SAP NetWeaver
How-To Guide

How To Write an OData Channel Gateway Service. Part 1 - The Model Provider Class.

Applicable Releases:

SAP NetWeaver 7.02 ≥SP6 + SAP NetWeaver Gateway 2.0 SP1 add-on

IT Practice / Topic Area:

SAP NetWeaver Gateway

IT Scenario / Capability:

Development of Gateway Services using the OData Channel approach

Version 1.0

August 2011

Document History

Document Version	Description
1.00	First official release of this guide

Typographic Conventions

Type Style	Description
Example Text	Words or characters quoted from the screen. These include field names, screen titles, pushbuttons labels, menu names, menu paths, and menu options. Cross-references to other documentation
Example text	Emphasized words or phrases in body text, graphic titles, and table titles
`Example text`	File and directory names and their paths, messages, names of variables and parameters, source text, and names of installation, upgrade and database tools.
`Example text`	User entry texts. These are words or characters that you enter in the system exactly as they appear in the documentation.
`<Example text>`	Variable user entry. Angle brackets indicate that you replace these words and characters with appropriate entries to make entries in the system.
`EXAMPLE TEXT`	Keys on the keyboard, for example, `F2` or `ENTER`.

Icons

Icon	Description
	Caution
	Note or Important
	Example
	Recommendation or Tip

Table of Contents

0. Introduction

This document is the first of two documents showing you how to implement a Gateway Service using the OData Channel approach (ABAP coding).

A Gateway Service is implemented by creating two ABAP classes – a Model Provider class and a Runtime Data Provider class.

In this document, we will cover the creation of the Model Provider Class. The coding in this document must be completed before you can start the coding described in the second document.

1. Business Scenario

You would like to develop some ABAP functionality for showing flights to and from various worldwide airports.

2. Background Information

Since you would like to expose this service to users outside the scope of your SAP system, this functionality needs to be made accessible via the SAP NetWeaver Gateway interface.

The SAP NetWeaver Gateway interface has been implemented using the Open Data Protocol (OData). This is a non-proprietary, license free interface based on the Atom Publishing format.

In this document, you will be shown various segments of ABAP. However, rather than simply providing you with working code for you to copy and paste from this document, you will be guided through a step by step analysis of the code in order to gain sufficient understanding to adapt what you have learnt here to your own business situation.

At each stage we will then look at the service through a browser in order to understand what XML elements are generated as a result of the ABAP coding that has just been entered.

3. Prerequisites

This document assumes the following:

- You have access to a NetWeaver 7.02 SP6 or higher system into which the SAP NetWeaver Gateway ABAP add-ons have been installed.
- You have at least a basic understanding of ABAP development.

It should be understood that an SAP NetWeaver Gateway system is not a new type of SAP system; it is merely a set of ABAP add-ons that can be applied to any SAP NetWeaver 7.02 SP6 system or higher.

The add-ons, taken together, form the Gateway system; however, the add-ons can be installed either all together on a single ABAP system, or split across two ABAP systems.

Although there are several ABAP add-ons making up a Gateway system, for the purposes of this discussion, the important ones are `GW_CORE` and `IW_BEP` because these two components determine in which system your development and configuration are performed.

a) **`GW_CORE` and `IW_BEP` in the same system**
If `GW_CORE` and `IW_BEP` are installed in the same SAP system, then this system will contain both your business data and functionality and will also be the system to which web-based clients connect in order to run the Gateway Services.

Such an installation scenario is recommended only for testing purposes, or implementations in which all the end users are within a corporate intranet.

b) **`GW_CORE` and `IW_BEP` in different systems**
If `GW_CORE` and `IW_BEP` are in different systems, then the system containing `GW_CORE` will be your Gateway server, and the system containing `IW_BEP` will be your backend business system.

This installation option allows you to isolate your backend business system behind a firewall, exposing only your Gateway Server to the public Internet.[1]

In order to perform the tasks in this tutorial, you will need to log on to whichever ABAP system contains the `IW_BEP` component with a user id having sufficient authorization to perform ABAP development and has been registered as a developer.

You will also need to log on to the ABAP system containing the `GW_CORE` add-on in order to perform a single configuration task.

[1] A trusted RFC connection must exist between the two systems. This will have been created during the post installation configuration of your Gateway system.

4. Step-by-Step Procedure

The development steps described in this document will show you how to build the first part (the Model Provider class) of the code sample used during the hands-on exercises for SAP NetWeaver Gateway at TechEd 2011.

4.1 Build the Data Model

The data model we will be using is shown on the right. It contains three Entity Types and two Complex Types.

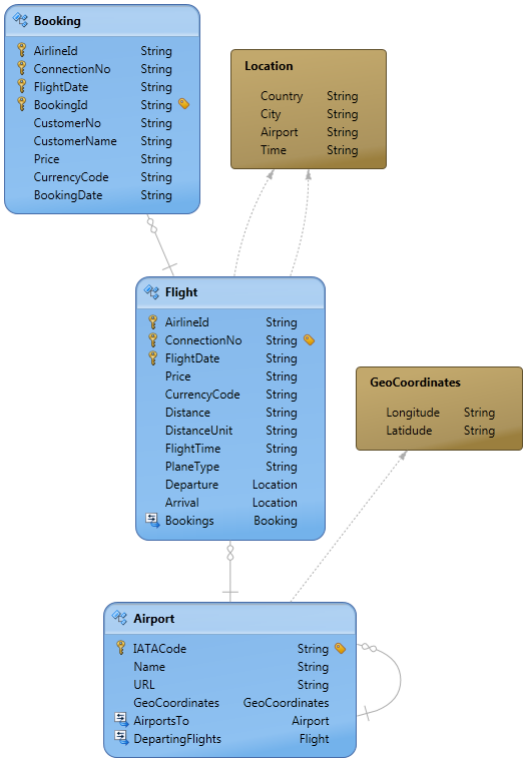
These data structures will be delivered by writing an ABAP class known as a Model Provider.

On its own, this data model would not be of much practical use because it is simply a description of various data structures and how they relate to each other.

However, OData allows us to create associations between the Entity Types to show how they relate to one another. Once an association is defined, we can create Navigation Paths.

The purpose of a Navigation Path is to allow the user to navigate from one side of an association to the other.

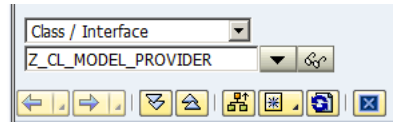
For instance, if you have selected a particular airport, then the Navigation Link will allow you see for instance, all the flights departing from that airport without the user needing to construct a specific query.



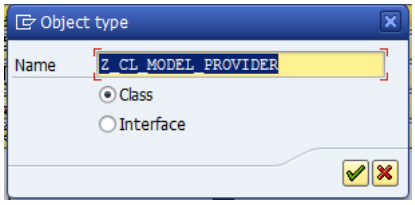
4.2 Create the Model Provider Class

Let's start with the Booking entity type because the fields in this data structure are all simple types.

