces. To start the mapping process, you must select **Map to Data Source** in the context menu of an entity set in the

Leveraging CDS views in SAP NetWeaver

**Service Implementation** folder, rather than selecting individual CRUD-Q methods.

CDS mapping is the only way to leverage CDS views in SAP NetWeaver 7.40. As of SAP NetWeaver 7.50, the option to reference a CDS view as a data source should be used. Both options are discussed in more detail in Chapter 8.

The mapping dialog in the Service Builder then allows you to define relations between the data source elements of a CDS view and the properties of an entity set, as shown in Figure 5.18; it also allows you to map an association of a CDS view to a navigation property of an entity set, as shown in Figure 5.19.

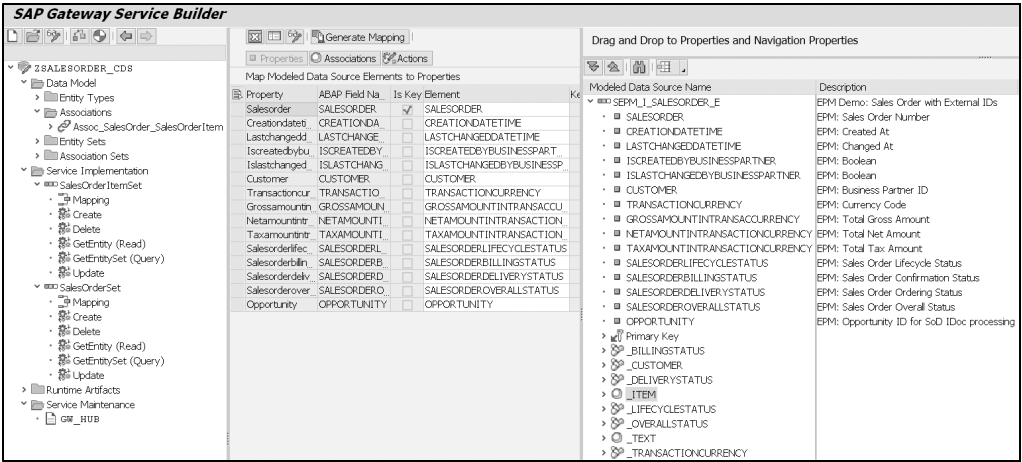


Figure 5.18 Mapping a CDS View: Properties

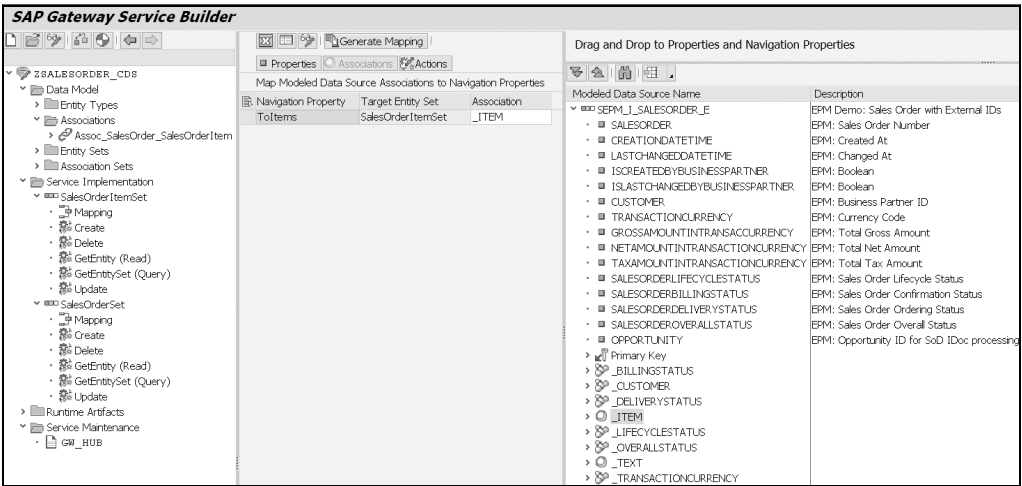


Figure 5.19 Mapping a CDS View: Association to Navigation Property

As a result, the READ and QUERY methods of an entity set are mapped. The implementation of CREATE, UPDATE, and DELETE methods (CUD) is still possible via a code-based implementation or via mapping of appropriate RFC function modules to the CUD methods.

Implementation for Mapping Search Help

The implementation for mapping a search help is even easier than mapping RFC/BOR interfaces or CDS views. This is already included in the data model definition step when creating an entity type based on a search help. The wizard that is used to import a search help not only offers to create an entity set but also already performs the mapping of the **Read** and **Query** method, as shown in Figure 5.20.

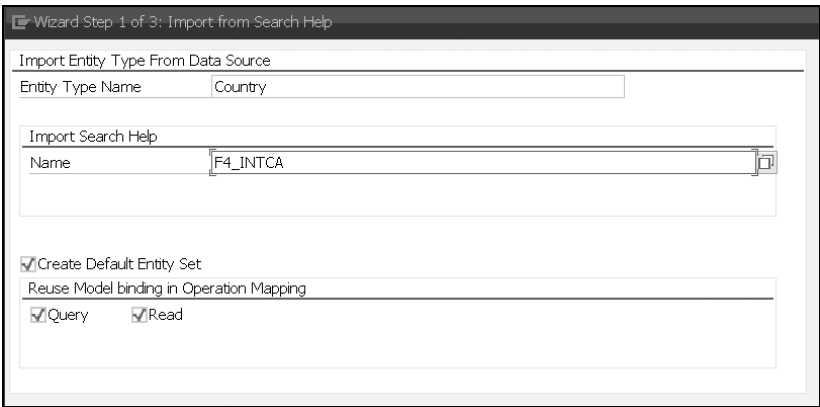


Figure 5.20 Import Search Help Wizard: Automatic Mapping of Query and Read Methods

As with entity sets, where the service implementation is based on CDS views, the implementation of the CUD methods can be performed via a code-based implementation or via mapping of RFC function modules that offer write access.

5.4.4 Service Maintenance

The service maintenance phase primarily consists of the service activation and service registration step in the SAP Gateway system. For SAP Gateway to enable the consumption of a service using an OData client, this service must be activated. This activation takes place in the SAP Gateway server and makes the service ready for consumption.

The registration and activation of services in the hub is performed using Transaction /IWFND/MAINT\_SERVICE (Activate and Maintain Service) is

Activate and maintain service

also used to maintain all activated services on the SAP Gateway server. Services must be changed if they've been registered in several/additional connected SAP backend systems, or they can simply be deactivated.

Because the Service Builder is a one-stop shop for service development, functionality has been added that allows developer to directly call a transaction for service maintenance from within the Service Builder. This access is even possible for remote systems.

A developer can either select an SAP Gateway system in the **Service Maintenance** node, as shown in Figure 5.21, or click the **Register** button.

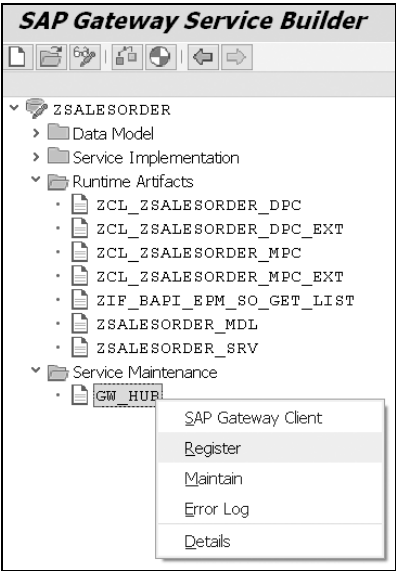


Figure 5.21 Registering a Service in the Hub from the SAP Backend

Service Generation

As outlined earlier in Section 5.1, when performing service generation via redefinition, RDS, or using Eclipse to create CDS views with the `publish:true` option, there is no service implementation step—only the data modeling phase—and the service can be published afterward.

5.4.5 Service Generation via Redefinition

As described in Section 5.2, *redefinition* is the process of generating a service based on an existing data source. This is done using a wizard and combines both the data model definition phase and the service implementation phase into the single phase of redefinition. The resulting generated service

must be registered and activated in the SAP Gateway server system (the service maintenance phase) and can then be consumed. The goal of redefinition is to reduce the effort required for service creation.

A number of business objects exist in an SAP system. SAP Customer Relationship Management (SAP CRM), SAP Product Lifecycle Management (SAP PLM), and SAP Enterprise Asset Management (SAP EAM), for example, all use some form of business object. Although these business object models have been designed for different use cases, they all define objects, relations, actions, and queries similar to objects found in the OData protocol. Unsurprisingly then, many of these business objects can be used to generate OData services.

You can also generate SAP Gateway services from existing SAP Gateway services. This scenario arises, for example, if a customer wants to extend an OData service delivered by SAP—perhaps an OData service used by an SAP Fiori application. The extensibility of SAP Fiori applications is discussed fully in Chapter 12.

On top of integrating existing SAP backend business objects, you can also integrate third-party OData services. However, this integration scenario has several technical restrictions; thus, the OData Services Consumption and Integration (OSCI) approach is no longer recommended. We'll describe the alternative, using the *OData client proxy* available as of SAP NetWeaver 7.52, later in this section.

The wizard for generating an OData service via redefinition is almost identical for all integration scenarios. Selecting one of the available options (based on the installed add-on) starts a wizard that guides you through the following three steps:

- 1. Select the business object.
- 2. Select artifacts of the data source (data model definition).
- 3. Generate runtime artifacts and service registration in the backend (service implementation).

In other words, the wizard starts with the data model definition phase but automatically performs the steps that belong to the service implementation phase. After the service has been registered and implemented in an SAP backend, the service must be activated in the SAP Gateway server.

The different integration scenarios described in this section are partly based on the specific add-ons listed in Table 5.1. If these add-ons have been deployed to an SAP backend system, the related context menu options in the Service Builder will be visible, as shown in Figure 5.22.

Existing business objects

Extensibility

Third-party OData services

Redefinition wizard

Add-ons



Name of Add-On	Integration Scenario	Remote Enabled	Support in SAP S/4HANA
IW_GIL	GenIL		Only up to SAP S/4HANA 2022
IW_SPI	SPI	X	No
SAP_GWFND or IW_BEP	Analytical queries	X	Yes
SAP_GWFND or IW_BEP and IW_FND	OData service (external)	X	Yes (but no longer recommend)
SAP_GWFND or IW_BEP	OData service (SAP Gateway)	X	Yes

Table 5.1 Add-Ons for Generating a Service Based on an Existing Data Source

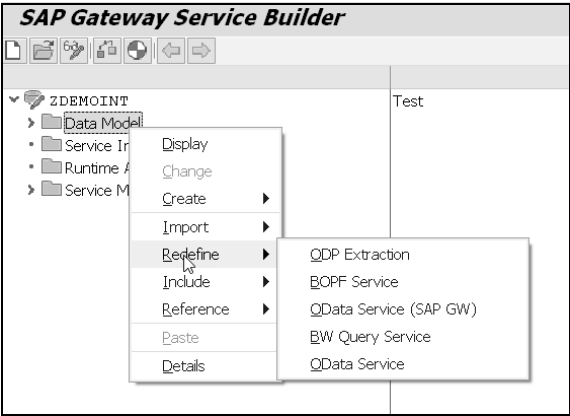


Figure 5.22 Context Menu Options to Create a Data Model Using Redefinition

Most of these scenarios are also remote enabled, which means that the business object to be consumed (e.g., an SPI object) doesn’t need to exist in the same system in which the BEP component is deployed. As a result, these scenarios can be implemented in the SAP Gateway server (assuming you’re using hub deployment with development on the hub).

Support in SAP S/4HANA

The IW\_SPI add-on is not supported for SAP S/4HANA and was thus made uninstalleable. IW\_GIL is only supported up to SAP S/4HANA 2022 and is uninstalleable, as described in the following SAP Notes:

- SAP Note 2319114 – Uninstallation of the add-on IW\_GIL 100 using Transaction SAINT
- SAP Note 2313222 – Uninstallation of the add-on IW\_SPI 100 using Transaction SAINT

Next, let’s look at the different possible sources for suitable business objects in more detail.

Generic Interaction Layer

The integration of GenIL with SAP Gateway enables the generation of OData services based on existing GenIL components. GenIL is intended to serve as a wrapper around existing business logic. It provides access to all business objects via a unified interface for consuming application logic in the UI layer by using the *Business Object Layer (BOL) API*. The BOL consists of two pieces:

- **GenIL**  
The lower layer is a “dispatcher” that manages GenIL components and their models at runtime and distributes requests from above to the respective components implementing the requested objects.
- **BOL**  
The stateful layer provides optimized performance by avoiding expensive repetitive access to the APIs, thus acting as a buffer for the UI.

