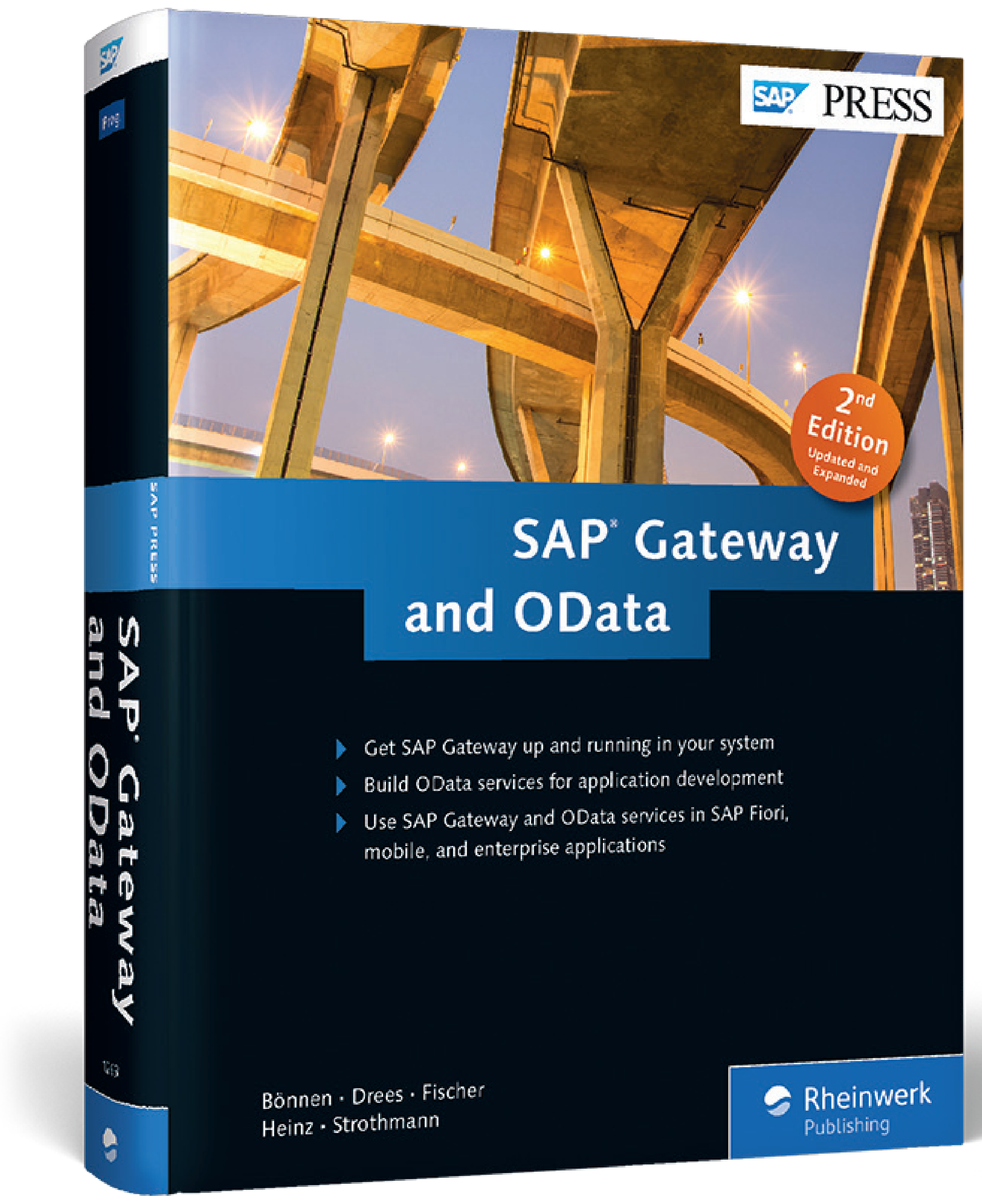
First-hand knowledge.



Reading Sample

In this reading sample, we look at Chapter 5, Introduction to OData Service Creation, which provides an overview of the service development and service generation process. The benefits of each method is presented along with the individual sub-steps in the processes. This chapter forms the foundation for Chapter 6 and 7.

Carsten Bönnen, Volker Drees, André Fischer, Ludwig Heinz, Karsten Strothmann

SAP Gateway and OData

785 Pages, 2016, $79.95
ISBN 978-1-4932-1263-7

www.sap-press.com/3904

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.

5 Introduction to OData Service Creation

Out-of-the-box OData services

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.

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, it 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).

The main objective of this chapter is to give you an overview of the process of service creation, which we then discuss in more detail in Chapter 6 and Chapter 7. To achieve this, in Section 5.1, we give you a brief overview of 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: 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 deeper into service creation and look in more detail at 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 have been delivered by SAP. Finally, we give you an introduction to the development paradigm used for service development: the OData channel (Section 5.5).

5.1 Methods for Creating an OData Service

There are two ways to create OData services with SAP Gateway:

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 way 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 CRUD-Q methods of an entity set to a data source. This is supported for the following data sources
    - Remote function call (RFC)/Business Object Repository (BOR) function modules
    - Search help (only `READ` and `QUERY` method)
    - Core Data Services (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*: Allows you to define a service based on a CDS view.
  - *Creating CDS views with Eclipse*: Generate 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, it's more limited, and thus is primarily recommended for developing very straightforward services. Service generation doesn't give you much optimization potential because, without custom coding, you're restricted to what the service generators offer. In most real-world situations, you'll want to opt for service development because the advantages are well worth the effort. Still, if you have search helps, CDS views, Generic Interaction Layer (GenIL) or Service Provider Interface objects, analytical queries such as SAP Business Warehouse (BW) Easy Queries, or a suitable RFC function module or Business Application Programming Interface (BAPI) and are aiming for a quick result, this might be an option for you.

Service creation process

However, with the advent of SAP S/4HANA, OData services based on CDS views can be generated to support the draft infrastructure. As shown in Figure 5.1, option ❶ will become the preferred approach for OData service development. Because this kind of service will also be able to support smart templates for user interface (UI) development, a lot of scenarios in SAP S/4HANA won't require SAPUI5 coding but the development of appropriate CDS views and Business Object Processing Framework (BOPF) objects.

Even when using OData services that are generated from CDS views, the execution of the Service Adaptation Definition Language (SADL) interface can be fine-tuned by implementing its query application programming interface (API) or by adding additional business logic in the data provider extension class. (We'll go into more detail about these specific options in Chapter 7, where we discuss service generation in detail.)

In systems that are based on SAP NetWeaver 7.50, it's still possible to develop OData services using service development and the mapping of data sources (see Figure 5.1, ❷). This way, customers will be able to leverage their existing resources such as ABAP classes and RFC function modules when using SAP Business Suite EHP 8 or higher or when using SAP S/4HANA on-premise.

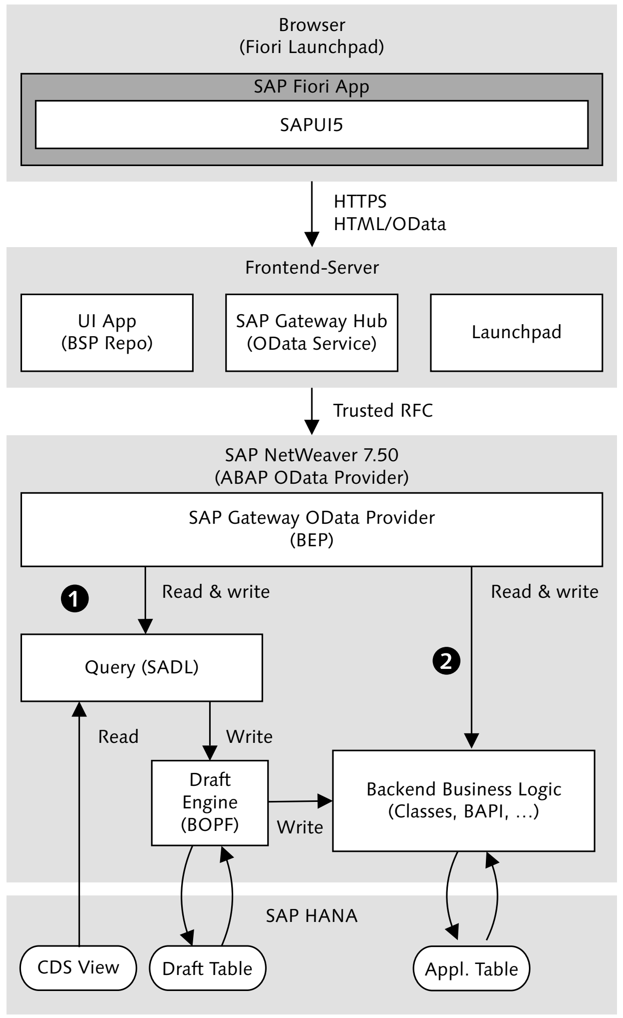


Figure 5.1 SAP Gateway OData Service Provisioning for SAP Fiori: The Transformation to SAP S/4HANA

Service creation process

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 the steps can also be performed out of order (an incremental approach). We'll go into a bit more detail about this 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 go for service development or 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 have to complete the process of *service definition* as a prerequisite. This is the process of identifying what service to create and specifying its details. Ideally, you've done all of this together with the client developers so that you know exactly what data they require and how this works with the artifacts in the SAP Business Suite 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.

Data model definition phase

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 Business Suite system so that you can proceed with the next main phase, *service implementation*.

Service implementation phase

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

- 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 have to 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 Eclipse 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, this means that the created OData service can then be consumed.

The three phases—data model definition, service implementation, and service maintenance—are depicted 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 have to be performed in both the development and generation of OData services in SAP Gateway are marked with both colors.

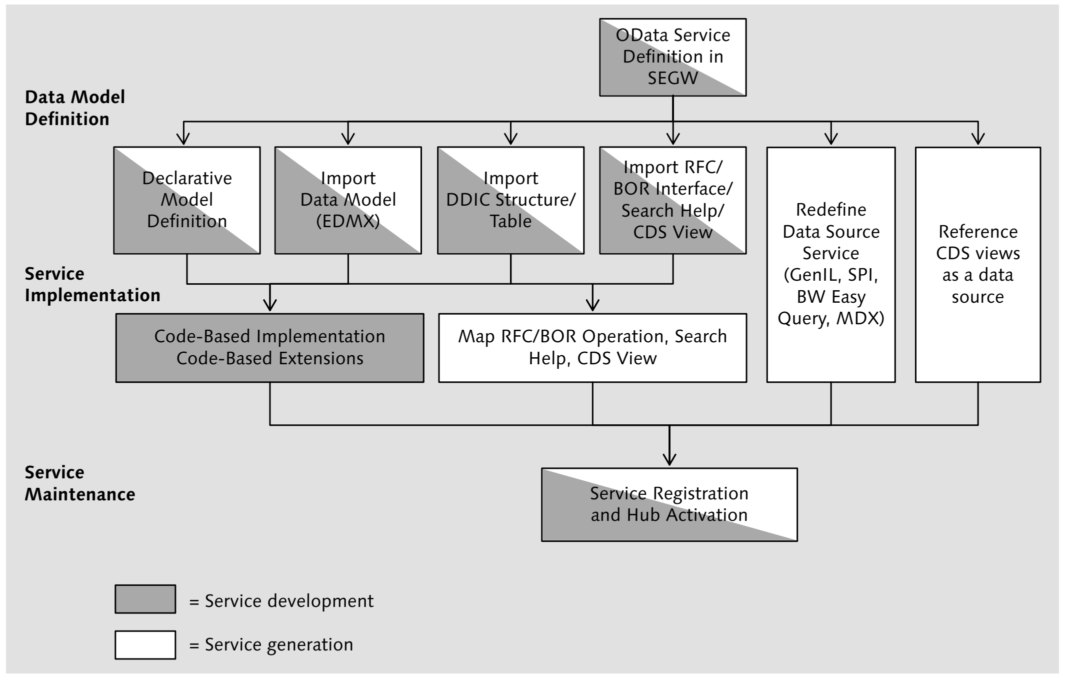

OData Service Definition in SEGW
Data Model Definition
Declarative Model Definition
Import Data Model (EDMX)
Import DDIC Structure/ Table
Import RFC/ BOR Interface/ Search Help/ CDS View
Redefine Data Source Service (GenIL, SPI, BW Easy Query, MDX)
Reference CDS views as a data source
Service Implementation
Code-Based Implementation Code-Based Extensions
Map RFC/BOR Operation, Search Help, CDS View
Service Maintenance
Service Registration and Hub Activation
= Service development
= Service generation

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). It's also possible to generate a read-only OData service based on an SAP BW query and extend the same via code-based implementation so that it also supports updating business data.