1. Log on to whichever NetWeaver ABAP system has the component `IW_BEP` installed.
2. Start transaction SE80 (ABAP Development Workbench)
3. Create a new ABAP class called `Z_CL_MODEL_PROVIDER`.
   a. Enter `Z_CL_MODEL_PROVIDER` into the Class/Interface field and press enter



   b. The object is of type Class

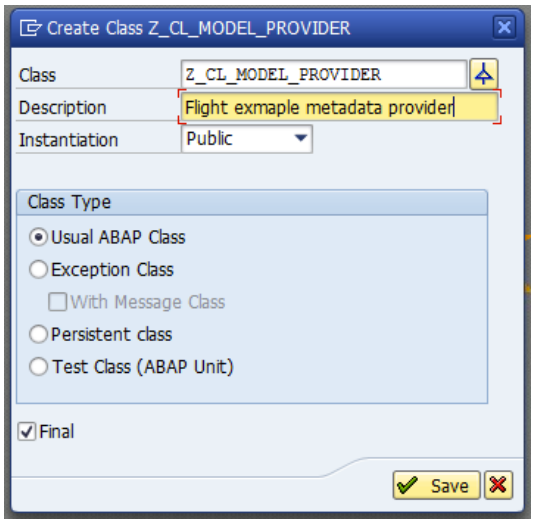


c. Give the class a meaningful description and press enter.

d. Since this is a training exercise, assign your development to Development Class `$TMP`.

e. Double click on the newly created class name to open the class editor.

f. Ensure that you are in edit mode by clicking the pencil icon on the toolbar.



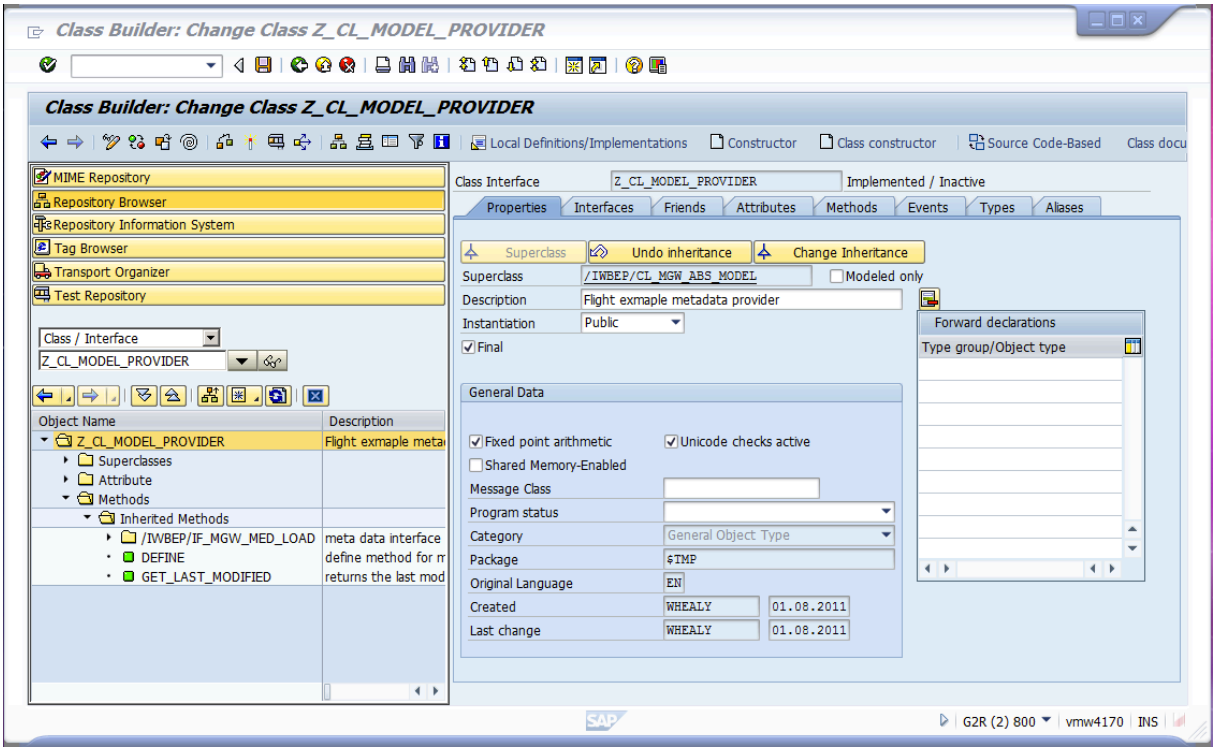
4. In order for this class to function correctly as a Model Provider class, it must inherit various methods and properties from an SAP provided abstract class called `/IWBEP/CL_MGW_ABS_MODEL`.

   (BTW, "`MGW`" in the name stands for "Minimal Gateway". This was the internal name for this design approach before the name "OData Channel" was used).

5. Select the "Properties" tab and click the Superclass button.

6. Enter `/IWBEP/CL_MGW_ABS_MODEL` and press enter and then save.

7. Once saved, the tree structure to the left shows the methods this class has inherited.



4.3 Define a Data Type

1. Before we can implement any methods, we must first define some data types within this class.

   The background to this is that all the properties used in the OData interface must have a complete metadata description (E.G. data type, length, short text etc).

   This metadata can either be supplied via the coding API when the property is defined, or it can be inherited by being associated with an appropriate data dictionary element.

   In this case, we will use the fields found within the data dictionary structure `SPFLI` to supply metadata for the OData interface properties.

2. Select the “Types” tab and enter the following:
   a. Name: `BOOKING_S`
   b. Visibility: Public
   c. Description: Booking structure
3. Click on the type definition button 
4. Replace the line `types booking_s.` with the following code:

```
types:
  begin of booking_s,
    airlineid    type spfli-carrid,
    connectionno type spfli-connid,
    flightdate   type sflight-fldate,
    bookingid    type sbook-bookid,
    customerno   type sbook-customid,
    customername type sbook-passname,
    price        type sflight-price,
    currencycode type sflight-currency,
    bookingdate  type sbook-order_date,
    title        type string,
  end of booking_s.

types:
  bookings_t type standard table of booking_s.
```

   We now have our own data types to define both a single booking (`booking_s`) and a table of such bookings (`booking_t`).