Wrapper around existing business logic

While the BOL was built for the SAP CRM WebClient, the role of GenIL is different because it can be used for other integration scenarios as well. The consumption of Simple Object Access Protocol (SOAP)-based web services using the Web Service tool that directly consumes GenIL is an example of such additional integration.

BOL versus GenIL

Similarly, SAP Gateway allows you to generate OData services leveraging GenIL, as shown in Figure 5.23. The nodes, relations, and queries in the GenIL model are transformed to the corresponding entities in an OData model, as shown in Figure 5.24.

Although BOL (and thus GenIL) is frequently used for SAP CRM WebClient, it has also been used in other SAP Business Suite applications, such as SAP ERP Financials and SAP ERP Human Capital Management (SAP ERP HCM). The integration is contained in the IW\_GIL add-on, which must be deployed locally on an SAP Business Suite system (e.g., SAP CRM) on top of the BEP component.

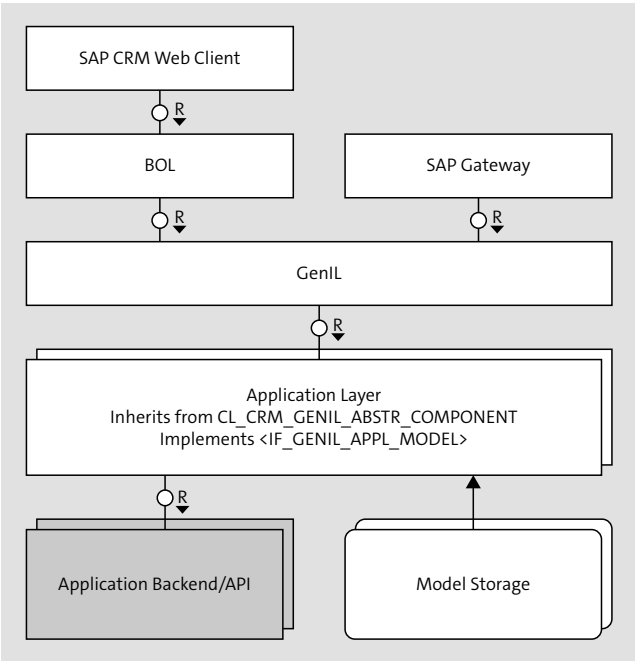


Figure 5.23 Integration of GenIL with SAP Gateway

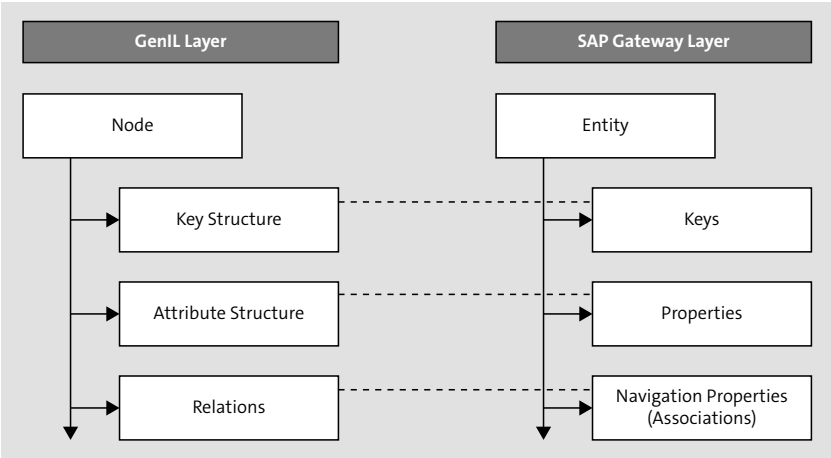


Figure 5.24 Mapping between the GenIL and OData Model

**Note**

The GenIL integration scenario isn't remote enabled. To use services that are generated based on GenIL objects, the IW\_BEP add-on component must be deployed on SAP Business Suite systems running on SAP ERP 6.0 EHP 6 or earlier.

**Service Provider Interface**

The SPI was originally developed for SAP PLM. The SPI is a framework generated within the application layer that has different consumers. The framework is currently used not only by the applications for which it was originally developed but also for various other applications within the SAP Business Suite.

SPI objects can be called remotely. As a result, it isn't mandatory to deploy the SAP Gateway IW\_SPI add-on for the SPI on the SAP Business Suite system. Because the add-on calls the RFC interface of the SPI layer, it can be deployed on the SAP Gateway server system. The IW\_GIL add-on instead must be deployed locally on an SAP Business Suite system (e.g., SAP CRM). The integration of an SPI with SAP Gateway allows SPI application building blocks to be provisioned as OData services. Note that the IW\_SPI add-on is not supported for SAP S/4HANA.

**Further Resources**

For more information about this topic, we recommend the following links:

- SPI wiki: <https://wiki.scn.sap.com/wiki/display/SPI>
- SAP online help: <http://s-prs.co/v575910>

**External OData Service**

Since SAP S/4HANA 2021, we recommend using the *OData client proxy* for the integration of external OData services instead of OSCI. The OData client proxy is the public API for ABAP-based client applications that should call an external OData service to consume its business data. The OData client proxy provides OData request details (such as the entity set to be read with which filter) using the client request object and OData response data using the client response object. In a Transaction SEGW project using code-based implementation, it can be used to implement all relevant methods such as `get_entityset`, `get_entity`, `create_entity`, and so on.

Though this approach requires more effort than the service generation using OSCI, there are less technical restrictions, and the implementation can be much more flexible.

The OData client proxy can also be used with the ABAP RESTful application programming model to build unmanaged, transactional ABAP RESTful application programming model business objects that use unmanaged queries (custom entities). Alternatively, you can use managed, transactional

OData client proxy

ABAP RESTful application programming model business objects that use external OData services for value helps, validations, or determinations.

**OSCI** Prior to the OData client proxy, OSCI was an additional integration scenario for enabling the consumption and integration of any OData service. Note that OSCI integration had several technical restrictions, as described in SAP Note 1574568, and is no longer recommended.

**OData Service (SAP Gateway)**

The Service Builder allows you to generate a service based on an existing OData service in SAP Gateway. This integration scenario can be used to extend an existing service. It creates a new service with the same interface as the original service but with a changed behavior, which is accomplished by redefining methods in the new DPC extension class. The extension of an OData service and an SAPUI5 application delivered by SAP as part of the SAP Fiori reference apps is discussed in detail in Chapter 12.

**5.4.6 Service Generation via Referenced Data Sources**

**Data provisioning  
in SAP S/4HANA**

The advent of SAP HANA was a paradigm shift in how business applications were developed at SAP following the ABAP programming model for SAP Fiori. Data provisioning in SAP S/4HANA is based on CDS and OData. This is possible because CDS not only addresses read-only scenarios but also transactional, analytical, and search use cases. Using CDS, it's possible to define semantically rich data models by providing annotations that can be leveraged by SAP Fiori elements (also known as smart templates). These are smart in the sense that the UI will provide an input field automatically if a property is marked as `sap:updatable`. CDS views can easily be extended by extending the view. The **Referenced Data Source** option allows ABAP developers to define dynamic OData services based on CDS view definitions in Transaction SEGW, as shown in Figure 5.25.

As a result, any change in the underlying CDS view is automatically reflected in the OData service that has been generated using the RDS concept. In the Service Builder, you can select a CDS view and select those entities and associations that should be part of the OData service.

Features that are not yet supported via CDS views can be implemented via custom logic in the model provider and the DPC extension class.

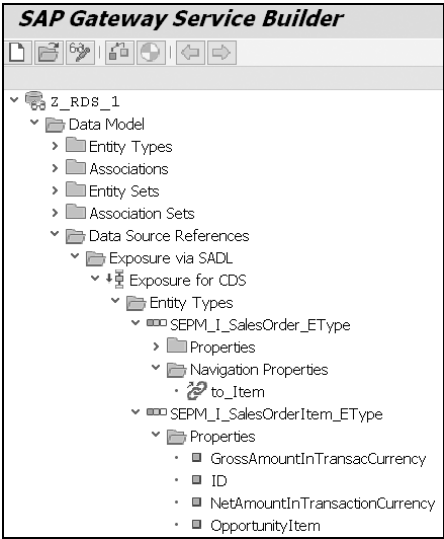


Figure 5.25 CDS View as an RDS in Transaction SEGW

**5.4.7 Service Generation via OData.publish:true**

Similar to the RDS approach, `OData.publish:true` allows you to publish CDS views as OData services directly from within ADT in Eclipse. By setting one simple annotation (`@OData.publish:true`), you can publish a CDS view as an OData service. Technically, an MPC and a DPC are generated, and these classes are registered as an OData service in the SAP backend. To publish the registered service, a developer or administrator must use Transaction /IWFND/MAINT\_SERVICE. In contrast to all the other options for creating OData services that we've shown thus far, this option doesn't make use of the Service Builder.

The `@OData.publish:true` option isn't only suitable for read-only and analytical services; it can also be used for transactional services by generating appropriate BOPF objects alongside the OData service, which are an essential part of the ABAP programming model for SAP Fiori. By performing a code-based implementation of those objects, the generated OData service does also support the capability to Create, Update, and Delete business data.

Which option to use to generate an OData service from a CDS view is also described in the SAP Help Portal at the following URL: <http://s-prs.co/v575911>.

In short, the RDS approach offers various features that are not available via the `@OData.publish:true` option.

**Transactional  
services**

5.4.8 Service Generation for Analytical Queries

Analytical queries are the main tools for consuming analytical data that is embedded in business applications such as SAP S/4HANA or SAP Business Suite and in data warehouses such as SAP BW/4HANA. While analytical queries in SAP S/4HANA or SAP Business Suite provide access to consistent operational data, analytical queries in the SAP BW hub offer access to consistent, highly aggregated data across the enterprise.

SAP Gateway and SAP BW integration allow you to publish SAP BW content as an OData service. Three options exist for building such services:

SAP BW Easy Queries

■ SAP BW Easy Queries

The oldest option is to publish the analytical query as an SAP BW Easy Query, which was invented with a focus on SOAP services with SAP NetWeaver 7.30. You had to redefine an SAP BW Easy Query using Transaction SEGW and to perform several manual steps in addition.

This option is currently in maintenance and should not be adopted if your system is based on SAP NetWeaver 7.40 or later, as described in SAP Note 3154424.

OData query

■ OData query

OData query is the successor to SAP BW Easy Queries and is available as of SAP NetWeaver 7.40 SP 06. Compared to the approach based on SAP BW Easy Queries, it provides several advantages such as improved performance and a broader support of features relevant for analytical queries. Also, lifecycle management is improved since, unlike SAP BW Easy Queries, the OData service is generated together with the OData query automatically, and thus, you no longer need to create the service in a separate manual step with Transaction SEGW or Transaction /IWBEP/ANA\_SRV\_GEN. To release an analytical query as an OData service, you must select the corresponding **By OData** checkbox in the query properties in the Query Designer, as shown in Figure 5.26. Then, once the query is saved, the generation process for an OData service is triggered.

■ CDS views

The third approach uses CDS views. Creating ABAP CDS views with the annotation `@Analytics.query: true` creates a transient analytical query, and the annotation `@OData.publish: true` automatically generates a CDS-based OData service, as described in Section 5.4.7.

OData CDS query

General rules that apply for OData queries are that characteristics are on the rows, key figures are on the columns, and free characteristics aren't mapped to OData.

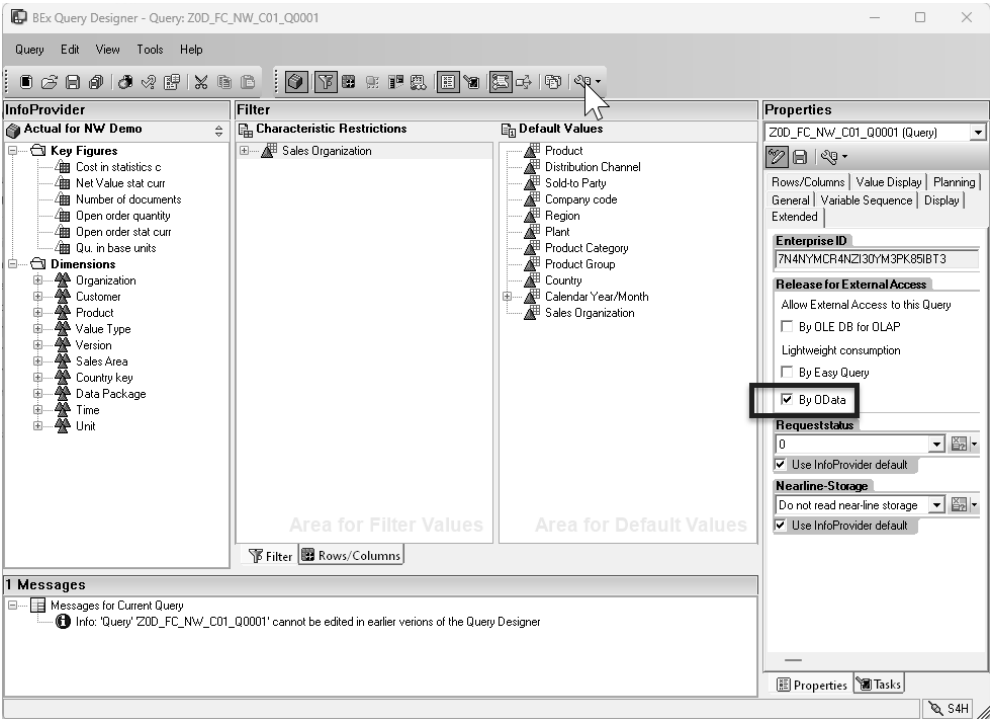


Figure 5.26 Defining a Query in the Query Designer

Dimensions, dimension attributes, and measures are represented as properties of an entity type. The entity type representing the results of an OData query is annotated as `sap:semantics=aggregate`. Focusing only on the main annotations, Table 5.2 shows how SAP BW objects such as dimensions, dimension attributes, and measures are represented in OData.