Incremental service creation process

As previously mentioned, we've presented the service creation process in a very 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

The service publication is a one-time activity as long as you don't perform major changes. Registering the service for additional SAP Business Suite systems, for example, is such an activity in which you have to 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 always 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 metadata (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 depicts the incremental service creation process. It's based on Figure 5.2 and adds incremental steps to the original process. These incremental steps are displayed by the solid line arrows that depict potential transitions among the three phases of data model definition, service implementation, and service maintenance, which are symbolized by the horizontal boxes. The dotted line stands for the one-time activity of service publication as part of the service maintenance phase.

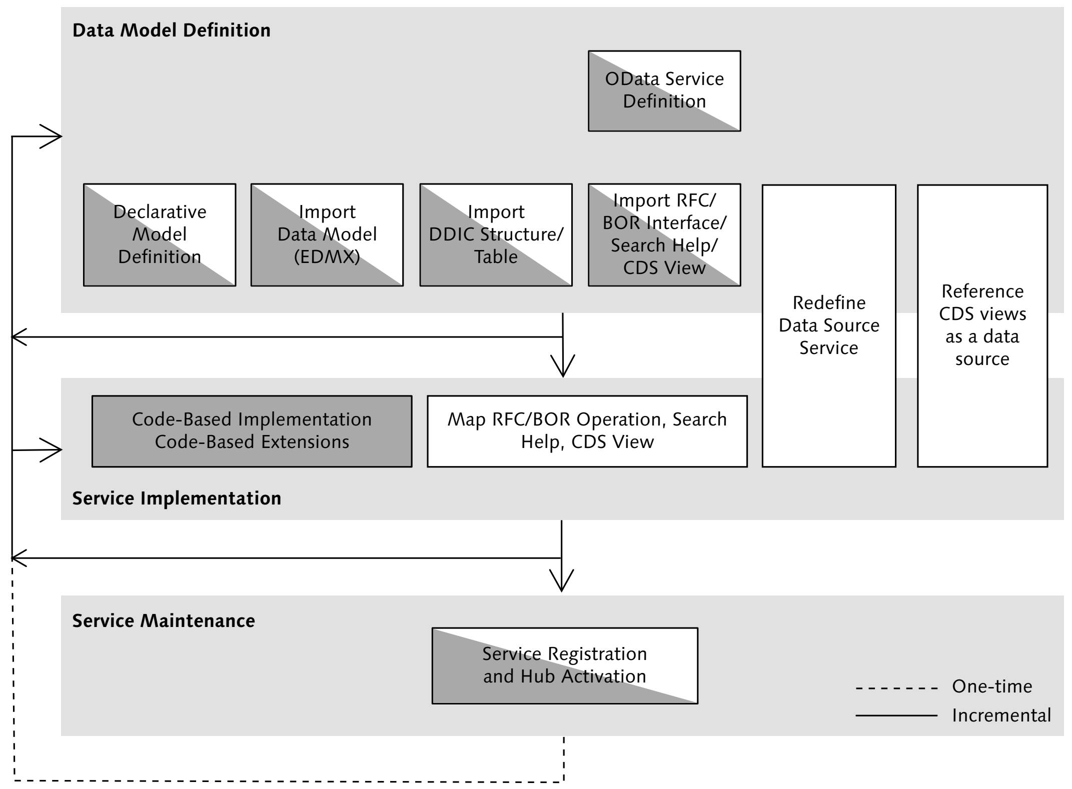

Data Model Definition
OData Service Definition
Declarative Model Definition
Import Data Model (EDMX)
Import DDIC Structure/ Table
Import RFC/ BOR Interface/ Search Help/ CDS View
Redefine Data Source Service
Reference CDS views as a data source
Code-Based Implementation Code-Based Extensions
Map RFC/BOR Operation, Search Help, CDS View
Service Implementation
Service Maintenance
Service Registration and Hub Activation
One-time
Incremental

Figure 5.3 Incremental Service Creation

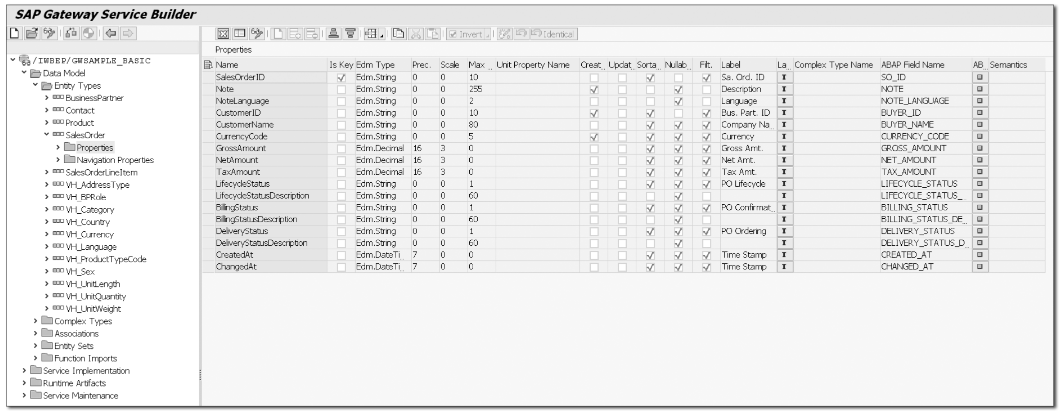
5.3 SAP Gateway Toolset

SAP Gateway provides a set of tools to address all 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 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

Supports development lifecycle of an OData service

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 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 (Figure 5.4).



```
˅ /IWBEP/GWSAMPLE_BASIC
  ˅ Data Model
    ˅ Entity Types
      › BusinessPartner
      › Contact
      › Product
      ˅ SalesOrder
        › Properties
        › Navigation Properties
      › SalesOrderLineItem
      › VH_AddressType
      › VH_BPRole
      › VH_Category
      › VH_Country
      › VH_Currency
      › VH_Language
      › VH_ProductTypeCode
      › VH_Sex
      › VH_UnitLength
      › VH_UnitQuantity
      › VH_UnitWeight
    › Complex Types
    › Associations
    › Entity Sets
    › Function Imports
  › Service Implementation
  › Runtime Artifacts
  › Service Maintenance
```

Properties

Name	Is Key	Edm Type	Prec.	Scale	Max ...	Unit Property Name	Creat...	Updat...	Sorta...	Nullab...	Filt.	Label	La...	Complex Type Name	ABAP Field Name	AB...	Semantics
SalesOrderID	✓	Edm.String	0	0	10				✓		✓	Sa. Ord. ID			SO_ID		
Note		Edm.String	0	0	255		✓			✓		Description			NOTE		
NoteLanguage		Edm.String	0	0	2					✓		Language			NOTE_LANGUAGE		
CustomerID		Edm.String	0	0	10		✓		✓		✓	Bus. Part. ID			BUYER_ID		
CustomerName		Edm.String	0	0	80				✓	✓	✓	Company Na...			BUYER_NAME		
CurrencyCode		Edm.String	0	0	5		✓		✓	✓	✓	Currency			CURRENCY_CODE		
GrossAmount		Edm.Decimal	16	3	0				✓	✓	✓	Gross Amt.			GROSS_AMOUNT		
NetAmount		Edm.Decimal	16	3	0				✓	✓	✓	Net Amt.			NET_AMOUNT		
TaxAmount		Edm.Decimal	16	3	0				✓	✓	✓	Tax Amt.			TAX_AMOUNT		
LifecycleStatus		Edm.String	0	0	1				✓	✓	✓	PO Lifecycle			LIFECYCLE_STATUS		
LifecycleStatusDescription		Edm.String	0	0	60					✓					LIFECYCLE_STATUS_...		
BillingStatus		Edm.String	0	0	1				✓	✓	✓	PO Confirmat...			BILLING_STATUS		
BillingStatusDescription		Edm.String	0	0	60					✓					BILLING_STATUS_DE...		
DeliveryStatus		Edm.String	0	0	1				✓	✓	✓	PO Ordering			DELIVERY_STATUS		
DeliveryStatusDescription		Edm.String	0	0	60					✓					DELIVERY_STATUS_D...		
CreatedAt		Edm.DateTi...	7	0	0				✓	✓	✓	Time Stamp			CREATED_AT		
ChangedAt		Edm.DateTi...	7	0	0				✓	✓	✓	Time Stamp			CHANGED_AT		

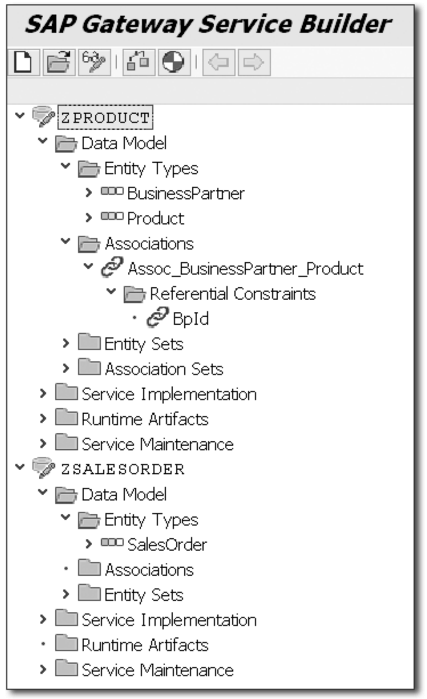
Figure 5.4 Service Builder

Overall, the Service Builder addresses the needs of both experienced and less experienced developers, as well as nondevelopers. 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.

Service Builder allows for centrally displaying and creating the definition of an OData service. This includes runtime artifacts (model provider class [MPC], data provider class [DPC], model, and service), OData artifacts (entity set, entity type, and properties), as well as data sources and models.

Project-based development

The modeling environment follows a project-based approach, and all relevant data are consolidated in 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 artifacts that are needed for service development in one central place, thereby providing a means to organize the development process. The Service Builder allows the developer to open several projects at the same time as shown in Figure 5.5 (in this example, ZPRODUCT and ZSALESORDER).



```
˅ ZPRODUCT
  ˅ Data Model
    ˅ Entity Types
      › BusinessPartner
      › Product
    ˅ Associations
      ˅ Assoc_BusinessPartner_Product
        ˅ Referential Constraints
          · BpId
    › Entity Sets
    › Association Sets
  › Service Implementation
  › Runtime Artifacts
  › Service Maintenance
˅ ZSALESORDER
  ˅ Data Model
    ˅ Entity Types
      › SalesOrder
    · Associations
    › Entity Sets
  › Service Implementation
  · Runtime Artifacts
  › Service Maintenance
```

Figure 5.5 Project-Based Development

Note

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 Business Suite 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 release 7.31.

As of SAP NetWeaver release 7.40 SP 02, the BEP component is included in SAP NetWeaver itself as part of the SAP_GWFND component. As a result, it's possible to perform development of OData services using the Service Builder without additional effort in all systems after they run on top of SAP NetWeaver 7.40 SP 02 or later.

Because the Service Builder is part of the BEP component that is typically (but not necessarily) installed on the SAP Business Suite 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 is important to understand if it comes to 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.

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 Business Suite system. However, there are restrictions in what can be declared or generated. Advanced OData features may need to be implemented manually, and certain operations aren't available in a refined 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.