5. Save your changes.

4.4 Implementation of Method DEFINE

1. Right click on the DEFINE method and select Redefine from the context menu.
2. Copy and paste the following code. (A full explanation of this code will be given shortly)

```
method DEFINE.
  data:
    lo_entity_type  type ref to /iwbep/if_mgw_odata_entity_typ,
    lo_property     type ref to /iwbep/if_mgw_odata_property.

**********************************************************************************************
*   ENTITIES
**********************************************************************************************

  "Define Entity Booking
  lo_entity_type = model->create_entity_type( 'Booking' ).

  "Define property AirlineId and set it as a key
  lo_property = lo_entity_type->create_property( iv_property_name  = 'AirlineId'
                                                 iv_abap_fieldname = 'AIRLINEID' ).
  lo_property->set_is_key( ).

  "Define property ConnectionNo and set it as a key
  lo_property = lo_entity_type->create_property( iv_property_name  = 'ConnectionNo'
                                                 iv_abap_fieldname = 'CONNECTIONNO' ).
  lo_property->set_is_key( ).

  "Define property FlightDate and set it as a key
  lo_property = lo_entity_type->create_property( iv_property_name = 'FlightDate'
                                                 iv_abap_fieldname = 'FLIGHTDATE' ).
  lo_property->set_is_key( ).

  "Define property BookingId and set it as a key
  lo_property = lo_entity_type->create_property( iv_property_name = 'BookingId'
                                                 iv_abap_fieldname = 'BOOKINGID' ).
  lo_property->set_is_key( ).

  lo_property = lo_entity_type->create_property( iv_property_name = 'CustomerNo'
                                                 iv_abap_fieldname = 'CUSTOMERNO' ).

  "Define property CustomerName
  lo_property = lo_entity_type->create_property( iv_property_name = 'CustomerName'
                                                 iv_abap_fieldname = 'CUSTOMERNAME' ).
  "The set_as_author() method assigns the current property to the ATOM author
  "field.  The Boolean parameter defines whether the property is visible in
  "the content
  lo_property->set_as_author( iv_keep_in_content = abap_true ).

  lo_property = lo_entity_type->create_property( iv_property_name  = 'Price'
                                                 iv_abap_fieldname = 'PRICE' ).
  lo_property = lo_entity_type->create_property( iv_property_name  = 'CurrencyCode'
                                                 iv_abap_fieldname = 'CURRENCYCODE' ).
  lo_property = lo_entity_type->create_property( iv_property_name  = 'BookingDate'
                                                 iv_abap_fieldname = 'BOOKINGDATE' ).

  "Define property Title.  Assign it to the ATOM field "Title", but do not
  "keep it in the content
  lo_property = lo_entity_type->create_property( iv_property_name  = 'Title'
                                                 iv_abap_fieldname = 'TITLE' ).
  lo_property->set_as_title( iv_keep_in_content = abap_false ).

  "Use our previously defined data type to supply the metadata for the properties
  "in the OData interface.  Fields are matched by name.
  lo_entity_type->bind_structure( 'Z_CL_MODEL_PROVIDER=>BOOKING_S' ).

  "Name the EntitySet Bookings. Omitting this statement will cause an entity set
  "to be created automatically and called BookingCollection
  lo_entity_type->create_entity_set( 'Bookings' ).

endmethod.
```

3. Save and activate class `Z_CL_MODEL_PROVIDER`.
4. We should now test the code we have developed so far. In order to do this, we must configure the Gateway system to recognize that this Model Provider class belongs to a Gateway service.

   However, a Gateway Service is composed of at least two classes; a Model Provider class and one or more Runtime Data Provider classes. The next step therefore is to create a Runtime Data Provider class.

   In part 1 of this How-To guide, this class will remain empty, but its existence will allow us to see the metadata definition of the interface.

4.5 Create an Empty Runtime Data Provider Class

1. Create class `Z_CL_DATA_PROVIDER`.
2. As with class `Z_CL_MODEL_PROVIDER`, the object type is “Class”, it should be given a meaningful description and be assigned to Development Class `$TMP`.
3. The Runtime Data provider class must have class `/IWBEP/CL_MGW_ABS_DATA` as it’s superclass.
4. Save your changes and activate the class.

At the moment, this class is nothing more than an empty shell; however, it implements the standard interface required by all Gateway Runtime Data Provider classes. Therefore, it will be functional enough to allow us to see the results of our ABAP code to create the metadata definition.

4.6 Configure the Runtime Data and Model Provider Classes to Act as a Gateway Service

One thing that is very important to understand is that at the ABAP coding level, the Model Provider and Runtime Data Provider classes ***are not required to have any direct reference to each other***. This may at first seem strange because you might reasonably assume that the Runtime Data Provider class would need to create an instance of the Model Provider class in order implement its own interface.

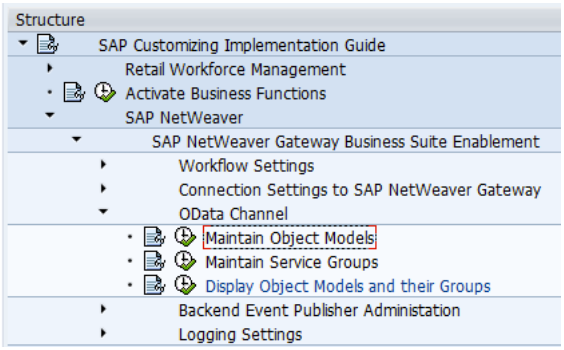
But this is not the case.

The Gateway Runtime provides a framework within which the Runtime Data Provider class receives all the necessary information without needing to directly reference the Model Provider class.

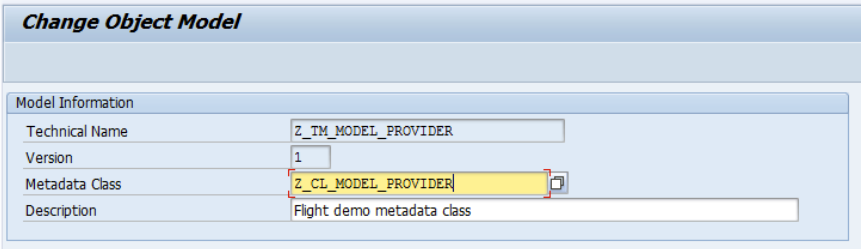
The association between the Model Provider and Runtime Data Provider classes is created using configuration, not coding.

1. The configuration that now needs to be performed is to associate the Model Provider class (`Z_CL_MODEL_PROVIDER`) with the runtime Data Provider class (`Z_CL_DATA_PROVIDER`) and is done by creating two wrapper objects.
   - The Model Provider class is wrapped in a Technical Model.
   - The Runtime Data Provider class is wrapped in a Service Group.

   The Technical Model is then assigned to the service group.
2. Start transaction `SPRO` and enter the SAP Reference IMG.
3. Follow the path SAP NetWeaver → SAP NetWeaver Gateway Business Suite Enablement → OData Channel



4. Start the “Maintain Object Models” transaction.
5. Here you will need to enter the Technical Model Name and a version number. At the moment, the version number is always 1. Enter a Technical Model Name of `Z_TM_MODEL_PROVIDER` and press “Create New Model”.
6. On the following screen, enter the name of the metadata class that is being wrapped and give it some meaningful description. In our case, the metadata class is `Z_CL_MODEL_PROVIDER`.