Analytical annotations

SAP BW Object	OData Representation	SAP Annotation
Cube of type query	Entity type	sap:semantics=aggregate
Dimension	Property	sap:aggregation-role=dimension
Dimension attribute	Property	sap:attribute-for=<dimension name>
Measure	Property	sap:aggregation-role=measure

Table 5.2 Analytical Annotations

A comparison of all three options can be found in the SAP Help Portal at <http://s-prs.co/v575912>.



5.5 OData Channel Development Paradigm

Now that we’ve discussed the basics of the different tracks for the SAP Gateway service creation process, let’s look a little more closely at the *OData channel development paradigm*, which describes the programming model used by the SAP Gateway runtime. This introduction lays the theoretical foundation for Chapter 6, which goes into more detail about service development.

The OData channel for SAP Gateway allows you to develop content by defining object models and registering a corresponding runtime DPC. The advantage of the OData channel paradigm is a certain freedom with respect to development; entire DDIC definitions and local interfaces of the SAP backend system can be used to develop SAP Gateway services. In addition, OData query options can be leveraged in your SAP backend systems so that only data that has been requested by the client is selected from the SAP backend system and sent back over the wire. This specificity results in highly optimized services and major performance improvements due to smaller data transfers.

Four components of an SAP Gateway service

With respect to the OData programming model, SAP Gateway services consist of four components:

- **MPC**  
Implemented to provide the runtime representation of your model definition.
- **DPC**  
Called at runtime to perform data requests.
- **Technical service name**  
Used to register the service in the SAP backend system together with the *technical model name*.
- **Technical model name**  
Used to register the service in the SAP backend system together with the *technical service name*.

The technical service name and technical model name are automatically generated with the MPC and DPC when generating a project using the Service Builder.

5.5.1 Model Provider Class

The MPC is an ABAP class that provides the runtime representation of your model definition—that is, the MPC defines the EDM of a service. As such, all model information that you’ve defined in your project is generated into

the MPC. Therefore, you must regenerate the MPC every time you change the model definition in your project. The MPC is important because everything you find in the service metadata document of an OData service published via SAP Gateway has programmatically been defined in the MPC.

Technically, the model definition is generated into two classes:

- **Base class**  
Technically, the base class is derived from the `/IWBEP/CL_MGW_PUSH_ABS_MODEL` superclass and has the suffix `_MPC`.
- **Extension class**  
The extension class has the base class as the superclass and has the suffix `_MPC_EXT`. The extension class will be registered via the technical model name. In the extension class, you can choose which methods to redefine and which methods to inherit from the base class.

In most cases, a developer doesn’t need to touch the model provider extension class with the suffix `_MPC_EXT` as generated by the Service Builder. An exception to this rule is, for example, if you want to build SAP Gateway services with features that can’t (yet) be modeled using SAP Gateway tools. In this case, a developer can redefine methods in the model provider extension class. Figure 5.27 shows the model provider base class at the top, which inherits from `/IWBEP/CL_MGW_PUSH_ABS_MODEL`. `ZCL_ZPRODUCT_MPC_EXT` is the subclass of the model provider class. The **DEFINE** method is overwritten.

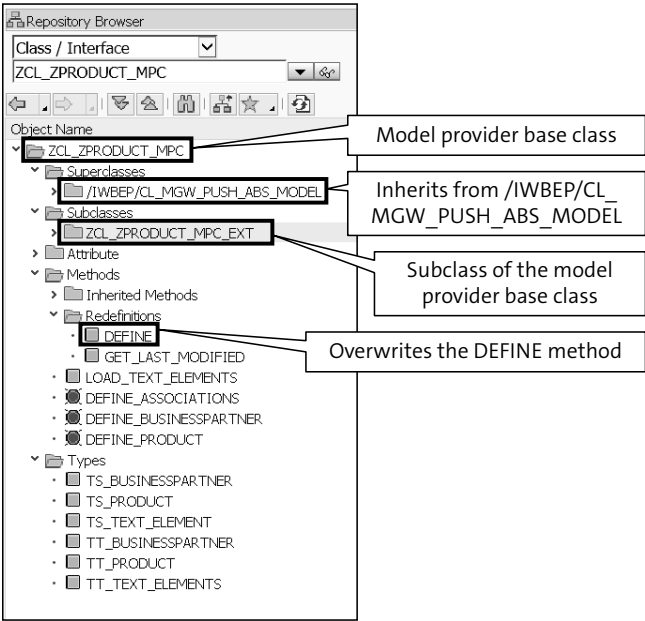


Figure 5.27 Model Provider Class Interface

Model Provider Class Deep Dive

Usually, a developer doesn't need to tap into the coding of the MPC being generated by the Service Builder. But let's still take a closer look at the methods being generated to get a better understanding of the underlying framework.

The GET\_LAST\_MODIFIED method is the basis for a handshake between the SAP backend system and SAP Gateway to start a refresh of the cached metadata of the service on the SAP Gateway backend and the SAP Gateway server after the class has been changed. This method shouldn't be changed manually.

In the entity type-specific DEFINE methods, the Service Builder generates the coding that creates the parts of the OData model that define the entity types and the entity sets that are based on entity type. The properties are created, and those properties that have been marked as a key field in the Service Builder are set as key fields in the coding:

```
lo_property = lo_entity_type->
create_property( iv_property_name = 'ProductID'
iv_abap_fieldname = 'PRODUCT_ID' ).
lo_property->set_is_key( ).
```

Finally, the entity type is bound to a DDIC structure, and one or more entity sets are created. Note that an entity type that is bound to an existing DDIC structure can leverage conversion exits as well as the labels of the data elements from the DDIC. The medium field label of a data element is used as sap:label by default:

```
...
lo_entity_type->
bind_structure( iv_structure_name =
'BAPI_EPM_PRODUCT_HEADER' iv_bind_conversions = 'X' ).
...
lo_entity_set = lo_entity_type->
create_entity_set( 'Products' )
```

In the DEFINE\_ASSOCIATION method, you can find the generated code that defines associations, association sets, referential constraints, and navigation properties of an OData model.

5.5.2 Data Provider Class and Data Provider Extension Class

The DPC is an ABAP class that provides all methods required to handle OData requests. It's called at runtime to perform these requests; essentially,

we're talking about the runtime representation of your service implementation. For instance, a DPC executes CRUD-Q operations, among many more operations.

Again, you can find an extension class (suffix \_DPC\_EXT) and a base class (suffix \_DPC). The data provider extension class inherits from the DPC base class, as shown in Figure 5.28. We can see the methods that have been inherited by the extension class from the base class for CRUD-Q and query operations and one method that has been redefined in the extension class for the code-based implementation. The DPC extension class is registered via the technical service name. So, the extension class is executed in your OData service.

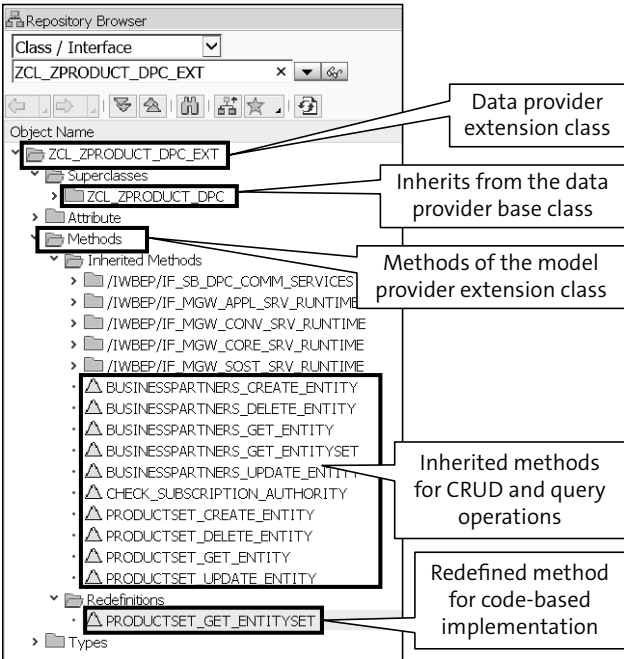


Figure 5.28 Data Provider Extension Class Interface

Note that, in the DPC, some methods are specific to an entity set, and some are not.

Entity set-specific methods

Data Provider Class Deep Dive

For each entity set, the Service Builder creates methods that are called by the framework if a create, read, update, and delete (CRUD) method is sent to this entity set. For an entity set called <ENTITYSET>, the methods created in the base class are shown in Table 5.3.

DPC Method Name	HTTP Verb	Target
<ENTITYSET>_CREATE_ENTITY	POST	Entity set
<ENTITYSET>_DELETE_ENTITY	DELETE	Entity
<ENTITYSET>_GET_ENTITY	GET	Entity
<ENTITYSET>_GET_ENTITYSET	GET	Entity set
<ENTITYSET>_UPDATE_ENTITY	UPDATE or PATCH	Entity

Table 5.3 Entity Set-Specific CRUD Method Implementation in the DPC

Additional methods available apply not only for a single entity set but for all of them (non-entity-set-specific methods). Examples of these methods include methods handling \$EXPAND statements, deep insert statements, or statements that are called when a function import is performed. Let’s take a closer look at these examples:

\$expand statements

■ **GET\_EXPANDED\_ENTITY and GET\_EXPANDED\_ENTITYSET**  
Handling of \$expand statements is offered by the SAP Gateway framework out of the box in a generic way after you’ve modeled the appropriate navigation property and implemented the handling of navigation properties. In some situations, you might instead handle \$expand requests by a specific application implementation. Examples are certain BAPIs such as BAPI\_EPM\_SO\_GET\_LIST that, along with the header data, also retrieve line items. In this case, when retrieving the sales order header data for a certain sales order, the corresponding sales order items are also read. If the entity set is also called to expand the line items alongside the sales order header, this results in unnecessary database requests.

Deep insert statement

■ **CREATE\_DEEP\_ENTITY**  
The counterpart of the \$expand statement is the *deep insert* statement, which calls the CREATE\_DEEP\_ENTITY method. A typical example is the case where a sales order can only be created alongside at least one sales order item. In contrast to the \$expand statement, there’s no generic handling of a deep insert request. The developer must implement this method.

■ **EXECUTE\_ACTION**  
The EXECUTE\_ACTION method is a non-entity-set-specific method as well. This method rather service semantic and is called if a function import into an OData service is called. Function imports allow you to execute functions that can read and/or write data. Function imports

are suitable whenever the business scenario requires data to be read or changed that can’t be modeled into an entity where you can use CRUD-Q methods.

5.5.3 Technical Considerations

OData channel development can take place either on the SAP backend system or on the SAP Gateway server, as shown in Figure 5.29. Both options are suitable for certain use cases and have their advantages. Wherever you develop, the BEP component must be installed in that system, or you must use a system based on SAP NetWeaver 7.40 or higher.