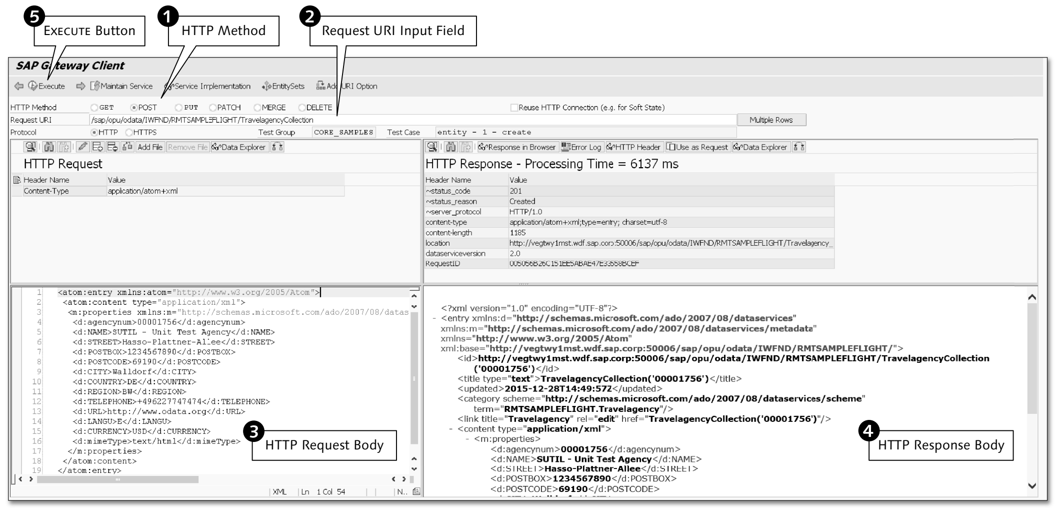
5.3.2 Beyond the Service Builder: Supporting Tools during the Service Creation Process

As stated, the main tool during the service creation process is the Service Builder. At the same time, SAP Gateway provides additional tools that are very useful during the development of SAP Gateway services. These tools allow, for example, for early testing of services or tracing what is happening when calling a service. As such, this section aims to briefly introduce you to some of the functionalities. For a more comprehensive description of the development support and administration toolset of SAP Gateway, see Chapter 14.

SAP Gateway Client

Testing and troubleshooting

The SAP Gateway client can be used for both testing and troubleshooting and is a Representational State Transfer (REST) client built into SAP Gateway. It can be started from within SAP GUI using Transaction /IWFND/GW_CLIENT. After you've created a service, you can use this tool for a first test, as shown in Figure 5.6.



HTTP Method: GET [✓ POST] PUT PATCH MERGE DELETE
Reuse HTTP Connection (e.g. for Soft State): ☐
Request URI: /sap/opu/odata/IWFND/RMTSAMPLEFLIGHT/TravelagencyCollection
Protocol: [✓ HTTP] HTTPS
Test Group: CORE_SAMPLES
Test Case: entity - 1 - create

HTTP Request

Header Name	Value
Content-Type	application/atom+xml

HTTP Response - Processing Time = 6137 ms

Header Name	Value
~status_code	201
~status_reason	Created
~server_protocol	HTTP/1.0
content-type	application/atom+xml;type=entry; charset=utf-8
content-length	1185
location	http://vegtwy1mst.wdf.sap.corp:50006/sap/opu/odata/IWFND/RMTSAMPLEFLIGHT/Travelagency...
dataserviceversion	2.0
RequestID	005056B26C151EE5ABAE47E33558BCEF

```
<atom:entry xmlns:atom="http://www.w3.org/2005/Atom">
 <atom:content type="application/xml">
  <m:properties xmlns:m="http://schemas.microsoft.com/ado/2007/08/datas
   <d:agencynum>00001756</d:agencynum>
   <d:NAME>SUTIL - Unit Test Agency</d:NAME>
   <d:STREET>Hasso-Plattner-Allee</d:STREET>
   <d:POSTBOX>1234567890</d:POSTBOX>
   <d:POSTCODE>69190</d:POSTCODE>
   <d:CITY>Walldorf</d:CITY>
   <d:COUNTRY>DE</d:COUNTRY>
   <d:REGION>BW</d:REGION>
   <d:TELEPHONE>+496227747474</d:TELEPHONE>
   <d:URL>http://www.odata.org</d:URL>
   <d:LANGU>E</d:LANGU>
   <d:CURRENCY>USD</d:CURRENCY>
   <d:mimeType>text/html</d:mimeType>
  </m:properties>
 </atom:content>
</atom:entry>
```

```
  <?xml version="1.0" encoding="UTF-8"?>
- <entry xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices"
  xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata"
  xmlns="http://www.w3.org/2005/Atom"
  xml:base="http://vegtwy1mst.wdf.sap.corp:50006/sap/opu/odata/IWFND/RMTSAMPLEFLIGHT/">
     <id>http://vegtwy1mst.wdf.sap.corp:50006/sap/opu/odata/IWFND/RMTSAMPLEFLIGHT/TravelagencyCollection
        ('00001756')</id>
     <title type="text">TravelagencyCollection('00001756')</title>
     <updated>2015-12-28T14:49:57Z</updated>
     <category scheme="http://schemas.microsoft.com/ado/2007/08/dataservices/scheme"
        term="RMTSAMPLEFLIGHT.Travelagency"/>
     <link title="Travelagency" rel="edit" href="TravelagencyCollection('00001756')"/>
   - <content type="application/xml">
     - <m:properties>
          <d:agencynum>00001756</d:agencynum>
          <d:NAME>SUTIL - Unit Test Agency</d:NAME>
          <d:STREET>Hasso-Plattner-Allee</d:STREET>
          <d:POSTBOX>1234567890</d:POSTBOX>
          <d:POSTCODE>69190</d:POSTCODE>
```

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 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, it's possible to use the 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 choosing Execute ❺.

Test cases

A very 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 70 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 have to be manually created from within the SAP Gateway client by selecting ❶ SAP Gateway Client and ❷ Create Core Samples from the menu as shown in Figure 5.7.

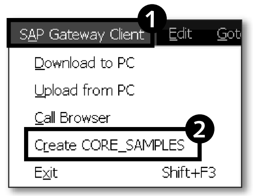


Figure 5.7 Creating Core Samples from within the SAP Gateway Client

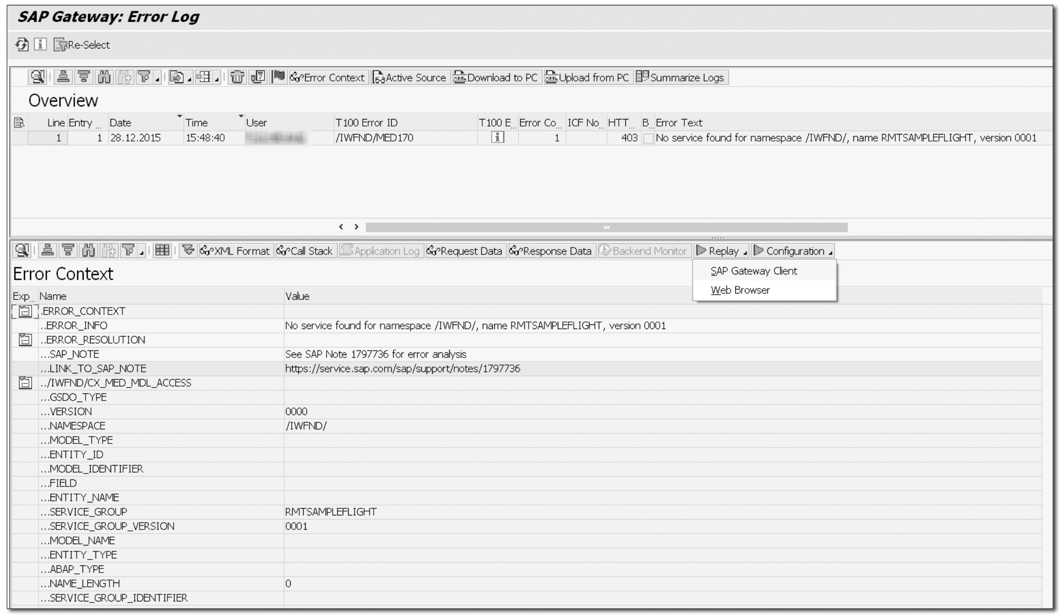
If you've saved a request as a test case, you can add or change the expected HTTP return code. A request can return multiple HTTP return codes that are valid (e.g., `200`, `401`, `402`, and `403`). Therefore, multiple statuses, including status ranges separated by a dash, can be entered (e.g., `201 401-403`). In addition, it's possible to 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 very 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 Business Suite system error log with a similar UI available that can be used to analyze errors that occurred in the SAP Business Suite system via Transaction /IWBEP/ERROR_LOG.

The error log is tightly integrated with the SAP Gateway client, so it's possible to rerun a request sent by a consumer that led to errors by selecting Replay • SAP Gateway Client as shown in Figure 5.8.



Line	Entry ...	Date	Time	User	T100 Error ID	T100 E...	Error Co...	ICF No...	HTT...	B...	Error Text
1	1	28.12.2015	15:48:40	[redacted]	/IWFND/MED170	i	1		403	☐	No service found for namespace /IWFND/, name RMTSAMPLEFLIGHT, version 0001

Exp...	Name	Value
▤	.ERROR_CONTEXT	
	..ERROR_INFO	No service found for namespace /IWFND/, name RMTSAMPLEFLIGHT, version 0001
▤	..ERROR_RESOLUTION	
	...SAP_NOTE	See SAP Note 1797736 for error analysis
	...LINK_TO_SAP_NOTE	https://service.sap.com/sap/support/notes/1797736
▤	../IWFND/CX_MED_MDL_ACCESS	
	...GSDO_TYPE	
	...VERSION	0000
	...NAMESPACE	/IWFND/
	...MODEL_TYPE	
	...ENTITY_ID	
	...MODEL_IDENTIFIER	
	...FIELD	
	...ENTITY_NAME	
	...SERVICE_GROUP	RMTSAMPLEFLIGHT
	...SERVICE_GROUP_VERSION	0001
	...MODEL_NAME	
	...ENTITY_TYPE	
	...ABAP_TYPE	
	...NAME_LENGTH	0
	...SERVICE_GROUP_IDENTIFIER	

Figure 5.8 Transaction /IWFND/ERROR_LOG

Logging and tracing

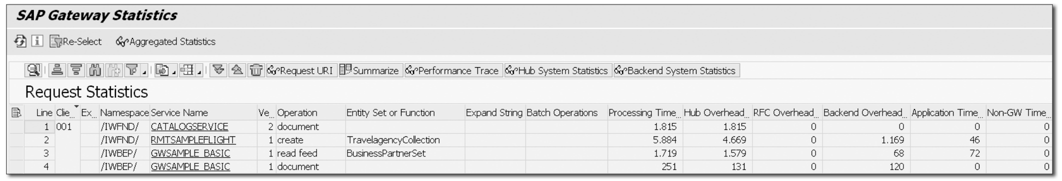
As another way to dig into potential problems, monitoring log entries can be generated for the system log and the application log of SAP Gateway. To access the system log, use Transaction SM21; to access the application log, use Transaction /IWFND/APPS_LOG.

SAP Gateway Statistics and Payload Trace

When developing an OData service or a client application, the developer will want to know about the performance of the service. The 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 the performance statistic 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.

Based on this data, Transaction /IWFND/STATS (SAP Gateway Statistics) provides a detailed statistics view of each service call handled by SAP

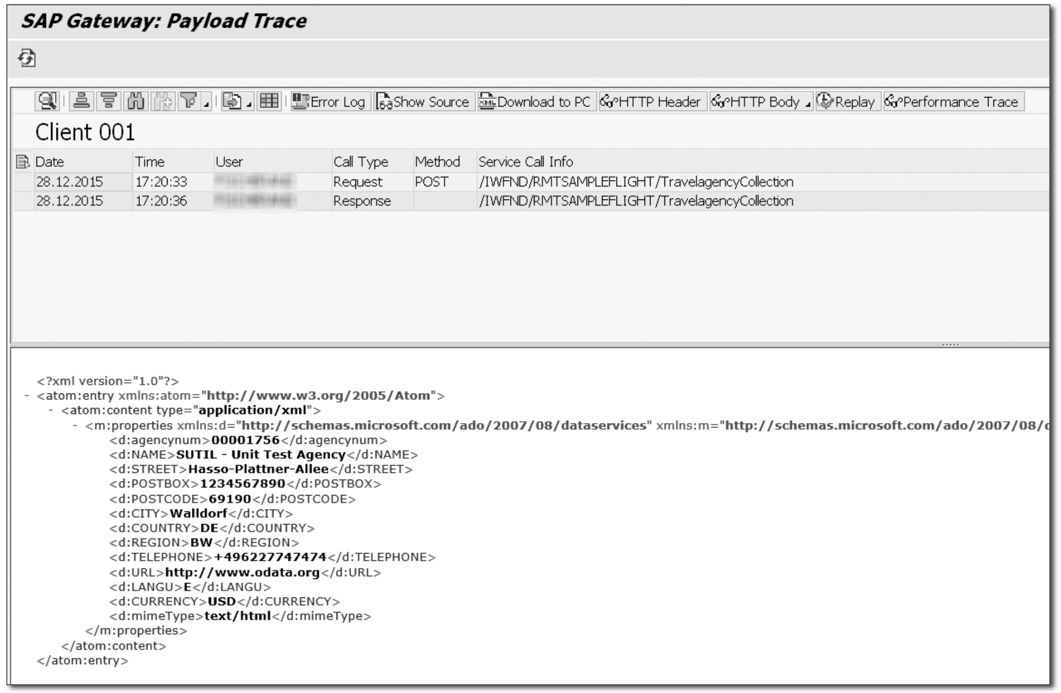
Gateway. The data are aggregated on a regular basis so that statistical data for each service can be analyzed easily. In a productive system, the transaction is of great value for the system administrator to check the performance of the OData services (see Figure 5.9).