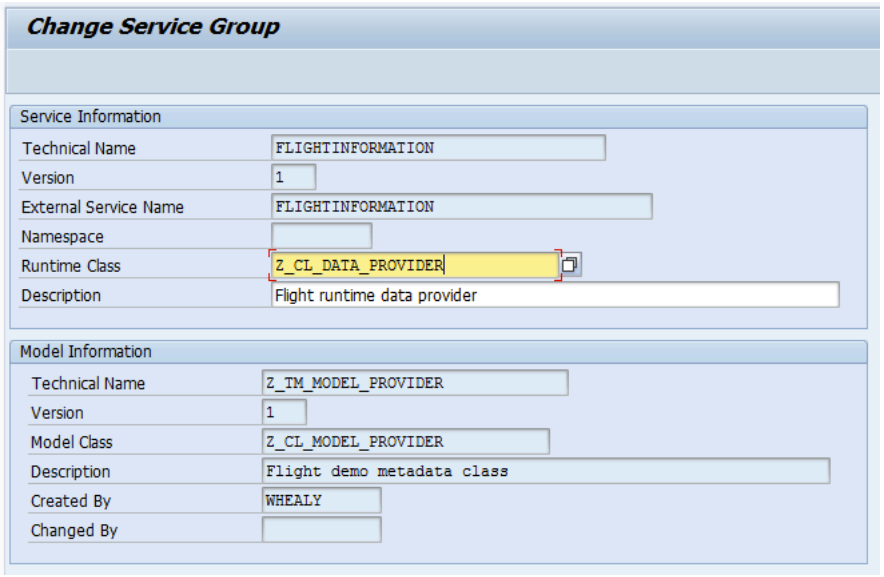
7. Save the changes and back out to the IMG tree structure.
8. Now run the “Maintain Service Groups” transaction. This is where the Runtime Data Provider class is wrapped using an Internal Name. There is a quirk in the configuration transaction that

means whatever name you enter as the Internal Service Name, is assigned to the External Service Name. Since the External Service Name is the name that will appear in the URL that runs this service, we should choose a name that will be meaningful to the end user.

Enter `FlightInformation` as the Internal Name and 1 as the version number.

Press the create button.

9. On the following screen, enter the name of the Runtime Data Provider class `Z_CL_DATA_PROVIDER` and a meaningful description. Remember that at the moment, this class is just an empty shell.
10. You must now save your configuration otherwise you will get an error message on the following pop-up screen.
11. Since we have already created a Technical Model to wrap our metadata class, we need to press the “Assign Model” button, not the “Create Model” button.
12. On the following pop-up, enter the name of the Technical Model we created above (`Z_TM_MODEL_PROVIDER`), the version number (which at the moment, is always 1) and press the create button.
13. Now press Save, and your completed configuration should look like this:

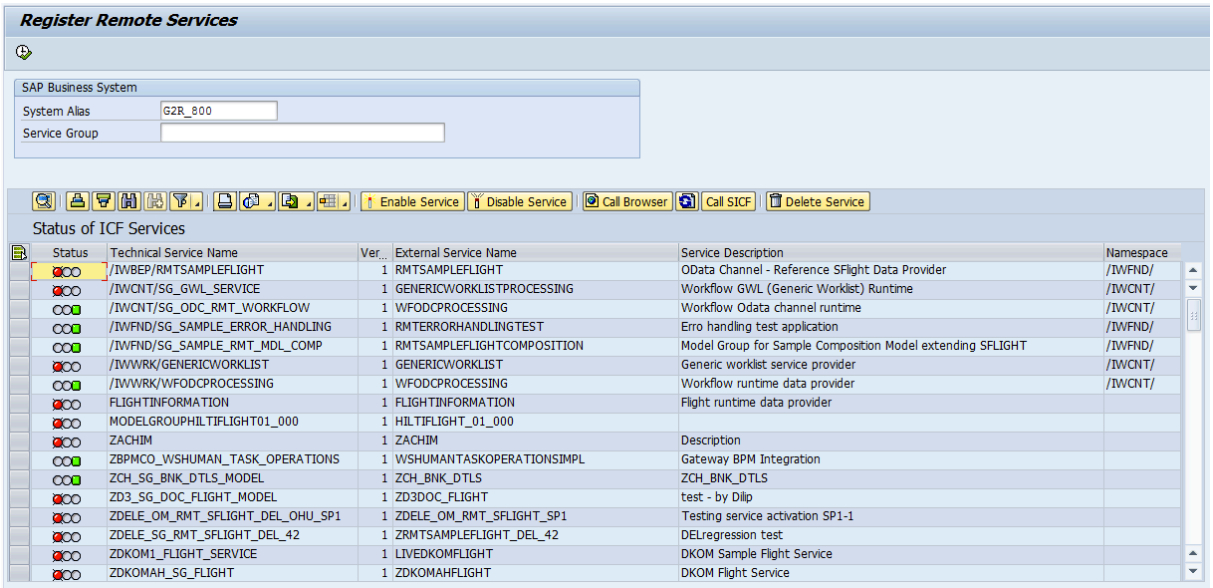


4.7 Expose the Gateway Service to the Outside World

Having created the configuration that connects the Model Provider and Runtime Data Provider classes to form a Gateway Service, it is now necessary to expose that Gateway Service to the big wide world.

1. Log on to whichever ABAP system contains the `GW_CORE` component. This may well be a different ABAP system from the one in which you have just performed the above coding and configuration.
2. Run transaction `/iwfnd/reg_service`.
3. Here, you need to know the system alias of the ABAP system in which you developed and configured the Gateway Service. There are two possibilities here:
   a. If the `GW_CORE` and `IW_BEP` components are installed on the same ABAP system, then the system alias will probably be "`LOCAL`".
   b. If they are installed on different systems, then although system aliases can be any name you like, they generally follow the naming convention of `<SID>_<Client>`. So the system alias for connecting to client 200 of system C11 would generally be `C11_200`.

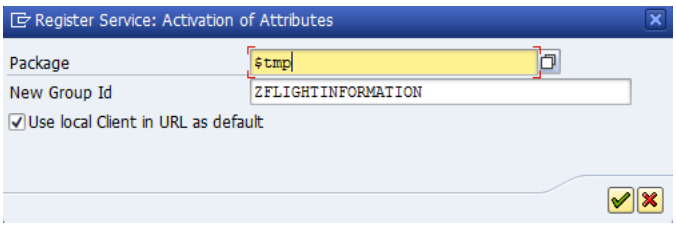
      However, this is only a convention, not a rule.
4. After entering the system alias, press enter and you will see a list of all Gateway services available from that system.