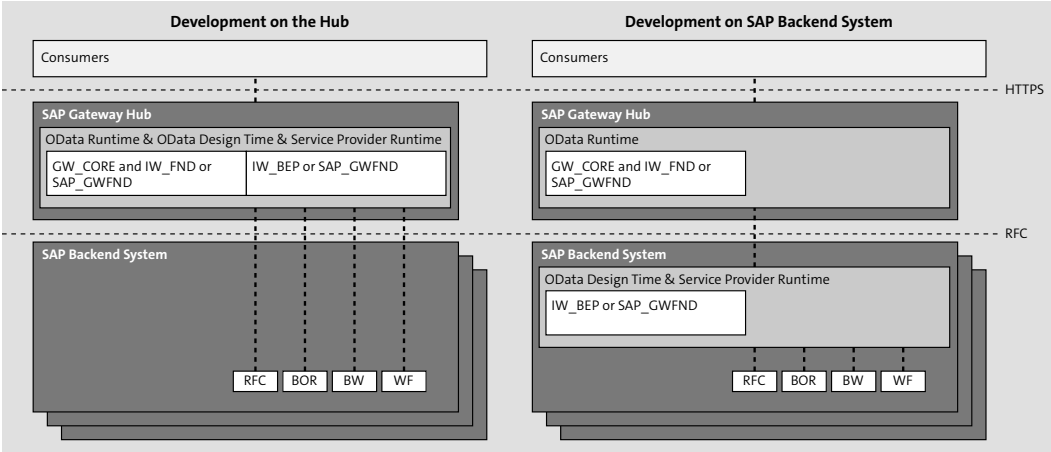


Figure 5.29 OData Channel Development on the Hub or on the SAP Backend

5.6 Summary

Building OData services with SAP Gateway is performed by following the SAP Gateway service creation process. This process is strongly supported and facilitated by the central SAP Gateway service creation tool: the Service Builder. In this chapter, we introduced you to the Service Builder and its process to establish a base of knowledge for the more technical step-by-step instructions in the next chapters, which focus in more detail on the processes of service development and service generation. In the next chapter (Chapter 6), you’ll also learn how to take advantage of the OData channel programming paradigm that we introduced in this chapter, as we dive deeper into service development.

# Contents

Foreword by Jürgen Müller .....	17
Introduction .....	19
Acknowledgments .....	23

## PART I   Getting Started

### **1   Introduction to SAP Gateway** 27

---

<b>1.1   Modern Business Applications</b> .....	28
1.1.1   User Interfaces .....	29
1.1.2   Infrastructures .....	36
<b>1.2   SAP Gateway for Modern Business Applications</b> .....	39
<b>1.3   SAP Gateway in SAP S/4HANA</b> .....	43
<b>1.4   Installation and Deployment</b> .....	45
1.4.1   Installation .....	45
1.4.2   Deployment .....	48
<b>1.5   SAP Gateway and Related Products</b> .....	52
1.5.1   SAP Business Technology Platform .....	52
1.5.2   SAP Integration Suite .....	53
1.5.3   SAP Extensibility .....	53
1.5.4   SAP Process Orchestration .....	54
1.5.5   SAP HANA .....	54
1.5.6   SAP Business Warehouse and SAP BW/4HANA .....	55
1.5.7   SAP BTP, ABAP Environment .....	55
1.5.8   SAP Fiori .....	56
1.5.9   SAP API Management .....	56
<b>1.6   Summary</b> .....	57

### **2   Introduction to OData** 59

---

<b>2.1   OData and REST</b> .....	59
2.1.1   What Is REST? .....	59
2.1.2   What Is OData? .....	62



<b>2.2</b>	<b>Structure of an OData Service</b>	67
2.2.1	Service Document	69
2.2.2	Service Metadata Document	73
<b>2.3</b>	<b>OData Operations</b>	76
2.3.1	Create	76
2.3.2	Read	76
2.3.3	Update	78
2.3.4	Delete	78
<b>2.4</b>	<b>OData Query Options</b>	78
2.4.1	Filtering and Projecting (\$filter and \$select)	80
2.4.2	Sorting (\$orderby)	83
2.4.3	Client-Side Paging (\$top, \$skip, and \$inlinecount)	84
2.4.4	Counting (\$count)	88
2.4.5	Inlining (\$expand)	88
2.4.6	Formatting (\$format)	92
<b>2.5</b>	<b>OData in SAP Solutions</b>	94
2.5.1	SAP Fiori	96
2.5.2	SAP Build	96
2.5.3	SAP Analytics Cloud	97
2.5.4	SAP Mobile Start	98
2.5.5	SAP Solution Manager	98
2.5.6	SAP S/4HANA	98
2.5.7	SAP Business Technology Platform	99
2.5.8	SAP SuccessFactors Solutions	100
<b>2.6</b>	<b>SAP Gateway OData Features</b>	101
<b>2.7</b>	<b>What's Different with OData 4.0?</b>	104
2.7.1	JavaScript Object Notation Format	104
2.7.2	Powerful Query Language	106
2.7.3	Cross-Service Navigation	107
2.7.4	Actions and Functions	107
2.7.5	Vocabularies and Annotations	108
<b>2.8</b>	<b>Summary</b>	108
<b>3</b>	<b>Architecture and Integration</b>	109
<b>3.1</b>	<b>Gateway Principles</b>	109
<b>3.2</b>	<b>SAP Gateway Architecture</b>	111
3.2.1	Consumer Tier	113

3.2.2	SAP Gateway Tier	114
3.2.3	SAP Backend Tier	116
3.2.4	Add-On Structure Evolution	116
<b>3.3</b>	<b>Integration with Other Technologies</b>	121
3.3.1	Remote Function Call	121
3.3.2	Business Object Repository	122
3.3.3	Service Provider Interface	122
3.3.4	SAP BW InfoCubes	123
3.3.5	Multidimensional Expressions	123
3.3.6	SAP BW Easy Queries and OData Queries	123
3.3.7	Generic Interaction Layer APIs	124
3.3.8	SAP Business Process Management	124
3.3.9	SAP Business Workflow	124
<b>3.4</b>	<b>ABAP Programming Model for SAP Fiori</b>	125
3.4.1	Architecture and Technology	125
3.4.2	SAPUI5 and SAP Fiori Elements	127
3.4.3	SAP Gateway and the ABAP Programming Model for SAP Fiori	129
3.4.4	SAP HANA	132
<b>3.5</b>	<b>ABAP RESTful Application Programming Model</b>	133
3.5.1	Architecture and Technology	133
3.5.2	CDS Data Modeling and Behavior Definition	139
3.5.3	Business Service Exposure	140
3.5.4	Service Consumption	140
<b>3.6</b>	<b>Summary</b>	141
<b>4</b>	<b>Deployment Options, Installation, and Configuration</b>	143
<b>4.1</b>	<b>Introduction to SAP Gateway Deployment</b>	143
4.1.1	Hub Deployment with Development in the SAP Backend System	146
4.1.2	Hub Deployment with Development on the Hub	147
4.1.3	Embedded Deployment	149
4.1.4	Mixed Deployment Options	150
4.1.5	Deployment Options for SAP Fiori and SAP S/4HANA	152
4.1.6	Comparison of Deployment Options	154
<b>4.2</b>	<b>Preparing for Installation and Configuration</b>	155

<b>4.3</b>	<b>Quick Start Guide</b>	157
4.3.1	Step 1: Deploy the SAP Gateway Add-Ons for Older SAP NetWeaver Versions	158
4.3.2	Step 2: Activate SAP Gateway	159
4.3.3	Step 3: Create an SAP System Alias	159
4.3.4	Step 4: Create an SAP Gateway Alias	161
4.3.5	Step 5: Activate the OPU Node	162
4.3.6	Step 6: Test Your Settings	163
<b>4.4</b>	<b>Installation and Configuration in Detail</b>	164
4.4.1	Installing the SAP Gateway Add-Ons	165
4.4.2	Basic Configuration Settings	165
4.4.3	OData Channel Configuration	167
4.4.4	Business Enablement Provisioning Configuration	173
4.4.5	Smoke Testing	173
<b>4.5</b>	<b>Summary</b>	175

**PART II Service Creation**

<b>5</b>	<b>Introduction to OData Service Creation</b>	179
<b>5.1</b>	<b>Methods for Creating an OData Service</b>	180
<b>5.2</b>	<b>Service Creation Process Overview</b>	182
<b>5.3</b>	<b>SAP Gateway Toolset</b>	186
5.3.1	Service Builder	186
5.3.2	Supporting Tools during the Service Creation Process	189
5.3.3	ABAP Development Tools and CDS Views	193
<b>5.4</b>	<b>Steps in the Service Creation Process</b>	196
5.4.1	Data Model Definition in the Service Builder	196
5.4.2	Service Registration in the SAP Backend System	198
5.4.3	Service Implementation	200
5.4.4	Service Maintenance	205
5.4.5	Service Generation via Redefinition	206
5.4.6	Service Generation via Referenced Data Sources	212
5.4.7	Service Generation via OData.publish:true	213
5.4.8	Service Generation for Analytical Queries	214
<b>5.5</b>	<b>OData Channel Development Paradigm</b>	216
5.5.1	Model Provider Class	216

5.5.2	Data Provider Class and Data Provider Extension Class	218
5.5.3	Technical Considerations	221
<b>5.6</b>	<b>Summary</b>	221
<b>6</b>	<b>Service Development</b>	223
<b>6.1</b>	<b>Data Model Definition</b>	224
6.1.1	Creating a Project	224
6.1.2	Creating a Data Model	228
<b>6.2</b>	<b>Service Registration</b>	250
<b>6.3</b>	<b>Service Stub Generation</b>	255
<b>6.4</b>	<b>Service Maintenance</b>	257
<b>6.5</b>	<b>Incremental Service Implementation and Model Enhancement</b>	261
6.5.1	Feed (GET_ENTITYSET)	263
6.5.2	Single Read (GET_ENTITY)	267
6.5.3	Query Options	270
6.5.4	Navigation Properties	279
6.5.5	Create, Update, and Delete Methods	286
6.5.6	Function Imports	294
6.5.7	Media Resources	301
6.5.8	Expand/Self-Expand	312
6.5.9	Deep Insert	320
6.5.10	Batch	324
6.5.11	Adding UI Annotations	328
<b>6.6</b>	<b>Summary</b>	335
<b>7</b>	<b>Service Generation</b>	337
<b>7.1</b>	<b>Generation via RFC/BOR Interface</b>	340
7.1.1	Data Model Definition	342
7.1.2	Service Registration: Stub Creation	347
7.1.3	Service Maintenance	348
7.1.4	Service Implementation: SalesOrderHeaderSet	350
7.1.5	Service Implementation: SalesOrderLineItemSet	362
<b>7.2</b>	<b>Generation via Search Help</b>	372

<b>7.3</b>	<b>Generation via Redefinition .....</b>	<b>374</b>
7.3.1	Redefinition Options .....	376
7.3.2	Redefine Service Provider Interface Service .....	378
7.3.3	Activate Service .....	381
<b>7.4</b>	<b>Generation via Analytical Queries .....</b>	<b>382</b>
7.4.1	Analytical Queries with Query Property by OData .....	382
7.4.2	CDS View-Based Analytical Queries .....	389
<b>7.5</b>	<b>Summary .....</b>	<b>392</b>
<b>8</b>	<b>ABAP Programming Model for SAP Fiori .....</b>	<b>395</b>
<hr/>		
<b>8.1</b>	<b>Development of CDS Views .....</b>	<b>397</b>
<b>8.2</b>	<b>Modeled Data Sources .....</b>	<b>403</b>
<b>8.3</b>	<b>Referenced Data Sources .....</b>	<b>408</b>
<b>8.4</b>	<b>Adding Annotations to an OData Service .....</b>	<b>413</b>
<b>8.5</b>	<b>ABAP Programming Model for SAP Fiori with Classic APIs .....</b>	<b>416</b>
<b>8.6</b>	<b>ABAP Programming Model for SAP Fiori with BOPF .....</b>	<b>426</b>
8.6.1	Defining the Business Object .....	426
8.6.2	Defining the Consumption View .....	431
8.6.3	Determinations, Validations, and Actions .....	434
8.6.4	Draft Support .....	437
<b>8.7</b>	<b>Summary .....</b>	<b>438</b>
<b>9</b>	<b>ABAP RESTful Application Programming Model .....</b>	<b>441</b>
<hr/>		
<b>9.1</b>	<b>Data Modeling .....</b>	<b>443</b>
9.1.1	Database Tables .....	444
9.1.2	Data Definition .....	447
9.1.3	Customer Data .....	451
<b>9.2</b>	<b>Business Service Exposure .....</b>	<b>455</b>
9.2.1	CDS Projection Views .....	455
9.2.2	CDS Metadata Extensions .....	458
9.2.3	Service Definition .....	463
9.2.4	Service Binding .....	464

<b>9.3</b>	<b>Service Consumption .....</b>	<b>467</b>
<b>9.4</b>	<b>Transactional Behavior .....</b>	<b>471</b>
9.4.1	Managed .....	472
9.4.2	Managed with Unmanaged Save .....	488
9.4.3	Unmanaged .....	490
<b>9.5</b>	<b>Generating an Application Using ABAP Development Tools .....</b>	<b>491</b>
<b>9.6</b>	<b>Entity Manipulation Language .....</b>	<b>495</b>
9.6.1	EML Read .....	495
9.6.2	EML Modify .....	498
9.6.3	EML Commit Entities .....	499
<b>9.7</b>	<b>Business Object Characteristics .....</b>	<b>499</b>
9.7.1	Draft .....	500
9.7.2	Numbering .....	501
9.7.3	Permissions .....	502
<b>9.8</b>	<b>Summary .....</b>	<b>504</b>