Line	Clie…	Ex…	Namespace	Service Name	Ve…	Operation	Entity Set or Function	Expand String	Batch Operations	Processing Time…	Hub Overhead…	RFC Overhead…	Backend Overhead…	Application Time…	Non-GW Time…
1	001		/IWFND/	CATALOGSERVICE	2	document				1.815	1.815	0	0	0	0
2			/IWFND/	RMTSAMPLEFLIGHT	1	create	TravelagencyCollection			5.884	4.669	0	1.169	46	0
3			/IWBEP/	GWSAMPLE_BASIC	1	read feed	BusinessPartnerSet			1.719	1.579	0	68	72	0
4			/IWBEP/	GWSAMPLE_BASIC	1	document				251	131	0	120	0	0

Figure 5.9 Transaction /IWFND/STATS

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



Date	Time	User	Call Type	Method	Service Call Info
28.12.2015	17:20:33	[redacted]	Request	POST	/IWFND/RMTSAMPLEFLIGHT/TravelagencyCollection
28.12.2015	17:20:36	[redacted]	Response		/IWFND/RMTSAMPLEFLIGHT/TravelagencyCollection

```
  <?xml version="1.0"?>
- <atom:entry xmlns:atom="http://www.w3.org/2005/Atom">
  - <atom:content type="application/xml">
    - <m:properties xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices" xmlns:m="http://schemas.microsoft.com/ado/2007/08/
        <d:agencynum>00001756</d:agencynum>
        <d:NAME>SUTIL - Unit Test Agency</d:NAME>
        <d:STREET>Hasso-Plattner-Allee</d:STREET>
        <d:POSTBOX>1234567890</d:POSTBOX>
        <d:POSTCODE>69190</d:POSTCODE>
        <d:CITY>Walldorf</d:CITY>
        <d:COUNTRY>DE</d:COUNTRY>
        <d:REGION>BW</d:REGION>
        <d:TELEPHONE>+496227747474</d:TELEPHONE>
        <d:URL>http://www.odata.org</d:URL>
        <d:LANGU>E</d:LANGU>
        <d:CURRENCY>USD</d:CURRENCY>
        <d:mimeType>text/html</d:mimeType>
      </m:properties>
    </atom:content>
  </atom:entry>
```

Figure 5.10 Transaction /IWFND/TRACES

Using the payload trace, it's even possible to 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

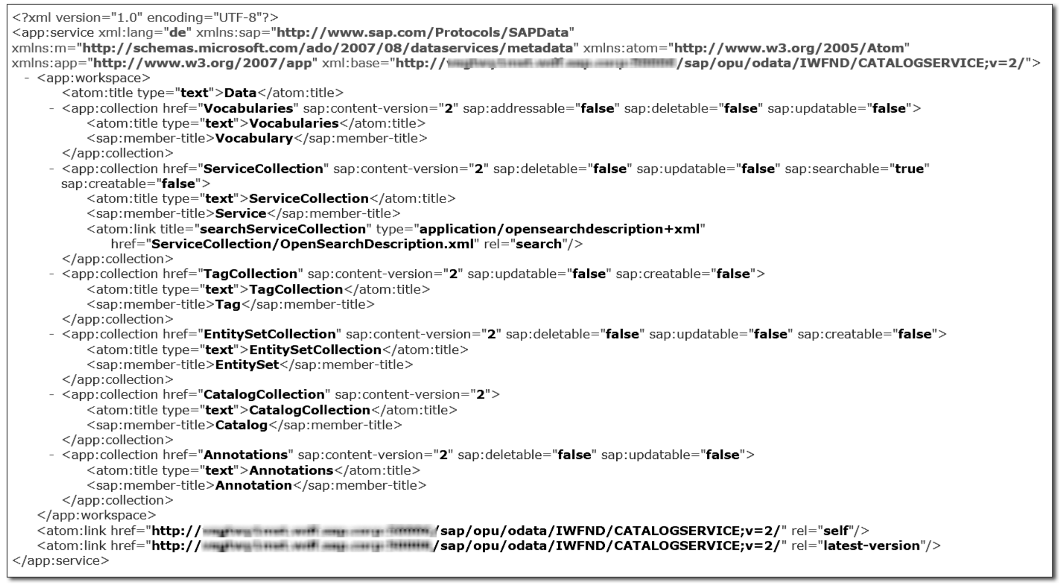
The replay capability can also be used to create test cases in the SAP Gateway client for your service in a convenient way. To use the performance and payload trace, it's necessary to activate those traces.

We'll discuss the SAP Gateway statistics transaction and the SAP Gateway performance and payload trace tool in more detail in Appendix A.

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 (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/CatalogCollection



```
<?xml version="1.0" encoding="UTF-8"?>
<app:service xml:lang="de" xmlns:sap="http://www.sap.com/Protocols/SAPData"
xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata" xmlns:atom="http://www.w3.org/2005/Atom"
xmlns:app="http://www.w3.org/2007/app" xml:base="http://[redacted]/sap/opu/odata/IWFND/CATALOGSERVICE;v=2/">
 - <app:workspace>
     <atom:title type="text">Data</atom:title>
   - <app:collection href="Vocabularies" sap:content-version="2" sap:addressable="false" sap:deletable="false" sap:updatable="false">
        <atom:title type="text">Vocabularies</atom:title>
        <sap:member-title>Vocabulary</sap:member-title>
     </app:collection>
   - <app:collection href="ServiceCollection" sap:content-version="2" sap:deletable="false" sap:updatable="false" sap:searchable="true"
     sap:creatable="false">
        <atom:title type="text">ServiceCollection</atom:title>
        <sap:member-title>Service</sap:member-title>
        <atom:link title="searchServiceCollection" type="application/opensearchdescription+xml"
            href="ServiceCollection/OpenSearchDescription.xml" rel="search"/>
     </app:collection>
   - <app:collection href="TagCollection" sap:content-version="2" sap:updatable="false" sap:creatable="false">
        <atom:title type="text">TagCollection</atom:title>
        <sap:member-title>Tag</sap:member-title>
     </app:collection>
   - <app:collection href="EntitySetCollection" sap:content-version="2" sap:deletable="false" sap:updatable="false" sap:creatable="false">
        <atom:title type="text">EntitySetCollection</atom:title>
        <sap:member-title>EntitySet</sap:member-title>
     </app:collection>
   - <app:collection href="CatalogCollection" sap:content-version="2">
        <atom:title type="text">CatalogCollection</atom:title>
        <sap:member-title>Catalog</sap:member-title>
     </app:collection>
   - <app:collection href="Annotations" sap:content-version="2" sap:deletable="false" sap:updatable="false">
        <atom:title type="text">Annotations</atom:title>
        <sap:member-title>Annotation</sap:member-title>
     </app:collection>
   </app:workspace>
   <atom:link href="http://[redacted]/sap/opu/odata/IWFND/CATALOGSERVICE;v=2/" rel="self"/>
   <atom:link href="http://[redacted]/sap/opu/odata/IWFND/CATALOGSERVICE;v=2/" rel="latest-version"/>
</app:service>
```

Figure 5.11 Service Catalog: Service Document

OpenSearch 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 that can be retrieved using the following URL: *http://<server>:<port>/sap/opu/odata/iwfnd/CATALOGSERVICE/ServiceCollection/OpenSearchDescription.xml*.

5.3.3 ABAP Development Tools for SAP NetWeaver and CDS Views

CDS views – one concept, two flavors 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 is needed because business data are usually distributed across several database tables.

CDS provide a specification for an SQL-based Data Definition Language (DDL). With SAP HANA CDS and ABAP CDS, there are two flavors of this specification available. Whereas SAP HANA CDS views only need to run on top of SAP HANA, ABAP CDS views have to support multiple databases. This is similar to the ABAP Open SQL syntax, which is the last common denominator of the different SQL dialects supported by SAP NetWeaver AS ABAP.

Additional Resources

You'll find a comprehensive and detailed comparison between ABAP CDS views and SAP HANA CDS views at *http://scn.sap.com/community/abap/blog/2015/07/20/cds--one-model-two-flavors*.

Let's look at the example in Listing 5.1 of an ABAP CDS view, which was taken from SAP online documentation at (*http://help.sap.com/saphelp_nw75/helpdata/en/7c/078765ec6d4e6b88b71bdaf8a2bd9f/content.htm*).

```
@AbapCatalog.sqlViewName: 'CUSTOMER_VW'

DEFINE VIEW cust_book_view_entity AS SELECT FROM scustom
     JOIN sbook
     ON scustom.id = sbook.customid
     {
           scustom.id,
           scustom.name,
           sbook.bookid
     }
```

Listing 5.1 Example of an ABAP CDS View

The CDS `cust_book_view_entity` entity creates a join on the two database tables `scustom` and `sbook`, which are part of the SFLIGHT demo data model. As a result, it's possible to access the data via the ABAP Open SQL statement in Listing 5.2.

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

Listing 5.2 Using ABAP CDS Views in ABAP Code

ABAP Development Tools for SAP NetWeaver A CDS view can be defined using the Eclipse-based ABAP Development Tools for SAP NetWeaver using the ABAP CDS statement `DEFINE VIEW`. This will create two objects in the ABAP DDIC, namely an SQL view and the CDS entity, as shown in Figure 5.12.

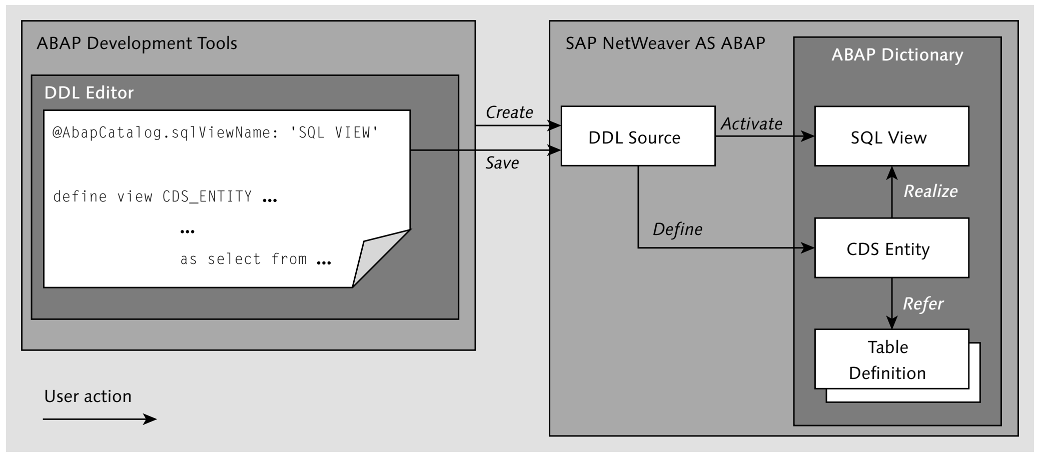


Figure 5.12 ABAP CDS View Building Architecture

Note

The SQL view and the CDS entity are created in the same namespace in the ABAP DDIC. As a result, both names have to be different. In Listing 5.1, the SQL view is therefore denoted as `CUSTOMER_VW`, whereas the CDS entity is denoted as `cust_book_view_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'll find it useful to refer to 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 that have either been defined by the OData Model Editor of the SAP Web IDE or the entity data modeler provided by Microsoft Visual Studio. In addition, it's possible to import 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 Business Suite system. This can be done by the import of DDIC structures/tables or by the generation of new entity types based on an RFC/BOR interface or a search help.

Next, we'll discuss all four options in a bit more detail.

Declarative Data Model

Entity types

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. (To model an OData service from scratch in WYSIWYG style, alternative OData modeling tools, such as the SAP Web IDE [see Chapter 9] and Microsoft Visual Studio, are better. However, in these cases, the model has to then be imported into the Service Builder.)