Status	Technical Service Name	Ver...	External Service Name	Service Description	Namespace
	/IWBEP/RMTSAMPLEFLIGHT	1	RMTSAMPLEFLIGHT	OData Channel - Reference SFlight Data Provider	/IWFND/
	/IWCNT/SG_GWL_SERVICE	1	GENERICWORKLISTPROCESSING	Workflow GWL (Generic Worklist) Runtime	/IWCNT/
	/IWCNT/SG_ODC_RMT_WORKFLOW	1	WFODCPROCESSING	Workflow Odata channel runtime	/IWCNT/
	/IWFND/SG_SAMPLE_ERROR_HANDLING	1	RMTERRORHANDLINGTEST	Erro handling test application	/IWFND/
	/IWFND/SG_SAMPLE_RMT_MDL_COMP	1	RMTSAMPLEFLIGHTCOMPOSITION	Model Group for Sample Composition Model extending SFLIGHT	/IWFND/
	/IWWRK/GENERICWORKLIST	1	GENERICWORKLIST	Generic worklist service provider	/IWCNT/
	/IWWRK/WFODCPROCESSING	1	WFODCPROCESSING	Workflow runtime data provider	/IWCNT/
	FLIGHTINFORMATION	1	FLIGHTINFORMATION	Flight runtime data provider	
	MODELGROUPHILTIFLIGHT01_000	1	HILTIFLIGHT_01_000		
	ZACHIM	1	ZACHIM	Description	
	ZBPMCO_WSHUMAN_TASK_OPERATIONS	1	WSHUMANTASKOPERATIONSIMPL	Gateway BPM Integration	
	ZCH_SG_BNK_DTLS_MODEL	1	ZCH_BNK_DTLS	ZCH_BNK_DTLS	
	ZD3_SG_DOC_FLIGHT_MODEL	1	ZD3DOC_FLIGHT	test - by Dilip	
	ZDELE_OM_RMT_SFLIGHT_DEL_OHU_SP1	1	ZDELE_OM_RMT_SFLIGHT_SP1	Testing service activation SP1-1	
	ZDELE_SG_RMT_SFLIGHT_DEL_42	1	ZRMTSAMPLEFLIGHT_DEL_42	DELregression test	
	ZDKOM1_FLIGHT_SERVICE	1	LIVEDKOMFLIGHT	DKOM Sample Flight Service	
	ZDKOMAH_SG_FLIGHT	1	ZDKOMAHFLIGHT	DKOM Flight Service	

5. In our example, we called our service `FLIGHTINFORMATION`. Notice that this service is visible in the list but has a red traffic light icon next to it. This means that this particular service has not yet been enabled. Select the service and press the Enable Service tool bar button.
6. A pop-up box now asks you to enter the package for this service. It is ***VERY IMPORTANT*** that you enter the correct package name here. In our case, this is `$tmp`.

   If you enter the wrong package name here, your service could become unreachable!



7. Once the service has been enabled, you will be asked if you want to open a browser and display the Service Document.

   Answer yes to this question.

8. The browser will now start and you will need to enter your logon credentials for your Gateway system.

   After your logon has been accepted, you will see your service's Service Document.

   This is a description of the collections (entity sets) provided by your Gateway Service.

4.8 Understanding the Generated OData XML

First, let's look at the address line of your browser. It contains the URL of you Gateway Service followed by a query string.

By default, the query string supplies two parameters:

```
?sap-client=100&$format=xml
```

The first is the client number in which the Gateway Service can be found (for instance `sap-client=100`) and the second instructs the server to return the OData message as plain XML.

Without the `$format=xml` parameter,[2] the Gateway server will return the OData message with a MIME type of `application/atomsvc+xml`, that is, as an Atom feed.

Modern browsers such as Firefox, Safari or Google Chrome understand this MIME type and will format the response. However, less capable browsers such as version 8 or earlier of Internet Explorer will create a pop-up dialogue asking you to select an application in order to interpret the response.

Adding the `$format=xml` parameter to the query string will change the MIME type of the server response from an Atom feed (`application/atomsvc+xml`) to plain XML (`application/xml`). This will cause the browser to display the response as indented plain text.

The XML shown below is the OData response for the Service Document. This is the highest-level description of the collections provided by our Gateway Service called `FLIGHTINFORMATION`.

```
<?xml version="1.0" encoding="utf-8" ?>
<app:service
    xml:base="http://vmw3815.wdf.sap.corp:50009/sap/opu/sdata/sap/FLIGHTINFORMATION/"
    xmlns:app="http://www.w3.org/2007/app"
    xmlns:atom="http://www.w3.org/2005/Atom"
    xmlns:sap="http://www.sap.com/Protocols/SAPData"
    xmlns:gp="http://www.sap.com/Protocols/SAPData/GenericPlayer"
    xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata">
  <app:workspace sap:semantics="things">
    <atom:title>Things</atom:title>
  </app:workspace>
  <app:workspace sap:semantics="data">
    <atom:title>Data</atom:title>
    <app:collection href="Bookings" sap:content-version="1">
      <atom:title>Bookings</atom:title>
      <sap:member-title>Booking</sap:member-title>
    </app:collection>
  </app:workspace>
</app:service>
```

[2] At the time of writing this document (Aug 2011), `$format=json` is not yet supported by Gateway

Let's look at the XML element by element:

```
<app:service
    xml:base="http://vmw3815.wdf.sap.corp:50009/sap/opu/sdata/sap/FLIGHTINFORMATION/"
    xmlns:app="http://www.w3.org/2007/app"
    xmlns:atom="http://www.w3.org/2005/Atom"
    xmlns:sap="http://www.sap.com/Protocols/SAPData"
    xmlns:gp="http://www.sap.com/Protocols/SAPData/GenericPlayer"
    xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata">
```

After the standard XML header line, the root element is an `<app:service>` element. This element has several parameters:

- `xml:base` The base URL that displayed this document
- `xmlns:app` Expect elements from the W3 namespace called `app`
- `xmlns:atom` Expect elements from the W3 namespace called `atom`
- `xmlns:sap` Expect elements from the SAP namespace called `SAPData`
- `xmlns:gp` Expect elements from the SAP namespace called `GenericPlayer`
- `xmlns:m` Expect elements from the Microsoft namespace called `metadata`

```
<app:workspace sap:semantics="things">
    <atom:title>Things</atom:title>
  </app:workspace>
```

The `<app:workspace>` element defines a list of collections delivered by the this Gateway Service. All `<app:workspace>` elements ***must*** contain an `<atom:title>` element.

The first `<app:workspace>` element is a placeholder for "thing types". Thing types are about as abstract a data type as you can get – anything can be represented as an object of type "thing".

This `<app:workspace>` element is present for semantic correctness and is generated automatically.

Fortunately, we don't have to worry about thing types here.

```
<app:workspace sap:semantics="data">
   ... snip ...
  </app:workspace>
```

The second `<app:workspace>` element is the interesting one because it contains the collection(s) supplied by our Gateway service.

```
<atom:title>Data</atom:title>
```

First, there is the mandatory `<app:title>` element. The OData XML is considered invalid if a workspace has no title.

```
<app:collection href="Bookings" sap:content-version="1">
      ...snip...
    </app:collection>
```

Now comes an interesting part – the `<app:collection>`. Here is where we see the first reference to something we coded in ABAP.