PART III Application Development

<b>10</b>	<b>SAPUI5 Application Development .....</b>	<b>507</b>
<hr/>		
<b>10.1</b>	<b>Building Blocks of Web Application Development .....</b>	<b>508</b>
<b>10.2</b>	<b>Introduction to SAP Fiori and SAPUI5 .....</b>	<b>509</b>
10.2.1	SAP Fiori .....	509
10.2.2	SAPUI5 .....	513
<b>10.3</b>	<b>Creating an SAPUI5 Application .....</b>	<b>515</b>
<b>10.4</b>	<b>Summary .....</b>	<b>517</b>
<b>11</b>	<b>SAP Business Application Studio .....</b>	<b>519</b>
<hr/>		
<b>11.1</b>	<b>Setting Up SAP Business Application Studio .....</b>	<b>520</b>
<b>11.2</b>	<b>Connecting to SAP Gateway .....</b>	<b>524</b>
11.2.1	Setting Up the Connection .....	525
11.2.2	Creating an SAP Business Application Studio Dev Space .....	529
<b>11.3</b>	<b>OData Sample Services .....</b>	<b>532</b>

<b>11.4</b>	<b>Developing SAP Fiori Applications</b>	534
11.4.1	Creating a Basic SAP Fiori App	534
11.4.2	Changing the UI and Adding an OData Entity Set	543
<b>11.5</b>	<b>Summary</b>	549
 <b>12 Extensibility</b>		551
<b>12.1</b>	<b>Redefining and Extending SAP Gateway OData Services</b>	551
12.1.1	Redefinition	552
12.1.2	Field Extensibility	556
12.1.3	Node Extensibility	557
12.1.4	Step-by-Step Guide	558
<b>12.2</b>	<b>Extending OData Services in SAP S/4HANA</b>	581
12.2.1	Extensibility Concepts for SAP S/4HANA	581
12.2.2	Key User Extensibility	582
12.2.3	In-App Developer/On-Stack Extensibility	588
12.2.4	Side-by-Side Extensibility	592
<b>12.3</b>	<b>Extending SAP Fiori Applications Using SAPUI5 Flexibility</b>	601
12.3.1	Key User Adaptation	601
12.3.2	Creating an Adaptation Project in SAP Business Application Studio	603
<b>12.4</b>	<b>Summary</b>	607
 <b>13 Mobile Application Development</b>		609
<b>13.1</b>	<b>Overview</b>	610
<b>13.2</b>	<b>Mobile Development Kits</b>	611
13.2.1	Accessing SAP Mobile Services	612
13.2.2	Creating a Mobile App	615
13.2.3	Creating a Project in SAP Business Application Studio	617
13.2.4	Deploying the Application	624
13.2.5	Using the Application	626
<b>13.3</b>	<b>Native Application Development</b>	628
13.3.1	SAP BTP SDK for iOS	629
13.3.2	SAP BTP SDK for Android	636
<b>13.4</b>	<b>Summary</b>	646

<b>14 Social Media Application Development</b>		647
<b>14.1</b>	<b>PHP</b>	648
<b>14.2</b>	<b>Facebook</b>	652
<b>14.3</b>	<b>X</b>	659
<b>14.4</b>	<b>Sina Weibo ( 新浪微博 )</b>	663
<b>14.5</b>	<b>Summary</b>	674
 <b>15 Enterprise Application Development</b>		675
<b>15.1</b>	<b>Microsoft Excel</b>	676
15.1.1	Microsoft Power Pivot	676
15.1.2	Get OData Service	681
15.1.3	\$format=xlsx	684
<b>15.2</b>	<b>Microsoft SharePoint/Microsoft 365</b>	685
<b>15.3</b>	<b>Microsoft Visual C# Windows Desktop</b>	692
<b>15.4</b>	<b>Microsoft ASP.NET</b>	697
<b>15.5</b>	<b>Summary</b>	697
 <b>PART IV Administration</b>		
<b>16 Lifecycle Management: Testing, Service Deployment, and Operations</b>		701
<b>16.1</b>	<b>Testing</b>	702
16.1.1	Testing SAP Gateway Services	703
16.1.2	Testing a Client Application or Extension Application	706
16.1.3	Best Practices for Testing in SAP Gateway	709
<b>16.2</b>	<b>Service Deployment</b>	710
16.2.1	Transport of Repository Objects between SAP Backend Systems	711
16.2.2	Transport of Repository Objects and Customizing Entries between SAP Gateway Server Systems	713
16.2.3	Versioning	716
16.2.4	Activating and Maintaining Services	717
16.2.5	Service Deployment in OData 4.0	718



<b>16.3 Operations</b>	720
16.3.1 Periodic Cleanup Tasks	720
16.3.2 Monitoring Overview	721
<b>16.4 DevOps and SAP Gateway Development</b>	728
<b>16.5 Summary</b>	730

**17 Security** 731

<b>17.1 Network and Communication Security</b>	731
17.1.1 Transport Protection	732
17.1.2 Input Validation	735
<b>17.2 User Management and Authorizations</b>	739
<b>17.3 Single Sign-On and Authentication Options</b>	741
17.3.1 Basic Authentication	743
17.3.2 SAP Logon Tickets with SAP Enterprise Portal	744
17.3.3 X.509 Client Certificates	745
17.3.4 SAML 2.0 Browser SSO Protocol	746
17.3.5 OAuth	748
17.3.6 Kerberos: Integrated Windows Authentication	749
<b>17.4 Recommended Authentication Options</b>	750
17.4.1 SAP Fiori Applications	751
17.4.2 Desktop Application	752
17.4.3 Mobile Application (Direct Access)	753
17.4.4 Cloud-Based Scenario with SAP BTP	755
17.4.5 SAP Mobile Services	756
17.4.6 Web Server	757
17.4.7 Business-to-Consumer Scenario	758
17.4.8 Read Access Logging	763
<b>17.5 Summary</b>	766

**Appendices** 769

<b>A Advanced Topics</b>	769
<b>B The Authors</b>	793
<b>Index</b>	797

# Index

.NET Framework ..... 692, 697  
@OData.publish:true ..... 131  
\$count ..... 88  
\$expand ..... 88, 262, 312, 319  
    *statement* ..... 220  
\$filter ..... 80  
\$format ..... 92  
\$format=xlsx ..... 684  
\$inlinecount ..... 85  
\$orderby ..... 83  
\$select ..... 80  
\$skip ..... 85  
\$top ..... 85  
\$value ..... 302

## A

ABAP Cloud ..... 581  
ABAP data type ..... 446  
ABAP Development Tools (ADT) ..... 115,  
    116, 134, 193, 195, 213, 398, 443  
    *add annotations* ..... 413  
    *generate application* ..... 491  
    *generator tool* ..... 492  
    *install* ..... 398  
ABAP Development Workbench ..... 227  
ABAP Flight Reference Scenario ..... 137  
ABAP for Cloud Development ..... 441,  
    588, 589  
ABAP for SAP HANA ..... 132  
ABAP in Eclipse ..... 116, 398  
ABAP Managed Database  
    Procedures (AMDPs) ..... 132  
ABAP programming model for  
    SAP Fiori ..... 137, 213, 395  
    *annotations* ..... 413  
    *architecture* ..... 125  
    *best practices* ..... 131  
    *BOPF* ..... 396, 426  
    *classic APIs* ..... 396, 416  
    *layers* ..... 126  
    *SAP Gateway* ..... 129  
    *service generation* ..... 339  
ABAP RESTful application  
    programming model ..... 133, 441  
    *advantages* ..... 136  
    *analytical queries* ..... 389  
    *architecture* ..... 133

ABAP RESTful application programming  
    model (Cont.)  
        *behavior extension* ..... 590  
        *business object characteristics* ..... 499  
        *business service exposure* ..... 140, 455  
        *customer data* ..... 451  
        *database tables* ..... 444  
        *data definition* ..... 447  
        *data modeling* ..... 443  
        *developer extensibility* ..... 588  
        *development flow* ..... 138  
        *EML* ..... 495  
        *events* ..... 779  
        *extend business object* ..... 589  
        *field mapping* ..... 477  
        *generate application* ..... 491  
        *implementation types* ..... 136, 472  
        *managed implementation* ..... 472  
        *managed with unmanaged save* ... 488  
        *mass-enabled handlers* ..... 483  
        *OData client proxy* ..... 211  
        *permissions* ..... 502  
        *protocol-agnostic development* .... 466  
        *service binding* ..... 464  
        *service consumption* ..... 140, 467  
        *service definition* ..... 463  
        *service generation* ..... 340  
        *tools* ..... 134  
        *transactional behavior* ..... 471  
        *unmanaged implementation* ..... 490  
Absolute path ..... 309  
Action ..... 434, 486  
    *projection* ..... 487  
Adaptation project ..... 603  
    *configuration* ..... 604  
Agility ..... 33  
Analytical app ..... 511  
Analytical queries ..... 214, 339, 382  
    *activate service* ..... 384, 387  
    *by OData* ..... 382  
    *CDS views* ..... 389  
    *create* ..... 383  
    *regenerate service* ..... 389  
Android emulator ..... 643  
Android Studio ..... 636  
Android Virtual Device (AVD) ..... 637  
Annotation ..... 226, 396, 413  
    *add* ..... 413

Annotation (Cont.)	Authentication .....	741, 743
<i>object page</i> .....	<i>anonymous access</i> .....	759
SAP .....	B2C .....	758
<i>semantics</i> .....	<i>basic</i> .....	743
UI .....	cloud .....	755
Annotation.xml .....	<i>desktop applications</i> .....	752
Annotation model for	Kerberos .....	749
referenced service .....	<i>mobile applications</i> .....	753
Append structure .....	OAuth .....	748
create .....	<i>recommendations</i> .....	750
Application	SAML .....	761
design .....	SAML 2.0 browser protocol .....	746
log .....	SAP Fiori apps .....	751
Application development	SAP logon tickets and	
enterprise .....	SAP Enterprise Portal .....	744
mobile .....	SAP Mobile Services .....	756
native .....	user self-service .....	759
SAP Business Application Studio ....	web server .....	757
SAP Fiori .....	X.509 client certificate .....	745
SAPUI5 .....	Authorization control .....	503
social media .....	Authorization master .....	475
Application Log Viewer .....	Authorizations .....	739
Application programming	Availability .....	35
interface (API) .....		
ABAP programming model for	<b>B</b>	
SAP Fiori .....	Backwards compatibility .....	66
economy .....	Base class .....	217
endpoint .....	Batch .....	324
management .....	Batch handling .....	101
service consumption models .....	Beautification .....	198, 386
Application requirements .....	Behavior definition .....	140, 443, 473
infrastructure .....	create .....	473
user interface .....	projection .....	479
App Preview .....	Behavior Definition	
Architecture .....	Language (BDL) .....	442, 473
ABAP programming model for	Behavior extension .....	590
SAP Fiori .....	Behavior implementation ...	140, 443, 478
ABAP RESTful application	class .....	478
programming model .....	BOPF object .....	244
backend system tier .....	define .....	426
consumer tier .....	define consumption view .....	431
SAP backend tier .....	draft-enabled .....	438
SAP Gateway Server tier .....	Breakpoint .....	785, 786
SAP Gateway tier .....	Bring your own device (BYOD) .....	31
Association .....	Browser-based access .....	94
68, 75, 411, 412, 429,	Browser protocol .....	147
453, 573	Business Application Programming	
define .....	Interface (BAPI) .....	51, 116, 337
redirect .....	Business Enablement	
set .....	Provisioning (BEP) .....	168, 173, 187
Atom .....	Business exception .....	266
AtomPub .....		