Import Data Model via EDMX

Using the import model option, the developer can import a complete OData model stored in an EDMX file, or a metadata document of an existing OData service, into the Service Builder. This 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 service metadata files of an existing OData service. If you perform an import on 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 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

DDIC type support

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 Business Suite system, you can import the following DDIC types into the Service Builder:

- 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, the 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 very 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 they are being used to access the data in the SAP Business Suite 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 Search Helps. Again a wizard is provided to guide you through the process. This wizard even performs the mapping of the `READ` and `QUERY` method in the same step so that there's no need for a separate service implementation step.

5.4.2 Service Registration in the SAP Business Suite System

After the data model is defined, it must then be registered. Service registration in the SAP Business Suite manifests the data model definition phase's results. This means that 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 Business Suite.

Service Registration versus Service Maintenance

As you may recall from Section 5.2, the service maintenance phase of service creation involves activating and registering the service on the SAP Gateway server. This isn't to be confused with service registration in the SAP Business Suite system, which is a process that occurs after the data model definition. In this section, we're focusing on service registration in the SAP Business Suite system. In Section 5.4.4, we'll discuss service maintenance.

The difference between service registration and service maintenance is as follows:

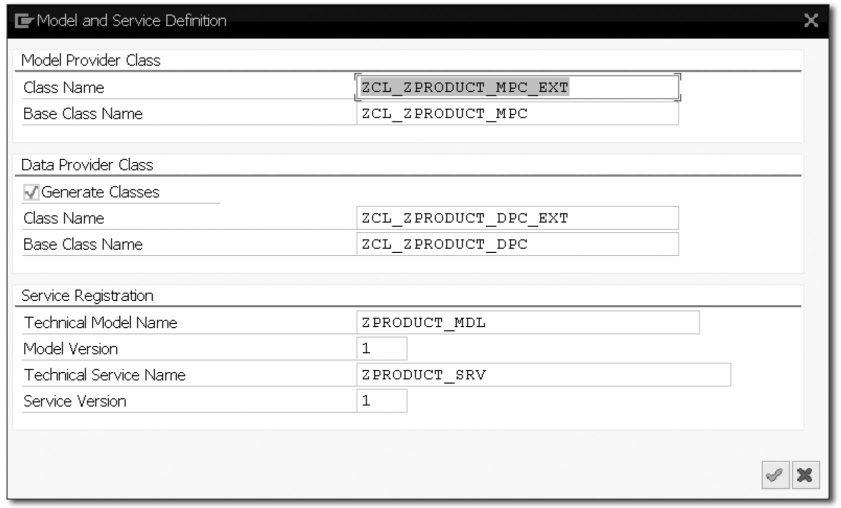
- Service registration is an activity during service development that results in the creation of artifacts needed for development.
- Service maintenance is an activity during the deployment/operation of an SAP Gateway service that activates the service for consumption.

Stub class creation

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 can 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.)

Service registration

To be used as a service, some configuration steps have to be performed, which are supported by the Service Builder (Figure 5.13).


Model and Service Definition

Model Provider Class

Class Name	ZCL_ZPRODUCT_MPC_EXT
Base Class Name	ZCL_ZPRODUCT_MPC

Data Provider Class

☑ Generate Classes

Class Name	ZCL_ZPRODUCT_DPC_EXT
Base Class Name	ZCL_ZPRODUCT_DPC

Service Registration

Technical Model Name	ZPRODUCT_MDL
Model Version	1
Technical Service Name	ZPRODUCT_SRV
Service Version	1

Figure 5.13 Model and Service Definition Using Service Builder

When generating a project for the first time, the developer has to specify the names of the MPC and its extension class and the DPC and its extension class. In addition, the developer has to 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 depicted in Figure 5.14. In addition to the MPC (covered in detail in Section 5.5.1) and the DPC (see Section 5.5.2), two additional repository objects for the model and the service are created as part of the registration process of a service in the SAP Business Suite.

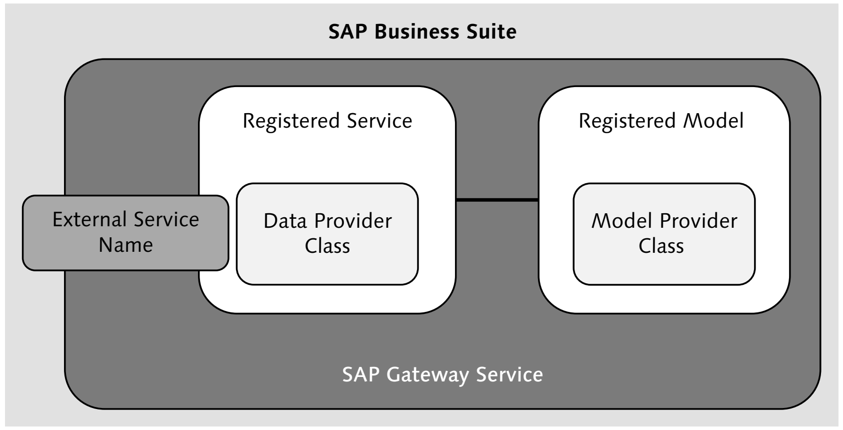


Figure 5.14 Register Service and Model

5.4.3 Service Implementation

During the service implementation phase of the service creation process, operations that are to 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 `CREATE`, `READ`, UPDATE, DELETE, and QUERY methods (CRUD-Q methods) when using RFC function modules or BAPIs, or they are limited to READ and QUERY when using Search Help or CDS views.

It's important to 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 referencing of a CDS view as a data source, the service implementation step isn't necessary because the implementation of the service will be generated based on the customizing that has been performed in the model definition step.

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 give you 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 have to 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`

Expand CRUD-Q methods

Access to these methods is offered in a very convenient way by the Service Builder. This takes place by expanding the service implementation node as depicted in Figure 5.15.

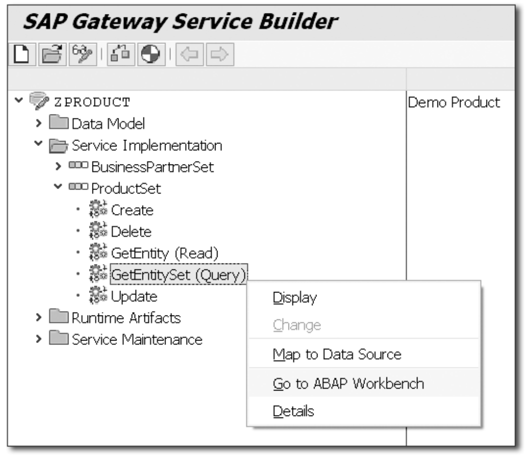


Figure 5.15 Code-Based Implementation

From there, you can navigate to the respective 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.

In addition, it might be necessary 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 (if, e.g., deep insert should be supported by the OData service).

Implementation for Mapping RFC/BOR Interfaces

The process of implementation for mapping RFC/BOR interfaces is different from that of service development. To start the mapping process, you have to select Map to Data Source in the context menu of a CRUD-Q method of an entity set in the Service Implementation folder (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.

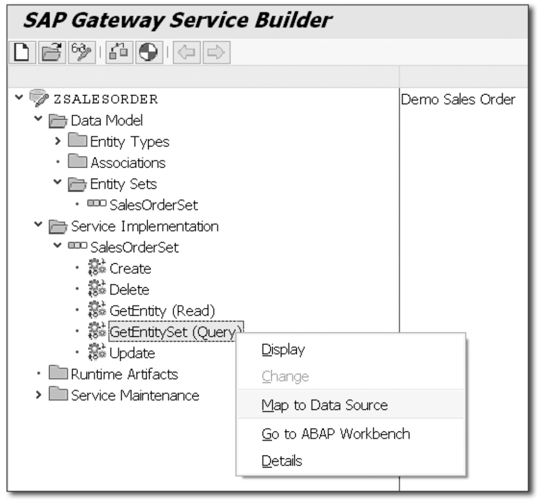
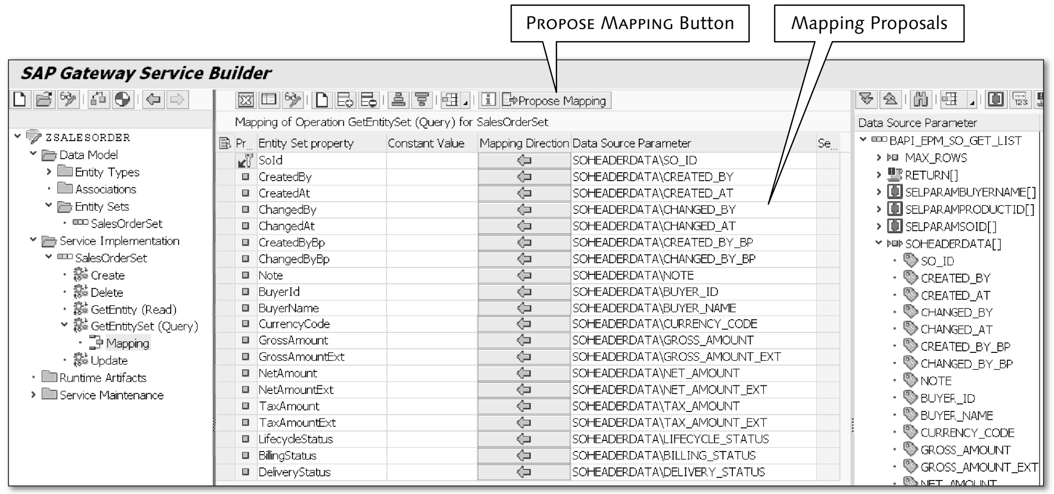


Figure 5.16 Mapping the Methods of an Entity Set to a Data Source

CRUD-Q

You can map the `CREATE`, `READ`, `UPDATE`, `DELETE`, and `QUERY` (CRUD-Q) methods of each entity set separately. The actual service implementation, that is, 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. 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.



Mapping of Operation GetEntitySet (Query) for SalesOrderSet

Pr...	Entity Set property	Constant Value	Mapping Direction	Data Source Parameter	Se...
	SoId		⇦	SOHEADERDATA\SO_ID	
	CreatedBy		⇦	SOHEADERDATA\CREATED_BY	
	CreatedAt		⇦	SOHEADERDATA\CREATED_AT	
	ChangedBy		⇦	SOHEADERDATA\CHANGED_BY	
	ChangedAt		⇦	SOHEADERDATA\CHANGED_AT	
	CreatedByBp		⇦	SOHEADERDATA\CREATED_BY_BP	
	ChangedByBp		⇦	SOHEADERDATA\CHANGED_BY_BP	
	Note		⇦	SOHEADERDATA\NOTE	
	BuyerId		⇦	SOHEADERDATA\BUYER_ID	
	BuyerName		⇦	SOHEADERDATA\BUYER_NAME	
	CurrencyCode		⇦	SOHEADERDATA\CURRENCY_CODE	
	GrossAmount		⇦	SOHEADERDATA\GROSS_AMOUNT	
	GrossAmountExt		⇦	SOHEADERDATA\GROSS_AMOUNT_EXT	
	NetAmount		⇦	SOHEADERDATA\NET_AMOUNT	
	NetAmountExt		⇦	SOHEADERDATA\NET_AMOUNT_EXT	
	TaxAmount		⇦	SOHEADERDATA\TAX_AMOUNT	
	TaxAmountExt		⇦	SOHEADERDATA\TAX_AMOUNT_EXT	
	LifecycleStatus		⇦	SOHEADERDATA\LIFECYCLE_STATUS	
	BillingStatus		⇦	SOHEADERDATA\BILLING_STATUS	
	DeliveryStatus		⇦	SOHEADERDATA\DELIVERY_STATUS	

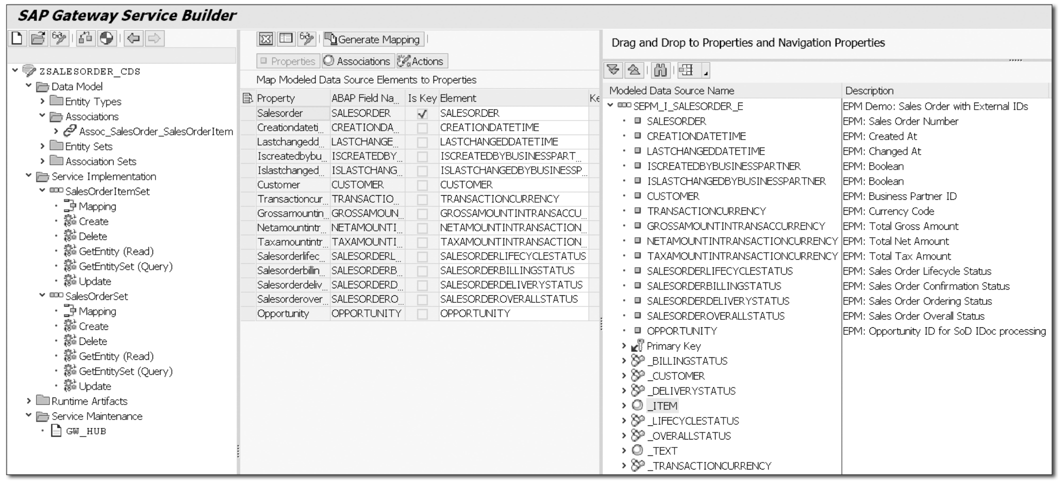
Figure 5.17 Mapping Proposals: RFC Function Module

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 Core Data Services 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 Service Implementation folder, rather than selecting the single CRUD-Q methods.