The `<app:collection>` element has two parameters:

- `href` The URL to access this particular collection.
  This URL is relative to the URL given in the `xml:base` parameter of the `<app:service>` element
- `sap:content-version` At the moment, the content version is always

```
<atom:title>Bookings</atom:title>
      <sap:member-title>Booking</sap:member-title>
```

The `<atom:title>` element is the human readable title for this particular collection.

The `<sap:member-title>` element is the human readable title for a single entry in the collection.

Well this is all very nice, but we have just written some ABAP code in which we started to define the structure of our Gateway Service's interface, and all we can see are the names of the highest level objects.

Where's the rest of it?

4.9 Viewing a Gateway Service's Metadata

Edit the URL for your Gateway Service as follows: Insert the command `$metadata` after the terminating slash of the base URL but before the question mark character that marks the start of the query string. Like this:

```
.../FLIGHTINFORMATION/$metadata?sap-client=100&$format=xml
```

Now issue this URL.

The response shows the metadata describing this Gateway Service. (The XML has been formatted for clarity).

```
<?xml version="1.0" encoding="utf-8" ?>
<edmx:Edmx Version="1.0"
    xmlns:edmx="http://schemas.microsoft.com/ado/2007/06/edmx"
    xmlns:gp="http://www.sap.com/Protocols/SAPData/GenericPlayer"
    xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata"
    xmlns:sap="http://www.sap.com/Protocols/SAPData">
  <edmx:DataServices m:DataServiceVersion="2.0">
    <Schema Namespace="FLIGHTINFORMATION" xmlns="http://schemas.microsoft.com/ado/2008/09/edm">
      <EntityType Name="Booking" sap:content-version="1">
        <Key>
          <PropertyRef Name="AirlineId" />
          <PropertyRef Name="ConnectionNo" />
          <PropertyRef Name="FlightDate" />
          <PropertyRef Name="BookingId" />
        </Key>
        <Property
            Name="AirlineId" Type="Edm.String" Nullable="false" MaxLength="3"
            sap:label="Airline" sap:filterable="false" />
        <Property
            Name="ConnectionNo" Type="Edm.String" Nullable="false" MaxLength="4"
            sap:label="Flight Number" sap:filterable="false" />
        <Property
            Name="FlightDate" Type="Edm.DateTime" Nullable="false" Precision="10"
            sap:label="Date" sap:filterable="false" />
        <Property
            Name="BookingId" Type="Edm.String" Nullable="false" MaxLength="8"
            sap:label="Booking number" sap:filterable="false" />
        <Property
            Name="CustomerNo" Type="Edm.String" MaxLength="8"
            sap:label="Customer Number" sap:filterable="false" />
        <Property
            Name="CustomerName" Type="Edm.String" MaxLength="25"
            sap:label="Passenger Name" sap:filterable="false"
            m:FC_TargetPath="SyndicationAuthorName"
            m:FC_KeepInContent="true" />
        <Property
            Name="Price" Type="Edm.Decimal" Precision="20" Scale="2"
            sap:label="Airfare" sap:filterable="false" />
        <Property
            Name="CurrencyCode" Type="Edm.String" MaxLength="5"
            sap:label="Airline Currency" sap:filterable="false" sap:semantics="currency-code" />
        <Property
            Name="BookingDate" Type="Edm.DateTime" Precision="10"
            sap:label="Booking date" sap:filterable="false" />
        <Property
            Name="Title" Type="Edm.String"
            sap:filterable="false"
            m:FC_TargetPath="SyndicationTitle" m:FC_KeepInContent="false" />
      </EntityType>
      <EntityContainer Name="FLIGHTINFORMATION" m:IsDefaultEntityContainer="true">
        <EntitySet
            Name="Bookings"
            EntityType="FLIGHTINFORMATION.Booking"
            sap:content-version="1" />
      </EntityContainer>
    </Schema>
  </edmx:DataServices>
</edmx:Edmx>
```

After the standard XML header line, there is the <edmx:Edmx> root element (EDMX stands for Entity Data Model XML). This element must be the root element of an entity data model document and must contain a version number.

```
<edmx:Edmx Version="1.0"
    xmlns:edmx="http://schemas.microsoft.com/ado/2007/06/edmx"
    xmlns:gp="http://www.sap.com/Protocols/SAPData/GenericPlayer"
    xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata"
    xmlns:sap="http://www.sap.com/Protocols/SAPData">
  ...snip...
</edmx:Edmx>
```

This element has various optional namespace parameters; in this case, two are from Microsoft and two from SAP.

```
  <edmx:DataServices m:DataServiceVersion="2.0">
    ...snip...
  </edmx:DataServices>
```

An <edmx:Edmx> element is required to contain a single <edmx:DataServices> element. The <edmx:DataServices> element then contains the metadata description of the Gateway Service (or "Data Service", to use the official OData terminology).

The <edmx:DataServices> element can contain zero or more <Schema> elements. Here the namespace of the schema is the name of Gateway service we have just created – FLIGHTINFORMATION. This namespace declaration is important because it is used to identify all the data structures ("entity types" in OData terminology) used by this Gateway Service.

```
    <Schema Namespace="FLIGHTINFORMATION" xmlns="http://schemas.microsoft.com/ado/2008/09/edm">
      ...snip...
    </Schema>
```

Now we're starting to get down to something that bears some relationship to the coding we wrote. The <EntityType> element contains the information to describe a single Booking. As has been stated earlier, the version number of these objects is (at the moment) always 1.

```
      <EntityType Name="Booking" sap:content-version="1">
        ...snip...
      </EntityType>
```

Now look back at the ABAP coding in the DEFINE method of class Z_CL_MODEL_PROVIDER.

The <EntityType> element exists in the metadata document because of the following ABAP statement:

```
  "Define Entity Booking
  lo_entity_type = model->create_entity_type( 'Booking' ).
```

We'll come back to the <Key> element in a minute. First, let's look at the <Property> elements. For each ABAP statement such as:

```
  lo_property = lo_entity_type->create_property( iv_property_name  = 'AirlineId'
                                                 iv_abap_fieldname = 'AIRLINEID' ).
```

A corresponding <Property> element exists in the OData XML.

```
        <Property
            Name="AirlineId" Type="Edm.String" Nullable="false" MaxLength="3"
            sap:label="Airline" sap:filterable="false" />
```

However, calling the ABAP method create_property() alone is not sufficient to generate the XML seen above.

Firstly, the OData field name is "AirlineId", but the associated ABAP field name is "AIRLINEID". Secondly, the <Property> tag seems magically to know the type, length and description of this field.