Business logic extensibility .....	Composite interface view .....	397
Business object .....	Composition .....	442
BOPF .....	<i>hierarchy</i> .....	448, 474
characteristics .....	Computing Center Management	
data definition .....	System (CCMS) .....	721, 722
view .....	Conceptual schema definition	
Business Object Layer (BOL) .....	language (CSDL) .....	73
Business Object Processing	Consumer .....	113, 170
Framework (BOPF) .....	<i>connection settings</i> .....	170
Business Object Repository (BOR) ...	Consumption layer .....	456
121, 122, 228, 244, 337	Consumption view .....	397, 418
import data model .....	BOPF objects .....	431
Business process expert (BPX) .....	contacts .....	419
Business service expoure .....	publish as OData service .....	420
	Content ID referencing .....	325
<b>C</b>	Continuous integration and	
C# .....	continuous delivery (CI/CD) .....	729
CamelCase .....	Conversion exit .....	269, 356
Cascading Style Sheets Level 3	Conversion handling .....	103
(CSS3) .....	Core components .....	114
Catalog service .....	Core data services (CDS) .....	121, 125,
CDS view .....	134, 401	
ABAP .....	behavior .....	140
activate .....	customer data .....	451
add association .....	data definition .....	447
analytical queries .....	data model extensions .....	591
architecture .....	data modeling .....	139, 443
development .....	entity .....	139, 195
entity .....	graphical modeler .....	529
extend .....	mapping .....	204
modeled data source .....	metadata extensions .....	458
naming .....	naming convention .....	400
projection .....	projection views .....	455
referenced data source .....	table functions .....	132
reuse .....	Counting .....	88
SAP HANA .....	Creatable .....	231
SAP NetWeaver .....	Create, read, update, and delete	
Central User Administration (CUA) ...	(CRUD) .....	62, 76, 101
Change and Transport System	Create, read, update, delete,	
(CTS) .....	query (CRUD-Q) .....	252, 264, 533
Check-before-save operation .....	Create, update, delete (CUD)	
Class Builder .....	method .....	286
Cleanup tasks .....	create .....	286
Client server architecture .....	delete .....	293
Client-side paging .....	media resources .....	101
Cloud .....	update .....	290
hybrid .....	Create-by-association (CBA) .....	475
private .....	Create operation .....	76, 357, 367
public .....	Cross-service navigation .....	107
Cloud connector .....	Cross-site request forgery	
Cloud Foundry .....	(XSRF) .....	735, 736
	Cross-site scripting (XSS) .....	735

Current settings .....	716	Deployment .....	48, 143, 144
Custom business object .....	584	<i>choosing</i> .....	51
Custom exception class .....	421	<i>cloud</i> .....	51, 145
Custom field .....	565	<i>comparison</i> .....	154
<i>create</i> .....	585	<i>costs</i> .....	154
<i>enable and add</i> .....	586	<i>embedded</i> .....	49, 145, 149
<i>service implementation</i> .....	566	<i>hardware requirements</i> .....	155
<i>test</i> .....	586	<i>hub</i> .....	50, 145
Custom Fields and Logic app .....	583, 585, 586	<i>installation and configuration</i> .....	154
		<i>introduction</i> .....	143
		<i>maintenance</i> .....	154
		<i>mixed deployment</i> .....	150
		<i>options</i> .....	711
		<i>performance</i> .....	154
		<i>SAP Fiori</i> .....	152
		<i>SAP S/4HANA</i> .....	152
		<i>software requirements</i> .....	155
		Desktop .....	707
		Destination .....	526, 622
		<i>properties</i> .....	528
		Determination .....	434, 481
		Developer extensibility .....	581, 588
		DevOps .....	728
		Dev space .....	520
		<i>create</i> .....	529, 617
		Digital transformation .....	28, 32
		Dispatcher method .....	557, 577
		Draft .....	415, 437, 444, 500
		<i>determine action</i> .....	501
		<i>tables</i> .....	500
		Durable lock .....	500
		Dynamic feature control .....	503
		<b>E</b>	
		Early numbering .....	502
		Eavesdropping .....	731
		Edm:Action .....	107
		Edm:Function .....	107
		Embedded deployment .....	149
		<i>advantages</i> .....	149
		<i>disadvantages</i> .....	150
		<i>FES</i> .....	153, 154
		<i>release consideration</i> .....	150
		<i>use cases</i> .....	149
		Enterprise application	
		development .....	675
		<i>Microsoft 365</i> .....	685
		<i>Microsoft ASP.NET</i> .....	697
		<i>Microsoft SharePoint</i> .....	685
		<i>Microsoft Visual C# Windows</i>	
		<i>Desktop</i> .....	692

Enterprise Procurement		Filterable .....	231
Model (EPM) .....	340, 533, 536, 577	Filters .....	80, 102
Entity .....	67	Flight example (read-only) .....	533
<i>data model</i> .....	73	Floorplan Manager (FPM) .....	122
Entity data model XML (EDMX) .....	228	Formatting .....	92
<i>import data model</i> .....	236	Framework template .....	168
Entity Manipulation		Function import .....	294
Language (EML) .....	443, 495	<i>create</i> .....	295
<i>commit entities</i> .....	499	Function module .....	86
<i>modify</i> .....	487, 498		
<i>read</i> .....	495		
Entity set .....	67, 70, 77, 95, 346, 546	<b>G</b>	
<i>attributes</i> .....	233	Gateway principle .....	109
<i>create</i> .....	228, 229	<i>division of work</i> .....	111
<i>custom</i> .....	570, 574	<i>ease of consumption</i> .....	110
Entity type .....	67, 75	<i>openness</i> .....	110
<i>attributes</i> .....	229	<i>timelessness</i> .....	110
<i>create</i> .....	228	<i>user focus</i> .....	111
Error context .....	724	Generic channel .....	115
Error log .....	190, 722, 723, 781	Generic Interaction Layer (GenIL) .....	121, 124, 209, 210, 338
<i>backend</i> .....	784	GET .....	60
<i>level</i> .....	785	GET_ENTITY .....	267, 283
ETag .....	475, 596	GET_ENTITYSET .....	263
Event .....	115, 778	GETDETAIL method .....	78
Event broker .....	780	Git repository .....	520
Event consumer .....	780	GW_CORE .....	116–119, 143, 144, 159
Event-driven architecture (EDA) .....	776	GWSAMPLE_BASIC service .....	68
Event source .....	779		
EXECUTE_ACTION method .....	220		
Expand .....	102	<b>H</b>	
Extended Computer Aided		Hello World .....	515
Test Tool (eCATT) .....	706	HTML5 .....	32, 513
Extensibility .....	53, 551	<i>browsers</i> .....	508
<i>options</i> .....	581	HTML5/SAPUI5 .....	31, 50, 96, 181, 212, 507
<i>SAP S/4HANA</i> .....	581	HTTP .....	358
<i>SAPUI5 flexibility</i> .....	601	<i>method</i> .....	78, 296
Extension class .....	217, 251	<i>status code</i> .....	101
Extension include .....	557, 558	HTTP(S) communication .....	733
External numbering .....	501	Hub deployment .....	145
		<i>advantages</i> .....	146, 148
		<i>development in SAP Business</i>	
		<i>Suite</i> .....	146
		<i>development on the hub</i> .....	50, 147
		<i>disadvantages</i> .....	147
		<i>FES</i> .....	154
		<i>on the backend system</i> .....	50
		<i>use case</i> .....	146, 148
		Hybrid container app .....	610
		Hypermedia link .....	61
		Hypertext Preprocessor (PHP) .....	648

		<b>F</b>	
Facebook development .....	647, 652		
<i>create application</i> .....	652		
<i>create developer account</i> .....	652		
<i>namespace and app domains</i> .....	653		
Facebook PHP SDK .....	655		
Fact sheet app .....	511		
Feature control .....	502		
Feeds .....	101, 263		
Field extensibility .....	556, 583		
Field mapping .....	477		



I

Idempotency ..... 102

Identity provider ..... 613

In-app extensibility ..... 581

Incremental exclusion ..... 728

InfoCube ..... 121, 123

Infrastructure as a service (IaaS) ..... 38

Inlining ..... 88

Input mapping ..... 354

Installation and configuration ..... 45, 143

*activation* ..... 159

*add-ons* ..... 158, 165

*Business Enablement*

*Provisioning* ..... 173

*node OPU* ..... 162

*OData channel* ..... 167

*Quick Start Guide* ..... 143, 157

*SAP Gateway alias* ..... 161

*SAP system alias* ..... 159

*settings* ..... 165

*smoke testing* ..... 173

*steps* ..... 164

*test settings* ..... 163

*trust relationship* ..... 170

Integrated development

    environment (IDE) ..... 113, 114

Interaction phase ..... 471

Interface parameter ..... 245

Interface view ..... 397

Internal numbering ..... 501

Internet Communication

    Framework (ICF) ..... 166, 348, 743

Internet Communications

    Manager (ICM) ..... 743

IW\_BEP ..... 118, 119, 143, 144, 159, 351

IW\_CBS ..... 118

IW\_CNT ..... 118

IW\_FND ..... 116–119, 143, 144, 159

IW\_FNDGC ..... 118

IW\_GIL ..... 120, 211

*SAP S/4HANA* ..... 208

IW\_HDB ..... 116, 118–120, 144

IW\_PGW ..... 119

IW\_SCS ..... 118

IW\_SPI ..... 120

*SAP S/4HANA* ..... 208, 211, 377

J

Java ..... 31

JavaScript ..... 507

JavaScript Object Notation (JSON) ..... 64, 101, 104, 114

Join ..... 406

jQuery ..... 513

K

Kerberos ..... 147, 749

Key user adaptation ..... 601

Key user extensibility ..... 581, 582, 602

L

Late numbering ..... 502

List report app ..... 331

List reporting template ..... 415, 423

Lock master ..... 474

M

Managed implementation

    type ..... 136, 472

Managed scenario ..... 426

manifest.json file ..... 467

Mapping ..... 351

Map to data source ..... 180, 357

Media resource ..... 301

Merge/patch ..... 102

Metadata component ..... 115

Metadata extension (MDE) ..... 418, 432, 458

Microsoft 365 ..... 685

*minimum requirements* ..... 685

*SharePoint* ..... 691

Microsoft ASP.NET ..... 697

Microsoft Excel ..... 676

*\$format=xlsx* ..... 684

*get OData service* ..... 681

Microsoft Excel development ..... 675

*Power Pivot* ..... 676

Microsoft SharePoint

    development ..... 675, 685

*specify source* ..... 688

*steps* ..... 685

Microsoft Visual C# development ..... 675

*development* ..... 692

Microsoft Visual C# Windows

    Desktop ..... 692

*minimum requirements* ..... 692

MIME type ..... 302, 304

Mobile application development ..... 609, 615, 707

*deploy* ..... 624

Mobile development kit (MDK) ..... 611

*app* ..... 620

*basic information* ..... 622

*create* ..... 615

*create project* ..... 617

*deploy* ..... 624

*using the app* ..... 626

Modeled data sources (MDS) ..... 396, 403

Model provider class (MPC) ..... 200, 216, 218, 224, 251, 345, 403

*base class* ..... 253

*extension class* ..... 302

Model-view-controller (MVC) ..... 513

Modifying request ..... 107

Monitoring ..... 115, 721

Multichannel support ..... 55

Multidimensional

    expressions (MDX) ..... 121, 123

Multiorigin/multidestination ..... 103

My Inbox app ..... 119

MySQL ..... 648

N

Native application development ..... 628

Navigation property ..... 279, 314, 572

*define* ..... 283

Node extensibility ..... 557, 592

Nondisruptiveness ..... 35

Nonmodifying request ..... 107

Non-standard operation ..... 486

Northwind service (read only) ..... 534

Notification

*configuring* ..... 769

*events* ..... 778

*pull* ..... 770, 771

*push* ..... 770, 771

Notification center ..... 773

Notification Channel hub ..... 773

Notification provider ..... 773

*development* ..... 775

*workflow items* ..... 774

Nullable ..... 231

Numbering ..... 501

O

OAuth ..... 662, 748

    2.0 ..... 748

Object page ..... 332

OData ..... 27, 39, 42, 52, 62

*access* ..... 95

OData (Cont.)