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 (see Figure 5.18), as well as mapping an association of a CDS view to a navigation property of an entity set as shown in Figure 5.19.

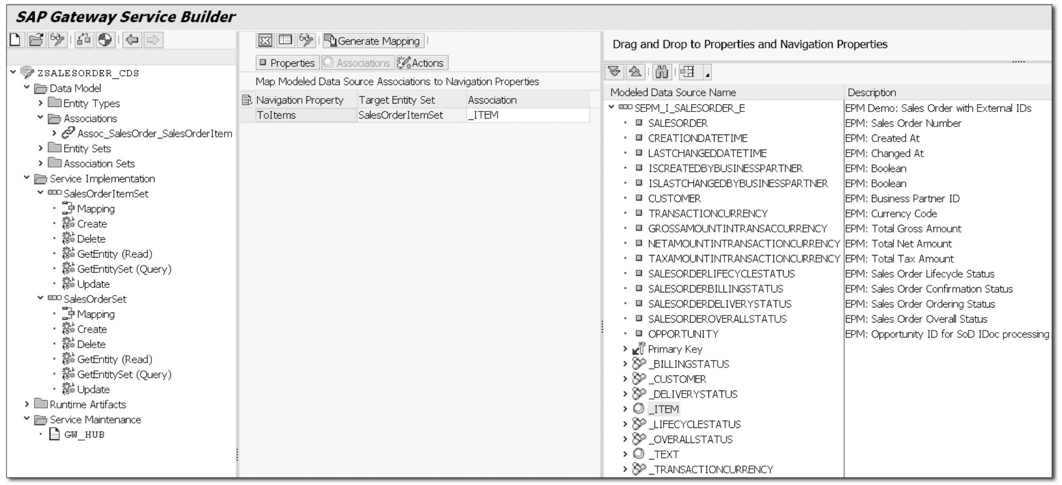


Map Modeled Data Source Elements to Properties

Property	ABAP Field Na...	Is Key	Element	Ke
Salesorder	SALESORDER	✓	SALESORDER	
Creationdateti...	CREATIONDA...		CREATIONDATETIME	
Lastchangedd...	LASTCHANGE...		LASTCHANGEDDATETIME	
Iscreatedbybu...	ISCREATEDBY...		ISCREATEDBYBUSINESSPART...	
Islastchanged...	ISLASTCHANG...		ISLASTCHANGEDBYBUSINESSP...	
Customer	CUSTOMER		CUSTOMER	
Transactioncur...	TRANSACTIO...		TRANSACTIONCURRENCY	
Grossamountin...	GROSSAMOUN...		GROSSAMOUNTINTRANSACCU...	
Netamountintr...	NETAMOUNTI...		NETAMOUNTINTRANSACTION...	
Taxamountintr...	TAXAMOUNTI...		TAXAMOUNTINTRANSACTION...	
Salesorderlifec...	SALESORDERL...		SALESORDERLIFECYCLESTATUS	
Salesorderbillin...	SALESORDERB...		SALESORDERBILLINGSTATUS	
Salesorderdeliv...	SALESORDERD...		SALESORDERDELIVERYSTATUS	
Salesorderover...	SALESORDERO...		SALESORDEROVERALLSTATUS	
Opportunity	OPPORTUNITY		OPPORTUNITY	

Modeled Data Source Name	Description
SEPM_I_SALESORDER_E	EPM Demo: Sales Order with External IDs
SALESORDER	EPM: Sales Order Number
CREATIONDATETIME	EPM: Created At
LASTCHANGEDDATETIME	EPM: Changed At
ISCREATEDBYBUSINESSPARTNER	EPM: Boolean
ISLASTCHANGEDBYBUSINESSPARTNER	EPM: Boolean
CUSTOMER	EPM: Business Partner ID
TRANSACTIONCURRENCY	EPM: Currency Code
GROSSAMOUNTINTRANSACCURRENCY	EPM: Total Gross Amount
NETAMOUNTINTRANSACTIONCURRENCY	EPM: Total Net Amount
TAXAMOUNTINTRANSACTIONCURRENCY	EPM: Total Tax Amount
SALESORDERLIFECYCLESTATUS	EPM: Sales Order Lifecycle Status
SALESORDERBILLINGSTATUS	EPM: Sales Order Confirmation Status
SALESORDERDELIVERYSTATUS	EPM: Sales Order Ordering Status
SALESORDEROVERALLSTATUS	EPM: Sales Order Overall Status
OPPORTUNITY	EPM: Opportunity ID for SoD IDoc processing
Primary Key	
_BILLINGSTATUS	
_CUSTOMER	
_DELIVERYSTATUS	
_ITEM	
_LIFECYCLESTATUS	
_OVERALLSTATUS	
_TEXT	
_TRANSACTIONCURRENCY	

Figure 5.18 Mapping a CDS View: Properties



Map Modeled Data Source Associations to Navigation Properties

Navigation Property	Target Entity Set	Association
ToItems	SalesOrderItemSet	_ITEM

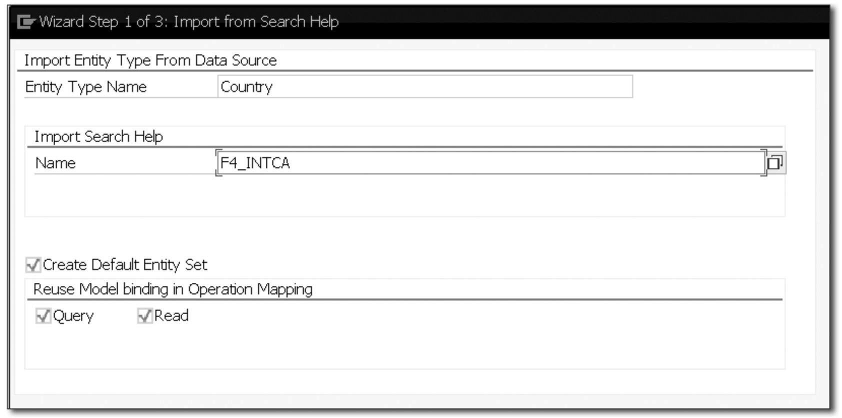
Modeled Data Source Name	Description
SEPM_I_SALESORDER_E	EPM Demo: Sales Order with External IDs
SALESORDER	EPM: Sales Order Number
CREATIONDATETIME	EPM: Created At
LASTCHANGEDDATETIME	EPM: Changed At
ISCREATEDBYBUSINESSPARTNER	EPM: Boolean
ISLASTCHANGEDBYBUSINESSPARTNER	EPM: Boolean
CUSTOMER	EPM: Business Partner ID
TRANSACTIONCURRENCY	EPM: Currency Code
GROSSAMOUNTINTRANSACCURRENCY	EPM: Total Gross Amount
NETAMOUNTINTRANSACTIONCURRENCY	EPM: Total Net Amount
TAXAMOUNTINTRANSACTIONCURRENCY	EPM: Total Tax Amount
SALESORDERLIFECYCLESTATUS	EPM: Sales Order Lifecycle Status
SALESORDERBILLINGSTATUS	EPM: Sales Order Confirmation Status
SALESORDERDELIVERYSTATUS	EPM: Sales Order Ordering Status
SALESORDEROVERALLSTATUS	EPM: Sales Order Overall Status
OPPORTUNITY	EPM: Opportunity ID for SoD IDoc processing
Primary Key	
_BILLINGSTATUS	
_CUSTOMER	
_DELIVERYSTATUS	
_ITEM	
_LIFECYCLESTATUS	
_OVERALLSTATUS	
_TEXT	
_TRANSACTIONCURRENCY	

Figure 5.19 Mapping a CDS View: Association to Navigation Property

As a result, the `READ` and `QUERY` method 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 well (see Figure 5.20).


Wizard Step 1 of 3: Import from Search Help
Import Entity Type From Data Source
Import Search Help
Reuse Model binding in Operation Mapping

Entity Type Name: Country

Name: F4_INTCA

☑ Create Default Entity Set

☑ Query ☑ Read

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 consume a service using an OData client, this service has to 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). **Activate and maintain service**

Transaction /IWFND/MAINT_SERVICE is also used to maintain all activated services on the SAP Gateway server. Services have to be changed if they've been registered in several/additional connected SAP Business Suite systems, or they can simply be deactivated.

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

The developer can either select a SAP Gateway system in the SERVICE MAINTENANCE node (Figure 5.21) or can click on the REGISTER button.

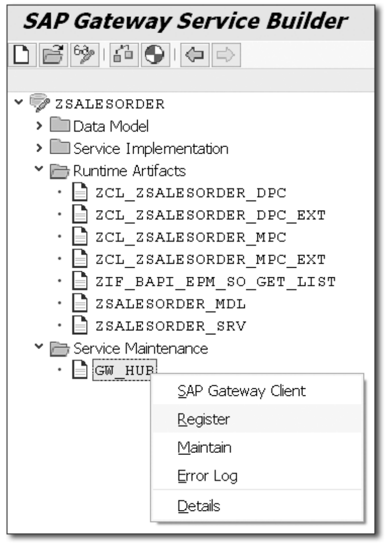


Figure 5.21 Registering a Service in the Hub from the SAP Business Suite

Service Generation

As outlined earlier in this chapter, when performing service generation via redefinition, referenced data sources, or using Eclipse to create CDS views with the `OData.publish:true` option, there is no service implementation step. There is only the data modeling phase, and the service can be published afterwards.

5.4.5 Service Generation via Redefinition

As explained 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 has to 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 allow for service creation with less effort.

Existing business objects

There are quite a number of existing business objects in an SAP system; SAP Customer Relationship Management (SAP CRM), SAP Product Lifecycle Management (SAP PLM), and SAP Enterprise Asset Management (EAM)—for example—all use a form of business object. Although these business object models have been designed for different use cases, all of them define objects, relations, actions, and queries similar to those that can be found in the OData protocol. It therefore comes as no surprise that a lot of these business objects can be used to generate OData services.

Extensibility

It's also possible to generate SAP Gateway services from existing SAP Gateway services. This scenario is used if a customer wants to extend an OData service delivered by SAP, for example, the OData service used by a SAP Fiori application. The extensibility of SAP Fiori applications is discussed end to end in Chapter 10.

Third-party OData services

On top of integrating existing SAP Business Suite business objects, it's also possible to integrate third-party OData services. However, this integration scenario has some technical restrictions.

Redefinition wizard

The wizard for generating an OData service using 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 part but automatically performs the steps that belong to the service implementation phase. After the service has been registered and implemented in the SAP Business Suite, it has to be activated in the SAP Gateway server.

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

Name of Add-On	Integration Scenario	Remote-Enabled
IW_GIL	Generic Interaction Layer (GenIL)	
IW_SPI	Service Provider Interface	X
SAP_GWFND or IW_BEP	Analytical Queries	X
SAP_GWFND or IW_BEP and IW_FND	OData service (external)	X
SAP_GWFND or IW_BEP	OData service (SAP Gateway)	X

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

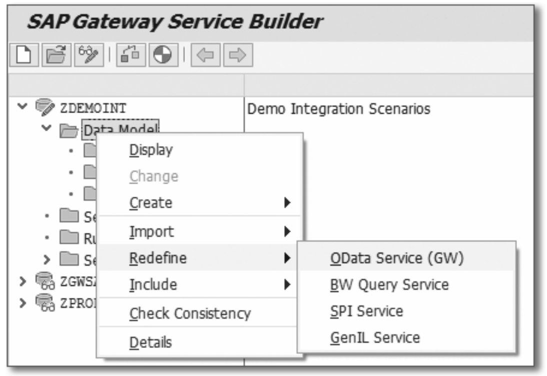


Figure 5.22 Context Menu Options to Create a Data Model Using Redefinition

Most of the scenarios are also remote-enabled, which means that the business object that is to be consumed (e.g., a Service Provider Interface

object) doesn't have 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).