However, Harry Potter was not involved here,[3] instead the actual metadata is obtained when the ABAP method bind_structure() is called:

```
lo_entity_type->bind_structure( 'Z_CL_MODEL_PROVIDER=>BOOKING_S' ).
```

We created a data type called BOOKING_S and used the ABAP dictionary information from various fields in structure SPFLI to define its fields. So ultimately, the metadata you see in the OData <Property> element was derived from the ABAP dictionary structure SPFLI.

```
"Name the EntitySet Bookings. Omitting this statement will cause an entity set
"to be created automatically and called BookingCollection
lo_entity_type->create_entity_set( 'Bookings' ).
```

If you omit the above ABAP statement to create an entity set, then an entity set will be created automatically for you, but it will be named according to the entity type name followed by the word "Collection".[4]

Now let's go back to the <Key> element.

```
<Key>
  <PropertyRef Name="AirlineId" />
  <PropertyRef Name="ConnectionNo" />
  <PropertyRef Name="FlightDate" />
  <PropertyRef Name="BookingId" />
</Key>
```

After each of these properties was created, we used an extra ABAP method call to set each one as a key field.

```
"Define property AirlineId and set it as a key
lo_property = lo_entity_type->create_property( iv_property_name  = 'AirlineId'
                                               iv_abap_fieldname = 'AIRLINEID' ).
lo_property->set_is_key( ).

"Define property ConnectionNo and set it as a key
lo_property = lo_entity_type->create_property( iv_property_name  = 'ConnectionNo'
                                               iv_abap_fieldname = 'CONNECTIONNO' ).
lo_property->set_is_key( ).

"Define property FlightDate and set it as a key
lo_property = lo_entity_type->create_property( iv_property_name  = 'FlightDate'
                                               iv_abap_fieldname = 'FLIGHTDATE' ).
lo_property->set_is_key( ).

"Define property BookingId and set it as a key
lo_property = lo_entity_type->create_property( iv_property_name = 'BookingId'
                                               iv_abap_fieldname = 'BOOKINGID' ).
lo_property->set_is_key( ).
```

[3] Too busy…
[4] The terms "Collection" and "Entity Set" can be used interchangeably.

There are a few other details that need to be explained concerning the properties.

There are various standard, Atom defined elements that can be recognised by most software packages that consume Atom feeds.

The table below shows how some these Atom elements can be assigned a value from ABAP:

Atom Element	ABAP Method
SyndicationAuthorName	set_as_author()
SyndicationPublished	set_as_published()
SyndicationTitle	set_as_title()

In our coding, we want to map the CustomerName and Title properties to the appropriate Atom defined elements. This is done by calling the set_as_author() and set_as_title() methods for each property respectively.

These methods provide the mapping information from the defined <Property> to the Atom element.

```
  "Define property CustomerName
  lo_property = lo_entity_type->create_property( iv_property_name  = 'CustomerName'
                                                 iv_abap_fieldname = 'CUSTOMERNAME' ).
  lo_property->set_as_author( iv_keep_in_content = bap_true ).

  lo_property = lo_entity_type->create_property( iv_property_name  = 'Title'
                                                 iv_abap_fieldname = 'TITLE' )
  lo_property->set_as_title( iv_keep_in_content = abap_false ).
```

At the moment, the Z_CL_DATA_PROVIDER class is nothing more than an empty shell needed to make the Gateway Service executable from the browser. The missing functionality will be implemented in part 2 of this How-To Guide, and one of the first things we will create is the implementation of the GETLIST operation for each entity set.

When this functionality is created, we will see the actual business data arriving within the OData message in the form of a set of <atom:entry> elements. If this Boolean parameter is set to true, then the property will appear as an additional <atom> element within the <atom:entry> element.

For instance, calling set_as_author(iv_keep_in_content = abap_true) for the CustomerName property will cause the addition of the following XML within the <atom:entry>.

```
<atom:entry>
  <atom:author>
    <atom:name>Harry Hawk</atom:name>
  </atom:author>

  ...snip...
</atom:entry>
```

Where “Harry Hawk” is the value of the CustomerName field.

If this parameter is set to false, then the corresponding Atom define property will be omitted.

IMPORTANT

Until we implement the functionality in the Runtime Data Provider class, the above <atom:entry> XML seen above cannot be generated.

Calling these ABAP methods causes 2 extra XML parameters to appear in the `<Property>` element:

- `m:FC_TargetPath` Target field to which this property is to be mapped
- `m:FC_KeepInContent` Whether or not the property appears only in the mapped field, or in the `<atom:entry>` field as well.

```
        <Property
            Name="CustomerName" Type="Edm.String" MaxLength="25"
            sap:label="Passenger Name" sap:filterable="false"
            m:FC_TargetPath="SyndicationAuthorName" m:FC_KeepInContent="true" />
...snip...
        <Property
            Name="Title" Type="Edm.String"
            sap:filterable="false"
            m:FC_TargetPath="SyndicationTitle" m:FC_KeepInContent="false" />
```

Finally, we come to the definition of the entity set. This definition describes how a single `Booking` is aggregated into an entity set known as `Bookings`.

```
      <EntityContainer Name="FLIGHTINFORMATION" m:IsDefaultEntityContainer="true">
        <EntitySet Name="Bookings" EntityType="FLIGHTINFORMATION.Booking" sap:content-version="1" />
      </EntityContainer>
```

The `<EntityContainer>` element contains one or more `<EntitySet>` elements. This is where we see that an entity set called `Bookings` has been defined and that it is an aggregation of the entity type `FLIGHTINFORMATION.Booking`.

The syntax for identifying the entity type (`FLIGHTINFORMATION.Booking`) is possible because in the schema, we defined `FLIGHTINFORMATION` as a namespace. Therefore `FLIGHTINFORMATION.Booking` should be read as “the entity type `Booking` that belongs to the namespace `FLIGHTINFORMATION`.

4.10 Creating a Complex Type

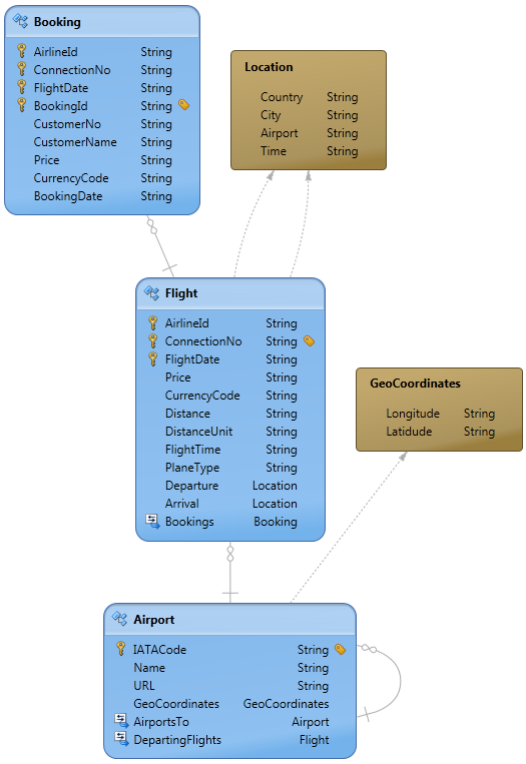
It is often valuable to be able to group a set of fields together in order to treat them as a single data type. A set of such fields is known as a "complex type".