*building blocks* ..... 66

*client library* ..... 66

*client proxy* ..... 207, 211, 595, 598

*consumer* ..... 64

*custom entity set* ..... 574

*custom field service*

*implementation* ..... 566

*data model* ..... 66

*design principles* ..... 65

*do and don't* ..... 108

*exposure* ..... 431

*history* ..... 63

*introduction* ..... 59

*library* ..... 114

*producer* ..... 64

*protocol* ..... 66, 130

*query options* ..... 78

*REST-based protocol* ..... 59

*SAP Analytics Cloud* ..... 97

*SAP BTP* ..... 99

*SAP Build* ..... 96

*SAP Fiori* ..... 96

*SAP Mobile Start* ..... 98

*SAP S/4HANA* ..... 43, 98

*SAP Solution Manager* ..... 98

*SAP solutions* ..... 94

*SAP SuccessFactors* ..... 100

*server* ..... 542

*versus OData 4.0* ..... 104

OData.publish:true ..... 213, 339, 432

OData 4.0 ..... 104

*actions and functions* ..... 107

*authorization checks* ..... 740

*cross-service navigation* ..... 107

*JSON format* ..... 104

*nonmodifying and modifying*

*request* ..... 107

*query language* ..... 106

*roadmap* ..... 104

*service binding* ..... 465

*service deployment* ..... 718

*versus OData 2.0* ..... 104

*vocabularies and annotations* ..... 108

OData channel

*activate SAP Gateway* ..... 171

*activate services* ..... 171

*authorization configuration* ..... 168

*configuration* ..... 167, 168

*SAP system alias* ..... 170

*settings for service development* ..... 171

*template* ..... 168

OData channel development	OData Services Consumption and
paradigm ..... 216	Integration (OSCI) ..... 207, 212
<i>data provider class</i> ..... 218	OData software development
<i>model provider class</i> ..... 216	kit (SDK) for PHP ..... 648
<i>technical considerations</i> ..... 221	Olingo OData Client for JavaScript ..... 130
OData client library ..... 114	Online analytical processing
OData Feed ..... 681	(OLAP) ..... 123
<i>steps</i> ..... 681	Online Database Connectivity
<i>tool</i> ..... 682	(ODBC) ..... 63
OData operation ..... 76	On-stack extensibility ..... 581, 588
<i>client-side paging</i> ..... 84	OpenSearch ..... 95, 193
<i>counting</i> ..... 88	<i>description</i> ..... 101
<i>create</i> ..... 76	Open standards ..... 37, 39
<i>delete</i> ..... 78	OpenUI5 ..... 507
<i>filtering and projecting</i> ..... 80	Operations ..... 720
<i>formatting</i> ..... 92	<i>application log</i> ..... 721
<i>inlining</i> ..... 88	<i>daily jobs</i> ..... 721
<i>query</i> ..... 77	<i>periodic tasks</i> ..... 720
<i>read</i> ..... 76	<i>troubleshooting tips</i> ..... 728
<i>single read</i> ..... 77	Optimistic locking ..... 437
<i>sorting</i> ..... 83	Order by ..... 103
<i>update</i> ..... 78	
OData programming model ..... 216	<b>P</b>
OData protocol ..... 113	Package ..... 252
<i>GW_CORE</i> ..... 117	Paging ..... 102
OData query ..... 123, 214	Pair testing ..... 728
<i>annotations</i> ..... 215	PATCH ..... 60, 78
OData sample service ..... 532	Payload Trace tool ..... 191, 792
<i>functionalities</i> ..... 533	Performance analysis ..... 788
<i>read/write</i> ..... 534	Performance trace ..... 722
<i>read-only</i> ..... 534	Performance Trace tool ..... 726, 727,
OData service ..... 66, 124, 538	789, 790
<i>ABAP programming model for</i>	Perl ..... 648
<i>SAP Fiori</i> ..... 129	Permissions ..... 502
<i>annotations</i> ..... 413	Persistent table ..... 474
<i>CDS via RDS</i> ..... 130	PHP development ..... 648
<i>CDS views</i> ..... 181	<i>download links</i> ..... 649
<i>code-based implementation</i> ..... 130	<i>generate proxy class</i> ..... 649
<i>create</i> ..... 179, 180, 182	Platform as a service (PaaS) ..... 38
<i>custom entity set</i> ..... 570	POST ..... 60
<i>custom fields</i> ..... 565	Power Pivot ..... 676
<i>extensibility</i> ..... 551, 558	<i>minimum requirements</i> ..... 676
<i>external</i> ..... 211	<i>plug-in</i> ..... 679
<i>options</i> ..... 553	<i>steps</i> ..... 678
<i>redefine</i> ..... 551, 561	Primary key ..... 496
<i>register and publish service</i> ..... 562	Principal entity ..... 346
<i>runtime object</i> ..... 562	Product availability matrix (PAM) ..... 156
<i>SAP S/4HANA</i> ..... 181	ProductCollection ..... 261, 267
<i>structure</i> ..... 67	Product Master app ..... 586
<i>test redefined service</i> ..... 564	Project Astoria ..... 64
<i>third-party</i> ..... 207	Project explorer ..... 623
<i>ZGWSAMPLE_SRV</i> ..... 69	

Projection ..... 80, 397	Remote functional call (RFC) ..... 37, 51,
Projection behavior definition ..... 479	116, 121, 148, 170, 228
Properties	<i>function module</i> ..... 255
<i>attribute flags</i> ..... 230	<i>import data model</i> ..... 244
<i>navigation</i> ..... 68, 73, 75	<i>wrapper function module</i> ..... 365
Property name ..... 269	Repeatable requests ..... 102
Provider contract ..... 457	Repository object ..... 341, 710, 713
PUT ..... 60, 78	<i>transport</i> ..... 711, 713
	Representational state
<b>Q</b>	transfer (REST) ..... 27, 39, 52, 59, 113
Query Designer ..... 123, 214, 383	<i>architecture</i> ..... 61
Query language ..... 106	<i>command</i> ..... 60
<i>analytical query options</i> ..... 106	<i>link</i> ..... 61
<i>filter expanded entities</i> ..... 106	<i>multiple representations of</i>
Query operation ..... 77, 350, 362	<i>a resource</i> ..... 62
Query option ..... 270	<i>principles</i> ..... 66
<i>\$filter</i> ..... 271	<i>stateless communication</i> ..... 62
<i>\$inlinecount</i> ..... 275	<i>uniform interface</i> ..... 62
<i>\$orderby</i> ..... 278	<i>URI</i> ..... 61
<i>\$select</i> ..... 271	Return on investment ..... 35
<i>\$skip</i> ..... 275	Return type ..... 295
<i>\$top</i> ..... 275	RFC/BOR generation ..... 337
Quick Assist ..... 478, 482	<i>create</i> ..... 357, 367
Quick Sizer ..... 175	<i>data model definition</i> ..... 342
Quick Start Guide installation ..... 157	<i>delete</i> ..... 361, 370
<i>steps</i> ..... 158	<i>interface</i> ..... 340, 362
	<i>process flow</i> ..... 342
<b>R</b>	<i>query</i> ..... 350
Read Access Logging (RAL) ..... 763	<i>service implementation</i> ..... 350
<i>\$batch</i> ..... 764	<i>service maintenance</i> ..... 348
<i>input logging</i> ..... 766	<i>single read</i> ..... 355, 364
Read media resources ..... 101	<i>stub creation</i> ..... 347
Read operation ..... 76	<i>update</i> ..... 360, 369
Redefinition ..... 180, 338, 374, 551,	Role ..... 169
552, 561	<i>administrator</i> ..... 169
<i>beautification</i> ..... 386	<i>developer</i> ..... 169
<i>GenIL</i> ..... 375	<i>user</i> ..... 169, 170
<i>ODP</i> ..... 375	ROOT ..... 444
<i>options</i> ..... 376	Runtime artifacts ..... 253
<i>SAP BW</i> ..... 375	Runtime components ..... 114
<i>Service Provider Interface</i> ..... 378	
<i>SPI</i> ..... 375	<b>S</b>
<i>three steps</i> ..... 376	SAML 2.0 Browser Protocol ..... 746
<i>URLs</i> ..... 553	SAP_GWFND ..... 118, 119, 144, 145,
Reference data sources (RDS) ..... 180, 212,	158, 351
396, 408, 432	SAP Add-On Installation Tool ..... 159
Referential constraint ..... 313, 346	SAP Analytics Cloud ..... 97
Remote API service consumption ..... 594	SAP API Management ..... 34, 56, 112

SAP BTP, Cloud Foundry	SAP BW/4HANA .....	55
environment .....	SAP BW Easy Queries .....	121, 123, 214
SAP BTP cockpit .....	SAP Cloud Application	
SAP BTP SDK for Android .....	Programming Model .....	530, 537
cloud configuration .....	SAP DemoContent for Features .....	383
create new project .....	SAP Developer Center .....	45
OData services .....	SAP Enterprise Portal .....	744
project configuration .....	SAP Event Mesh .....	778, 780
using the app .....	SAP Extension Suite .....	53
SAP BTP SDK for iOS .....	SAP Fiori .....	43, 56, 96, 112, 119, 125, 207,
build application .....	507, 509	
create new project .....	apps reference library .....	152, 510, 534
UI configuration .....	architecture .....	510
using the app .....	deployment .....	152
SAP Build .....	templates .....	536
SAP Build Apps .....	tools .....	529
SAP Build Process Automation .....	SAP Fiori elements .....	126, 128, 212, 328,
SAP Build Work Zone .....	413, 601	
SAP Business Accelerator Hub .....	App Preview .....	467, 480, 493
537, 779	architecture .....	127
SAP Business Application Studio .....	draft .....	500
512, 519	generate .....	423
connect to SAP Gateway .....	object page .....	333
create adaptation project .....	SAP Fiori frontend server (FES) .....	152
create dev space .....	SAP Fiori launchpad .....	153, 509, 601
create MDK project .....	extensibility .....	584
data binding .....	notifications .....	770, 772, 776
data source .....	SAP Gateway .....	27, 143, 512
deployment configuration .....	ABAP programming model for	
develop SAP Fiori apps .....	SAP Fiori .....	129
entity sets .....	add-on structure .....	116
explorer .....	advanced topics .....	769
layout editor .....	Analytics Service Generator .....	123
project attributes .....	application creation workflow .....	40
project from template .....	application requirements .....	42
project generation .....	architecture .....	111
run configurations .....	Atom .....	27
setup .....	connecting to SAP backend	
SAP Business Process Management	system .....	170
(SAP BPM) .....	consumer .....	113
SAP Business Suite .....	deployment .....	143
connecting to SAP Gateway .....	events .....	781
notifications .....	extending OData service .....	552
SAP Business Technology Platform	installation and configuration .....	143,
(SAP BTP) .....	155, 164	
extensibility .....	installation and deployment .....	45
SAP Mobile Services .....	integration and deployment .....	48
trial account .....	integration with other	
SAP Business Warehouse	technologies .....	121
(SAP BW) .....	introduction .....	19, 27
SAP Business Workflow .....	lifecycle management .....	701, 711
SAP BW .....	modern business applications .....	39

SAP Gateway (Cont.)		SAP NetWeaver (Cont.)	
monitoring	721	7.02	168
OData	27	7.31	168
OData features	101	7.40	49, 143, 144, 146, 168
open standard	39	7.50	181
operations	701, 720	7.56	168
prerequisites	155	SAP OData Provisioning	49, 51,
related products	52	112, 145	
REST	27	SAP Process Orchestration	54
SAP BW integration	214	SAP S/4HANA	43, 98, 181, 212, 339
SAP NetWeaver 7.40	49	connecting to SAP Gateway	170
SAP S/4HANA	43, 98	deployment	152
security	731	extensibility	581
service deployment	701	notifications	772
service enablement	161	OData 4.0	131
statistics	191, 788	on-premise	98
testing	701	SAP Gateway	98
tier	117	SAP S/4HANA Cloud,	
toolset	186	private edition	99
versioning	716	SAP S/4HANA Cloud,	
SAP Gateway client	189, 703, 704	public edition	98, 588
test cases	190	SAP Single Sign-On	745
SAP Gateway components		SAP Software Download Center	156
GW_CORE	46	SAP Solution Manager	98, 722, 725, 734
IW_BEP	46	SAP SuccessFactors	100
IW_FND	46, 48	SAPUI5	126, 127, 507, 513, 519, 520
IW_FNDGC	48	architecture	127
IW_GIL	47, 48	creating an application	515
IW_HDB	47	flexibility	551, 601
IW_PGW	47	framework platforms	515
IW_SCS	47	JavaScript file	516
IW_SPI	47	visual editor	605
SAP_GWFND	46	SAP Web IDE	53, 519, 541
SAP Gateway Productivity		Save sequence	471
Accelerator (GWPA)	524	Search help	
SAP GUI	32, 33	service generation	372
SAP HANA	54, 120, 127, 132	service implementation	374
architecture	132	Secure sockets layer/transport	
SAP Integration Suite	53	layer security (SSL/TLS)	732
SAP Integration Suite, advanced		Security	731
event mesh	780	cross-site request forgery	735
SAP Interactive Forms by Adobe	122	cross-site scripting	735
SAP logon ticket	744	input validation	731, 735
SAP Mobile Platform	98, 770, 771	network and communication	731
SAP Mobile Services	112, 611, 630	transport protection	731, 732
access	612	virus scan interface	739
authentication	756	Security Assertion Markup	
create mobile app	615	Language (SAML)	147, 746
SAP Mobile Start	98	Semantic annotations	95
SAP NetWeaver		Service Adaptation Definition	
7.0	167	Language (SADL)	405
7.01	167		