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

Generic Interaction Layer

Wrapper around existing business logic

Integration of GenIL with SAP Gateway offers the possibility of generating OData services based on existing GenIL components. GenIL is meant to be 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 and thus acts as a buffer for the UI.

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

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.

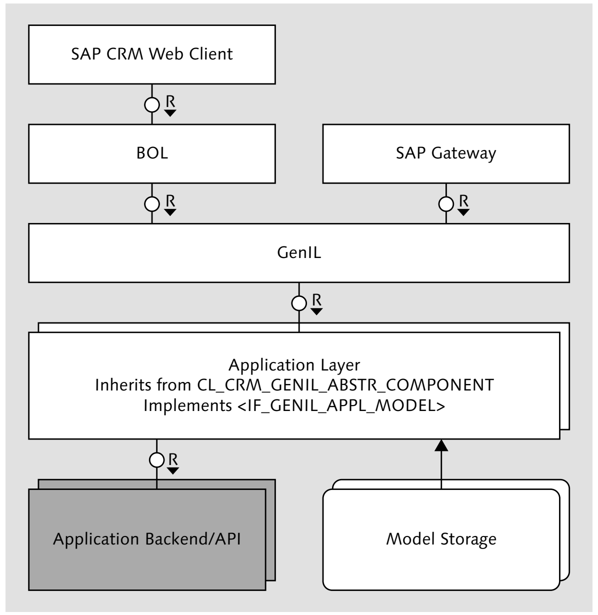


Figure 5.23 Integration of GenIL with SAP Gateway

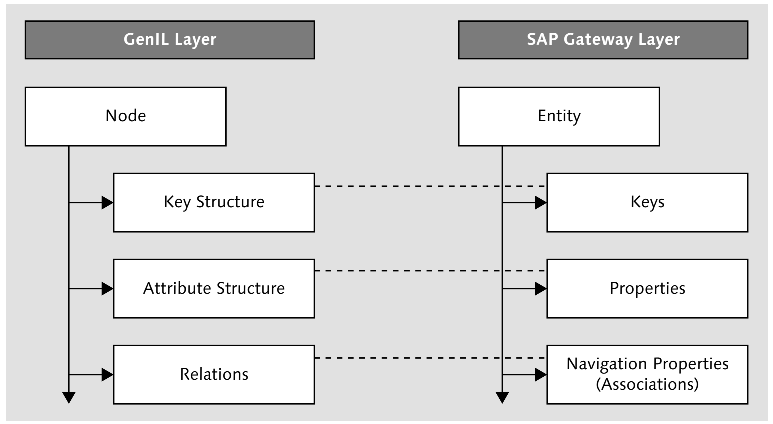


Figure 5.24 Mapping between the GenIL and OData Model

Although BOL (and thus GenIL) are frequently used for SAP CRM Web Client, 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.

This must be deployed locally on the SAP Business Suite system (e.g., SAP CRM) on top of the BEP component.

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 (SAP_GWFND starting from SAP NetWeaver release 7.40) has to be deployed on the SAP Business Suite system.

Service Provider Interface

The Service Provider Interface was originally developed for SAP Product Lifecycle Management (PLM). Service Provider Interface 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.

Service Provider Interface objects can be called remotely. As a result, it isn't mandatory to deploy the SAP Gateway IW_SPI add-on for Service Provider Interface on the SAP Business Suite system. Because the add-on calls the RFC interface of the Service Provider Interface layer, it can be deployed on the SAP Gateway server system. The IW_GIL add-on instead must be deployed locally on the SAP Business Suite system (e.g., SAP CRM). The integration of Service Provider Interface with SAP Gateway allows Service Provider Interface application building blocks to be provisioned as OData services.

Further Resources

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

- SPI wiki on SCN: *https://wiki.scn.sap.com/wiki/display/SPI*
- SAP Online Help: *http://help.sap.com/saphelp_crm70/helpdata/en/7c/0f77e9f297402aacb48ca7110c7f2a/frameset.htm*

Analytic Queries

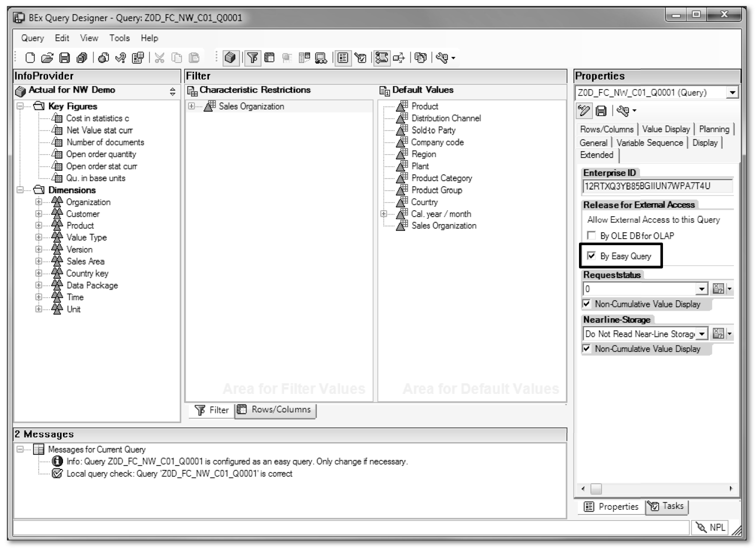
Analytic queries are the main tools for consuming analytical data that are embedded in business applications such as the SAP Business Suite and in data warehouses such as SAP BW. While analytic queries in SAP Business Suite provide access to consistent operational data, analytic queries in the SAP BW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

SAP Gateway and SAP BW integration allows you to publish SAP BW content as an OData service that has been defined using multidimensional expressions (MDX) or SAP BW Easy Queries. While the MDX approach can also be used for SAP BW systems starting with 7.0, the SAP BW Easy Query approach is only supported for release 7.30 and higher. SAP BW Easy Queries are, however, easier to understand and to handle, so they are recommended.

SAP BW Easy Queries

SAP BW Easy Queries are analytic queries that meet certain criteria. For a given SAP BW Easy Query, an RFC module is created in the system. This is done automatically by the system, based on the available SAP BW query definition. Using this RFC, an SAP BW Easy Query interface can be defined as an OData service.

To release an analytical query as an SAP BW Easy Query, you have to mark the corresponding checkbox in the query properties in the BEx Query Designer (see Figure 5.25).



Z0D_FC_NW_C01_Q0001 (Query)

Enterprise ID: 12RTXQ3YB85BGIIUN7WPA7T4U

Release for External Access
Allow External Access to this Query
☐ By OLE DB for OLAP
☑ By Easy Query

Requeststatus: 0
☑ Non-Cumulative Value Display

Nearline-Storage: Do Not Read Near-Line Storag
☑ Non-Cumulative Value Display

2 Messages
Messages for Current Query
Info: Query Z0D_FC_NW_C01_Q0001 is configured as an easy query. Only change if necessary.
Local query check: Query 'Z0D_FC_NW_C01_Q0001' is correct

Figure 5.25 Defining an SAP BW Easy Query in the BEx Query Designer

After this has been done, and the query is saved, the generation of the RFC is triggered. General rules that apply for SAP BW Easy Queries are that characteristics are on the rows, key figures are on the columns, and free characteristics aren't mapped to OData.

Analytical annotations

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

SAP BW Objects	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

External OData Service

OSCI

OData Services Consumption and Integration (OSCI) is an additional integration scenario that aims at enabling consumption and integration of any OData service. With SP 07 of SAP Gateway 2.0, this functionality is fully integrated with the Service Builder. The integration has to be implemented on the SAP Gateway server system, where the IW_BEP add-on also has to be deployed. This is required because you need the OData library for the consumption of an OData service, and this library only resides on the SAP Gateway server. In addition, you also need IW_BEP for service development on the SAP Gateway server.

As of SAP NetWeaver ABAP 7.40 SP 02, this prerequisite will be fulfilled by any SAP NetWeaver ABAP system because the SAP_GWFND software component comprises the required functionality.

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 10.

5.4.6 Service Generation via Referenced Data Sources

With the advent of SAP HANA, there was a paradigm shift in how business applications were developed at SAP. 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 reach data models by providing annotations that can be leveraged by Smart Templates. These are smart in a 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 (see Figure 5.26).

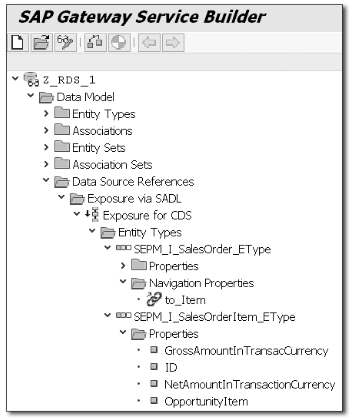


Figure 5.26 CDS View as a Referenced Data Source in Transaction SEGW

This means that any change in the underlying CDS view is automatically reflected in the OData service that has been generated using the referenced data source 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.

5.4.7 Service Generation via OData.publish:true

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

It's planned that the `@OData.publish:true` option won't only be suitable for read-only and analytical services but will also be used for transactional services by generating appropriate BOPF objects alongside the OData service. By performing a code-based implementation of those objects, the generated OData service will also support the capability to `create`, `update`, and `delete` business data.

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 closer at the *OData channel development paradigm*, which is a specific approach for service development. This introduction lays the theoretical foundation for Chapter 6, which goes into great detail about service development. The OData channel is part of the SAP Gateway basics if you plan on using 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 Business Suite can be used to develop SAP Gateway services. In addition, OData query options can be leveraged in the SAP Business Suite systems so that only data that has been requested by the client are selected from the SAP Business Suite system and sent back over the wire. This results in highly optimized services and major performance improvements due to a lower transferred data size.

Four components of an SAP Gateway service

SAP Gateway services with respect to the OData programming model 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 Business Suite system together with the *technical model name*.
- **Technical model name**
  Used to register the service in the SAP Business Suite 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. As a consequence, you have to 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 actually generated into two classes:

- **Base class**
  Technically, the base class is derived from the `/IWBEP/CL_MGW_PUSH_ABS_MODEL` superclass and has 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, there's no need for a developer to touch the MPC that has been generated by the Service Builder. The exception to that 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, the developer can redefine methods in the model provider extension class (see Figure 5.27).

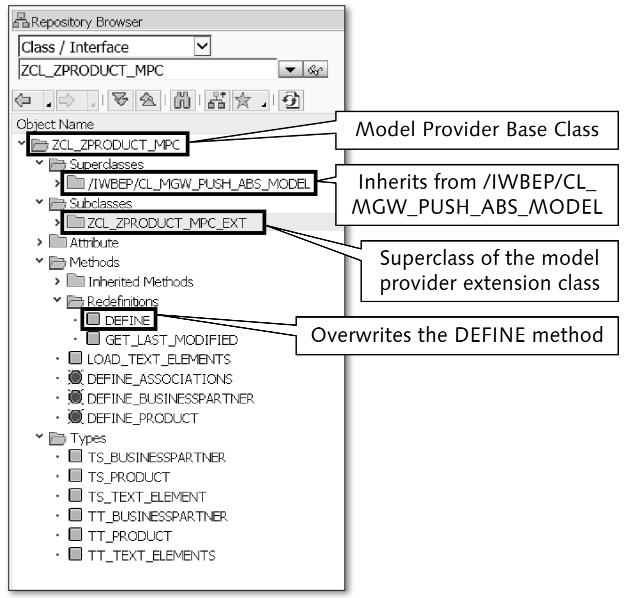


Figure 5.27 Model Provider Class

Model Provider Class Deep Dive

Usually, there's no need for a developer to tap into the coding of the MPC being generated by the Service Builder. Let's still take a closer look at the methods being generated to get a better understanding of the underlying framework.

The `DEFINE` method in the MPC generated by the Service Builder contains calls to the entity type-specific `define_<entity_type>` methods and in addition a call to the `define_Association` method that creates the associations, association sets, referential constraints, and navigation properties.

The `GET_LAST_MODIFIED` method is the basis for a handshake between the SAP Business Suite 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:

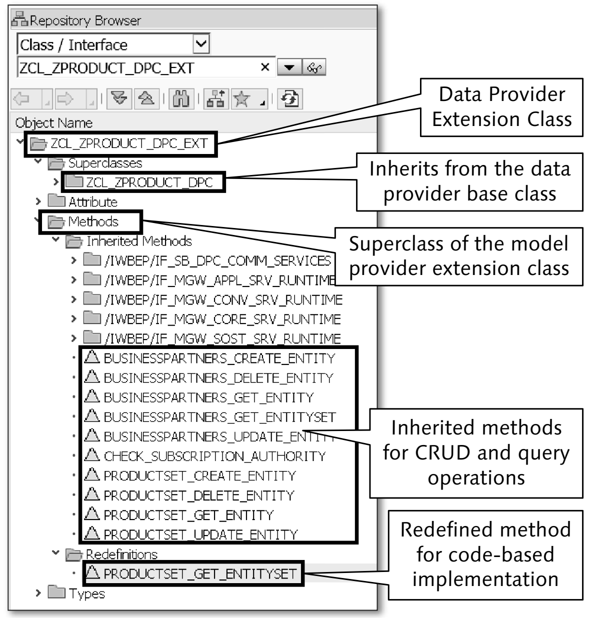
```
...
lo_entity_type->
bind_structure( iv_structure_name   =
'BAPI_EPM_PRODUCT_HEADER' iv_bind_conversions = 'X' ).
...
lo_entity_set = lo_entity_type->
create_entity_set( 'Products' )
```

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 `CREATE`, `READ`, `UPDATE`, `DELETE`, `QUERY`, and 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 (see Figure 5.28). The DPC extension class is registered via the technical service name. So the extension class is executed in your OData service.


Repository Browser
Class / Interface
Data Provider Extension Class
Inherits from the data provider base class
Superclass of the model provider extension class
Inherited methods for CRUD and query operations
Redefined method for code-based implementation

```
ZCL_ZPRODUCT_DPC_EXT
Object Name
ZCL_ZPRODUCT_DPC_EXT
  Superclasses
    ZCL_ZPRODUCT_DPC
  Attribute
  Methods
    Inherited Methods
      /IWBEP/IF_SB_DPC_COMM_SERVICES
      /IWBEP/IF_MGW_APPL_SRV_RUNTIME
      /IWBEP/IF_MGW_CONV_SRV_RUNTIME
      /IWBEP/IF_MGW_CORE_SRV_RUNTIME
      /IWBEP/IF_MGW_SOST_SRV_RUNTIME
      BUSINESSPARTNERS_CREATE_ENTITY
      BUSINESSPARTNERS_DELETE_ENTITY
      BUSINESSPARTNERS_GET_ENTITY
      BUSINESSPARTNERS_GET_ENTITYSET
      BUSINESSPARTNERS_UPDATE_ENTITY
      CHECK_SUBSCRIPTION_AUTHORITY
      PRODUCTSET_CREATE_ENTITY
      PRODUCTSET_DELETE_ENTITY
      PRODUCTSET_GET_ENTITY
      PRODUCTSET_UPDATE_ENTITY
    Redefinitions
      PRODUCTSET_GET_ENTITYSET
  Types
```

Figure 5.28 Data Provider Extension Class Interface

It's important to note that in the DPC, there are methods that are and are not specific to an entity set.

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`, or `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

There are additional methods available that apply not only for a single entity set but for all of them (nonentity set-specific methods). Examples of these methods are the methods handling `$EXPAND` statements, deep insert statements, or those that are called when a function import is performed. Let's take a closer look at these examples:

- **`GET_EXPANDED_ENTITY`, `GET_EXPANDED_ENTITYSET`**
  Handling of `$expand` statements is offered by the SAP Gateway framework out of the box in a generic way after you've modeled the appropriate navigation property and implemented the handling of navigation properties. There might be situations where you instead handle `$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.
- **`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 has to implement this method.
- **`EXECUTE_ACTION`**
  The `EXECUTE_ACTION` method is a nonentity set-specific method as well.

It's 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 the CRUD-Q methods.

5.5.3 Technical Considerations

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

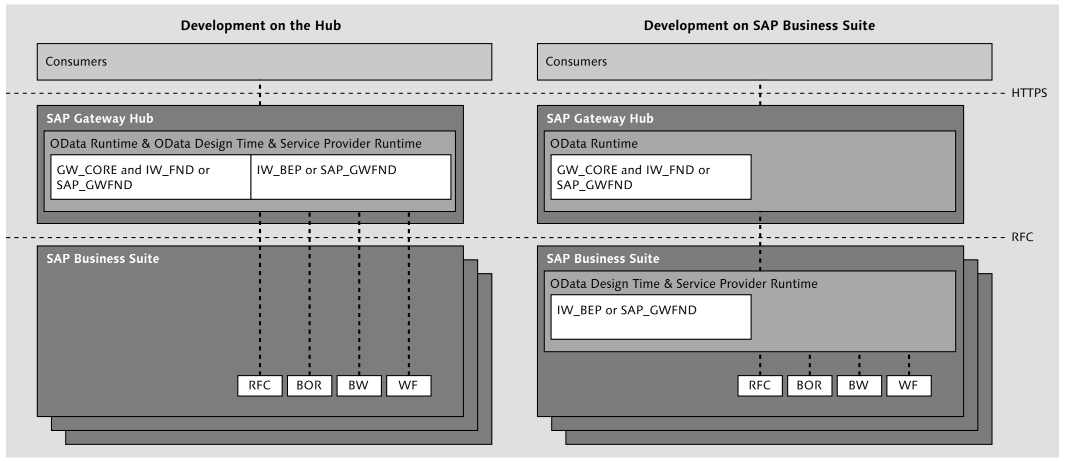

Development on the Hub
Development on SAP Business Suite
Consumers
Consumers
HTTPS
SAP Gateway Hub
OData Runtime & OData Design Time & Service Provider Runtime
GW_CORE and IW_FND or SAP_GWFND
IW_BEP or SAP_GWFND
SAP Gateway Hub
OData Runtime
GW_CORE and IW_FND or SAP_GWFND
RFC
SAP Business Suite
RFC
BOR
BW
WF
SAP Business Suite
OData Design Time & Service Provider Runtime
IW_BEP or SAP_GWFND
RFC
BOR
BW
WF

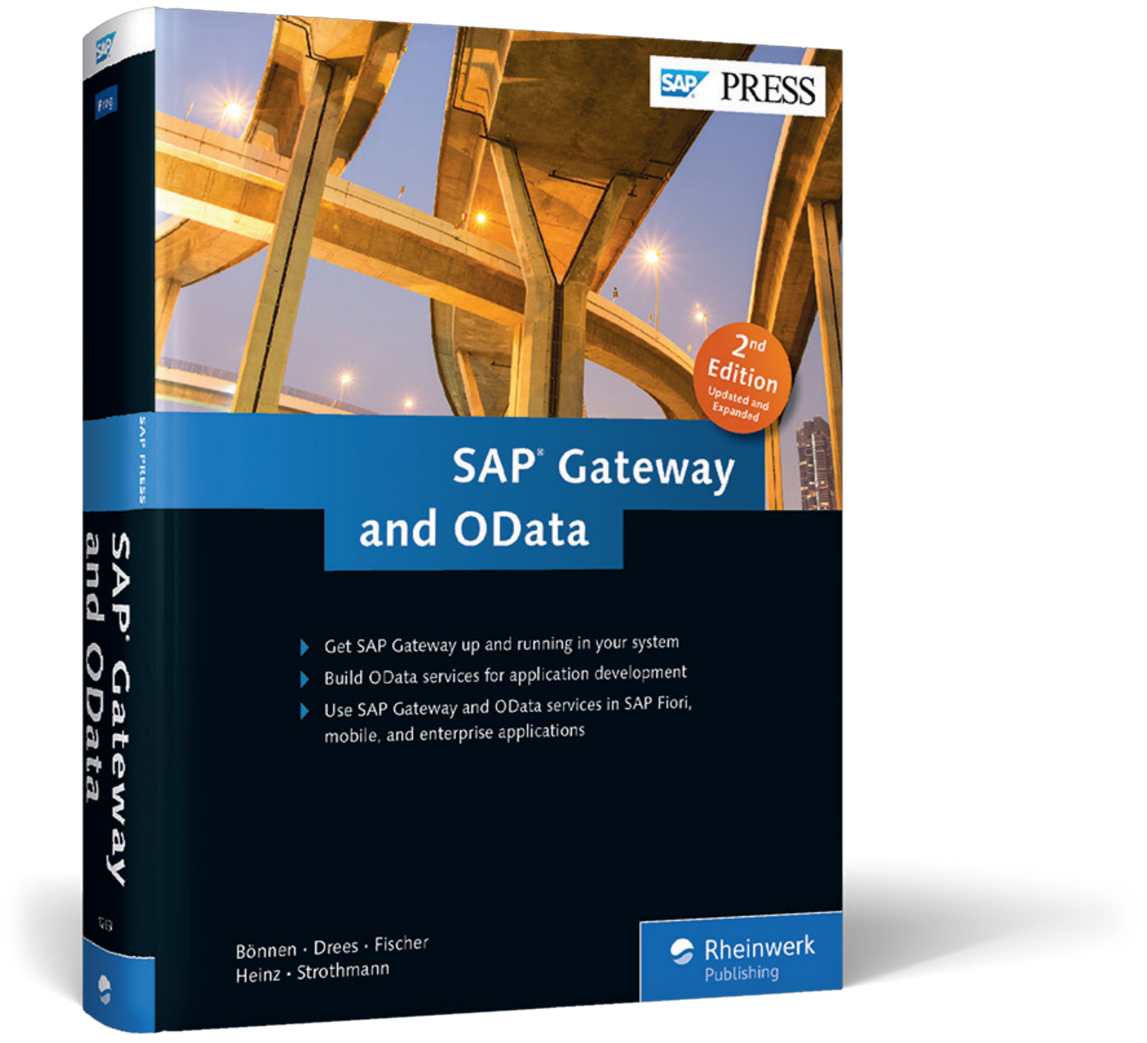
Figure 5.29 OData Channel Development on the Hub or on SAP Business Suite

5.6 Summary

Building OData services with SAP Gateway is done 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 tool and the process to establish a base of knowledge for the more technical step-by-step instructions in Chapter 6 and Chapter 7, which focus in detail on the processes of service development and service generation. In Chapter 6, you'll also be able to take advantage of the OData channel programming paradigm that you've learned about here.

Contents

Index


SAP PRESS
2nd Edition Updated and Expanded
SAP® Gateway and OData
Get SAP Gateway up and running in your system
Build OData services for application development
Use SAP Gateway and OData services in SAP Fiori, mobile, and enterprise applications
Bönnen · Drees · Fischer
Heinz · Strothmann
Rheinwerk Publishing

Carsten Bönnen, Volker Drees, André Fischer, Ludwig Heinz, Karsten Strothmann

SAP Gateway and OData

785 Pages, 2016, $79.95
ISBN 978-1-4932-1263-7

www.sap-press.com/3904

Carsten Bönnen has been a member of the SAP NetWeaver Gateway product team since 2012. He currently coordinates go-to-market activities for SAP Gateway and SAP API Management, and works on the SAP API Management product management team.

Volker Drees works as a product expert for SAP Gateway in the Products & Innovation Technology, Core Platform Gateway Division. He studied electrical engineering at Fachhochschule in Wiesbaden, Germany, and holds a degree in communications engineering (Nachrichtentechnik).

André Fischer has worked in product management for SAP Gateway since the launch of the product in 2011. Over the past 10 years at SAP, André has focused on the interoperability of SAP NetWeaver and Microsoft technologies, SAP Enterprise Search, single sign-on (SSO), and SAP Gateway.

Ludwig Heinz is the CIO of the Europe-wide acting recycling company, Theo Steil GmbH. In addition, he works as a college lecturer and supports students working on bachelor's theses that focus on mobile UI technologies.

Karsten Strothmann is the global head of SAP Gateway Customer and Product Success (CPS) at SAP SE in Walldorf, Germany. Karsten has more than 17 years of experience in the software industry, 15 of those at SAP.