Service binding .....	442, 464	Service generation (Cont.)	
<i>activate</i> .....	465	<i>redefinition</i> .....	206, 338, 374
Service Builder .....	68, 116, 123, 129, 131, 161, 172, 179, 186, 201, 224, 243, 337, 432	<i>referenced data sources</i> .....	212
<i>add annotations</i> .....	413	<i>RFC/BOR Generator</i> .....	337, 340
<i>annotations</i> .....	334	<i>search help</i> .....	338, 372
<i>catalog service</i> .....	193	Service group .....	718
<i>create project</i> .....	226	<i>/IWBEF/ALL</i> .....	719
<i>error log</i> .....	190	Service implementation .....	183, 200, 223
<i>functionality</i> .....	187	<i>CDS view</i> .....	404
<i>generate runtime objects</i> .....	250	<i>map CDS views</i> .....	203
<i>mapping dialog</i> .....	204	<i>mapping search help</i> .....	205
<i>objective</i> .....	188	<i>RFC/BOR interface</i> .....	350, 362
<i>project</i> .....	187, 225	<i>RFC generation</i> .....	202
<i>service maintenance</i> .....	206, 258	<i>service development</i> .....	201
<i>service registration</i> .....	199	Service Maintenance	
Service consumption .....	467	<i>node</i> .....	163
Service consumption model .....	594	Service maintenance .....	184, 185, 199, 205, 223, 257
Service creation .....	179, 182, 185	<i>RFC/BOR interface</i> .....	348
<i>data model definition</i> .....	183, 196	Service metadata document .....	67, 69, 73
<i>incremental</i> .....	186	Service provider .....	742
<i>process overview</i> .....	196	Service Provider Interface (SPI) .....	122, 211, 337
<i>service implementation</i> .....	183, 200	<i>activate service</i> .....	381
<i>service maintenance</i> .....	184, 205	<i>create new project</i> .....	378
<i>steps</i> .....	183	Service redefinition .....	180, 206
<i>supporting tools</i> .....	189	<i>data sources</i> .....	209
<i>waterfall approach</i> .....	223	<i>GenIL</i> .....	209
Service definition .....	183, 442, 463	<i>main process steps</i> .....	207
Service deployment .....	710	<i>SPI</i> .....	211
<i>activate and maintain service</i>		Service registration .....	115, 198–200, 250, 254
<i>transaction</i> .....	717	<i>RFC/BOR interface</i> .....	347
<i>OData 4.0</i> .....	718	Service validation tool .....	706
Service development .....	179, 180, 185, 201, 223	Service with SAP annotations .....	226
<i>data model definition</i> .....	224	Service with vocabulary-based	
<i>example</i> .....	223	<i>annotations</i> .....	227
<i>navigation property</i> .....	279	Side-by-side extensibility .....	581, 592
<i>service implementation</i> .....	261	<i>advantages</i> .....	592
<i>service maintenance</i> .....	257	<i>challenges</i> .....	594
<i>service registration</i> .....	250	Sina Weibo .....	647, 663
<i>stub generation</i> .....	255	Sina Weibo development .....	663
Service document .....	67, 69, 384	<i>create user account</i> .....	664
Service generation .....	179–181, 185, 337	<i>PHP SDK</i> .....	670
<i>ABAP programming model for</i>		Single read .....	77, 267, 270, 311
<i>SAP Fiori</i> .....	339	<i>operation</i> .....	355, 364
<i>ABAP RESTful application</i>		Single sign-on .....	741
<i>programming model</i> .....	340	Skip token .....	103
<i>add-ons</i> .....	208	Social media development .....	647
<i>analytical queries</i> .....	214, 339, 382	<i>Facebook</i> .....	652
<i>CDS views</i> .....	339, 396	<i>PHP</i> .....	648
<i>OData.publish:true</i> .....	213	<i>Sina Weibo</i> .....	663

Social media development (Cont.)		Transaction (Cont.)	
<i>strategy</i>	647	<i>SMI2</i>	438
<i>X</i>	659	<i>SM59</i>	598
Software as a service (SaaS)	38	<i>SRALMANAGER</i>	764
Software development kit (SDK)	610	<i>ST22</i>	782
Sortable	231	<i>SWF_PUSH_NOTIFI</i>	774
Sorting	83	Transactional app	510
Source URI	301	Transactional behavior	471
SQL view	401	<i>managed implementation</i>	473
Star schema	390	Transactional buffer	472
Statelessness	59	Transactional key	483, 485, 496
Static feature control	502	Twitter development	659
Stub creation	347		
Subject matter experts	29		
Subscription	770		
Supportability	115		
System alias	258		
<i>transport</i>	715		
<b>T</b>		<b>U</b>	
Technical model name	200, 216, 252	Uniform resource identifier (URI)	61
Technical service name	200, 216	Unmanaged implementation type	136, 472, 490
Testing	702	Updatable	231
<i>best practices</i>	709	Update operation	78, 292, 360, 369
<i>client application</i>	706	User interface (UI)	28
<i>services</i>	703	<i>adjustments</i>	583
Total cost of ownership	35	<i>agility</i>	33
Tracing	115	<i>annotations</i>	328, 461
Transaction		<i>appeal</i>	30
<i>/IWBEP/ERROR_LOG</i>	190, 722, 784	<i>availability</i>	32
<i>/IWBEP/VIEW_LOG</i>	722	<i>business orientation</i>	30
<i>/IWFND/APPS_LOG</i>	191, 722	<i>component</i>	512
<i>/IWFND/ERROR_LOG</i>	190, 703, 722	<i>innovation</i>	31
<i>/IWFND/GW_CLIENT</i>	189, 260, 703	<i>integration</i>	33
<i>/IWFND/MAINT_SERVICE</i>	163, 205, 213, 257, 384, 432, 703, 711, 715, 717	<i>intuitive</i>	29
<i>/IWFND/MAINT_SERVICES</i>	171	<i>maintainability</i>	34
<i>/IWFND/SRV_VALIDATE</i>	706	<i>nondisruptiveness</i>	35
<i>/IWFND/STATS</i>	191, 789	<i>reduced TCO</i>	35
<i>/IWFND/TRACES</i>	192, 722, 726, 790	<i>requirements</i>	29
<i>/IWFND/V4_ADMIN</i>	493, 774	<i>security</i>	34
<i>RSRT</i>	384, 389, 391	User management	739
<i>RZ20</i>	722	User mapping	741
<i>SAINT</i>	120, 159, 165		
<i>SE11</i>	401, 559		
<i>SE16</i>	401		
<i>SE24</i>	224		
<i>SE80</i>	227, 365		
<i>SEGW</i>	116, 161, 163, 172, 179, 187, 212, 224, 378, 552		
<i>SICF</i>	162, 166		
		<b>V</b>	
		Validation	434, 484
		Value help	464
		Versioning	103, 716
		View	397
		Virtual data model (VDM)	397
		Visual Studio, Microsoft Visual C#	
		Windows Desktop	692
		Visual Studio 2022	692



W

Web API service ..... 492

Web application development ..... 508

Web services ..... 37

Wrapper function module ..... 365, 367

X

X.509 client certificate ..... 741, 745

X (formerly Twitter) ..... 647

XAMPP ..... 648

Xcode ..... 31, 629

X development ..... 659

*create developer account* ..... 659

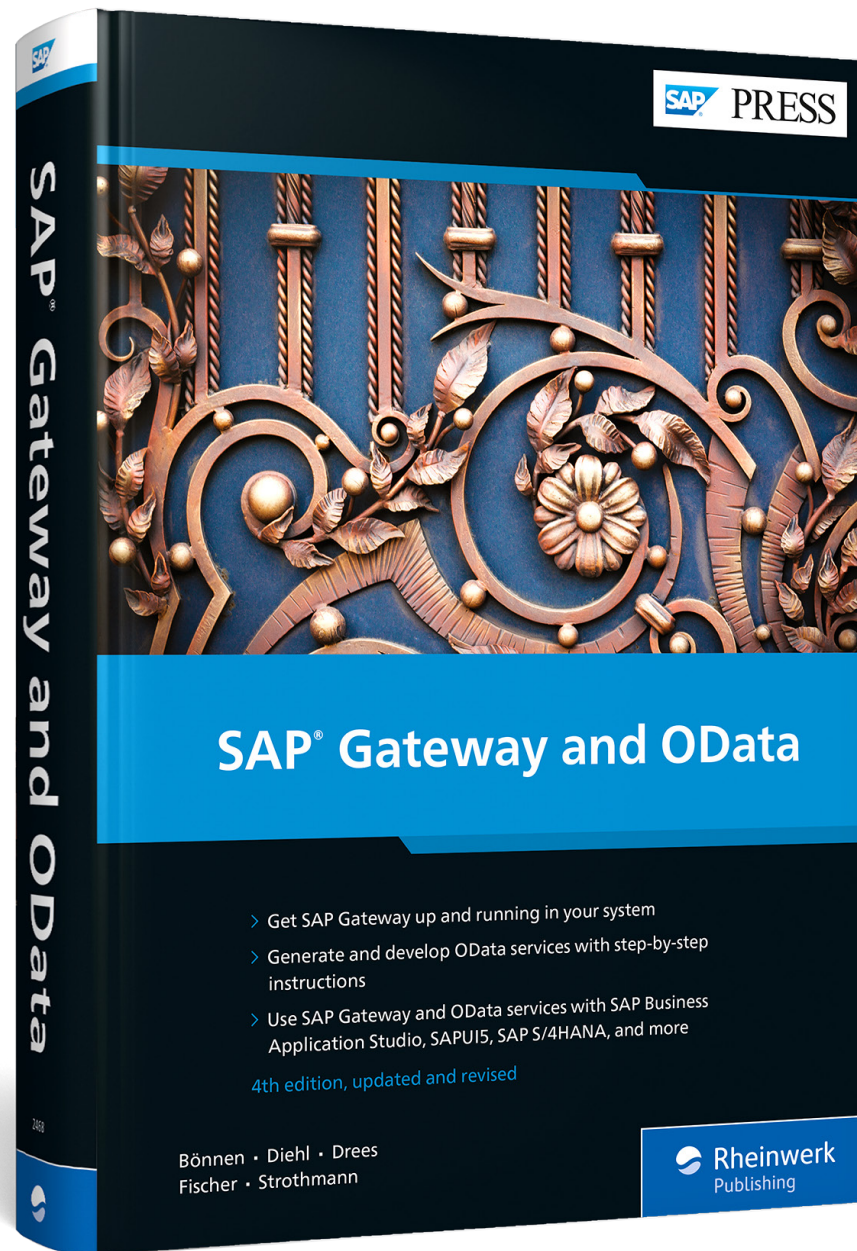
*OData tweet* ..... 663

*SDKs* ..... 659

*TwitterOAuth-library* ..... 661

XML ..... 61, 114, 724

XSRF token-based protection ..... 104



Bönner, Diehl, Drees, Fischer, Strothmann

## SAP Gateway and OData

810 pages | 12/2023 | \$89.95 | ISBN 978-1-4932-2468-5

 [www.sap-press.com/5759](http://www.sap-press.com/5759)



**Carsten Bönner** works for SAP SE as a product management director in the Adoption Enablement Team for SAP Business Technology Platform (SAP BTP). He received his MA in computer linguistics and artificial intelligence in Germany in 2001 and started working at SAP that same year. Initially a Java developer and trainer, he soon became a consultant and led strategic projects in the then-new field of enterprise portals. Since 2002, he has worked as a product manager for SAP NetWeaver Portal, SAP NetWeaver Visual Composer, SAP Gateway, and SAP API Management. For another four years, he has worked as the director for technology strategy for the strategic alliance management at Microsoft.



**Ludwig Diehl** is the head of processes and IT at Wilkhahn, the worldwide acting and leading German manufacturer of furniture. In this role, Ludwig is responsible for digitalization, business processes, and IT services and infrastructure. He has previously worked in this field in various roles and at various companies. Ludwig studied business informatics at the University of Applied Sciences FHDW in Bergisch Gladbach, Germany. He was a member of the SAP Design Partner Council for SAP Gateway from 2011 to 2013. In this role, he worked with SAP and other partners to improve SAP Gateway.



Volker Drees studied electrical engineering at Fachhochschule in Wiesbaden, Germany, and holds a degree in communications engineering (Nachrichtentechnik). He began his SAP career in 1998 in the consulting department and has experience in a number of areas: ABAP development, R/3 implementations, mySAP CRM, mobile sales, mobile asset management, and mobile infrastructure. He recently worked as a regional group expert for mobile applications in the Business User and Information Worker Division at SAP. Currently, Volker works at SAP SE as a product expert for SAP Gateway and the ABAP programming model for SAP Fiori in the Products & Innovation, Core Platform Division.



**André Fischer** has worked in product management for SAP Gateway since the launch of the product in 2011. His current focuses in the product management of the ABAP platform are the ABAP RESTful application programming model and SAP BTP, ABAP environment. After finishing his physics degree at RWTH Aachen University and Heidelberg University, Germany, he started his professional career in 1995 as a technology and security consultant for an SAP partner. In 2004, he joined SAP. André is a frequent speaker at conferences, including SAP TechEd, and he has published a multitude of articles and blogs on the SAP Community. With more than 30 years of experience in various SAP technologies, André is a trusted advisor for many SAP customers and partners.



**Karsten Strothmann** is the lead product manager for event-driven integration at SAP SE in Wall-dorf, Germany. He has 25 years of experience in the software industry, more than 20 of which at SAP. Karsten is currently working on event-driven architectures and SAP's event-driven technology ecosystem. Previously, Karsten worked on SAP Gateway for 6 years, from its very beginnings through to its fruition. During his career, he has applied himself to highly varied roles, from product management, to development, to quality assurance, to consulting. This has given him a holistic view of both software creation and its usage. Karsten holds a master's degree in computer science from Technical University Dortmund, Germany.

*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.*