If we look at the data model again, we can see that we have two complex types: `Location` and `GeoCoordinates`.

So the next things we need to build are the complex types. Bearing in mind that the easiest place from which to obtain our metadata is the ABAP dictionary, we need to locate some suitable data structures whose metadata can be borrowed.

In the case of the `GeoCoordinates` complex type, the `sgeocity` dictionary structure has suitable fields.

The `Location` complex type does not rely so much on the need for dictionary metadata. It does however need to be defined as a complex type to make the treatment of airport departures and arrivals easier.



1. Run transaction `se80` and enter the ABAP class name `Z_CL_MODEL_PROVIDER`.
2. Double click on the class name to display the class overview.
3. Ensure that you are in change mode and select the Types tab.
4. Enter the name `geocoordinates_s`, set the visibility to Public and press the type definition button  .
5. Replace the statement `types geocoordinates_s.` with the following type definitions.

```
  types:
    begin of geocoordinates_s,
      longitude type sgeocity-longitude,
      latidude  type sgeocity-latitude,
    end of geocoordinates_s.

  types:
    begin of location_s,
      country type string,
      city    type string,
      airport type string,
      time    type spfli-deptime,
    end of location_s .
```

Since the complex types `GeoCoordinates` and `Location` will never be entity sets in their own right, there is no need to define tables based on these types.

6. Go back to the redefinition of method `DEFINE` and double click to edit it.
7. First, you need to add an extra declaration for the local object `lo_complex_type`:

```
method DEFINE.
  data:
    lo_entity_type  type ref to /iwbep/if_mgw_odata_entity_typ,
    lo_property     type ref to /iwbep/if_mgw_odata_property,
    lo_complex_type type ref to /iwbep/if_mgw_odata_cmplx_type.
```

8. Then add the following code immediately after the variable declaration section:

```
**********************************************************************************************
*   COMPLEX TYPES
**********************************************************************************************
  "Complex type for Location
  lo_complex_type = model->create_complex_type( 'Location' ).
  lo_complex_type->create_property( iv_property_name  = 'Country'
                                    iv_abap_fieldname = 'COUNTRY' ).
  lo_complex_type->create_property( iv_property_name  = 'City'
                                    iv_abap_fieldname = 'CITY' ).
  lo_complex_type->create_property( iv_property_name  = 'Airport'
                                    iv_abap_fieldname = 'AIRPORT' ).
  lo_complex_type->create_property( iv_property_name  = 'Time'
                                    iv_abap_fieldname = 'TIME' ).
  lo_complex_type->bind_structure( 'Z_CL_MODEL_PROVIDER=>LOCATION_S' ).

  "Complex type for GeoCoordinates
  lo_complex_type = model->create_complex_type( 'GeoCoordinates' ).
  lo_complex_type->create_property( iv_property_name  = 'Longitude'
                                    iv_abap_fieldname = 'LONGITUDE' ).
  lo_complex_type->create_property( iv_property_name  = 'Latidude'
                                    iv_abap_fieldname = 'LATIDUDE' ).
  lo_complex_type->bind_structure( 'Z_CL_MODEL_PROVIDER=>GEOCOORDINATES_S' ).
```

9. Now save and activate your changes.
10. Go back to your browser and re-issue the `$metadata` URL.
11. You can now see that although the complex types for `GeoCoordinates` and `Location` have not yet been assigned to an entity, they exist in the metadata definition. (The XML has been formatted for clarity).

```
<ComplexType Name="Location">
  <Property Name="Country" Type="Edm.String" sap:filterable="false" />
  <Property Name="City"    Type="Edm.String" sap:filterable="false" />
  <Property Name="Airport" Type="Edm.String" sap:filterable="false" />
  <Property Name="Time"    Type="Edm.Time"   sap:filterable="false"
            Precision="8" sap:label="Departure" />
</ComplexType>

<ComplexType Name="GeoCoordinates">
  <Property Name="Longitude" Type="Edm.Double" sap:label="Longitude" sap:filterable="false" />
  <Property Name="Latitude"  Type="Edm.Double" sap:label="Latitude"  sap:filterable="false" />
</ComplexType>
```

4.11 Define the Remaining Entity Types

Several more entity types need to be defined. Specifically, these are the `Airport` and `Flight` entity types. As with the other entity types we have created, we will reference built-in data types within the `Z_CL_MODEL_PROVIDER` that in turn, derive their metadata from the ABAP dictionary.

1. Go back to the Types tab and enter the name `FLIGHT_S` of visibility "Public" and press the type definition button.
2. Replace the statement `types flight_s.` with the following type definitions

```
types:
  begin of flight_s,
    airlineid    type spfli-carrid,
    connectionno type spfli-connid,
    flightdate   type sflight-fldate,
    price        type sflight-price,
    currencycode type sflight-currency,
    distance     type spfli-distance,
    distanceunit type spfli-distid,
    flighttime   type spfli-fltime,
    planetype    type sflight-planetype,
    departure    type location_s,
    arrival      type location_s,
    title        type string,
  end of flight_s.

types:
  flights_t type standard table of flight_s.

types:
  begin of airport_s,
    iatacode       type char3,
    name           type char20,
    geocoordinates type geocoordinates_s,
    url            type string,
    title          type string,
  end of airport_s.

types:
  airports_t type standard table of airport_s.
```

3. Save these changes.

4. Edit method `DEFINE` and after the coding to create the `Booking` entity, add the following coding:

```
**********************************************************************
  "Define Entity Airport
  lo_entity_type = model->create_entity_type( 'Airport' ).
  lo_property = lo_entity_type->create_property( iv_property_name  = 'IATACode'
                                                 iv_abap_fieldname = 'IATACODE' ).
  lo_property->set_is_key( ).

  lo_property = lo_entity_type->create_property( iv_property_name  = 'Name'
                                                 iv_abap_fieldname = 'NAME' ).
  lo_entity_type->create_complex_property( iv_property_name     = 'GeoCoordinates'
                                           iv_complex_type_name = 'GeoCoordinates' ).
  lo_property = lo_entity_type->create_property( iv_property_name  = 'URL'
                                                 iv_abap_fieldname = 'URL' ).

  "Set property title as atom title and do not keep the property in the content
  lo_property = lo_entity_type->create_property( iv_property_name  = 'Title'
                                                 iv_abap_fieldname = 'TITLE' ).
  lo_property->set_as_title( iv_keep_in_content = abap_false ).

  lo_entity_type->bind_structure( 'Z_CL_MODEL_PROVIDER=>AIRPORT_S' ).

  "Name the EntitySet Airports instead of the default AirportCollection
  lo_entity_type->create_entity_set( 'Airports' ).

**********************************************************************
  "Define Entity Flight
  lo_entity_type = model->create_entity_type( 'Flight' ).
  lo_property = lo_entity_type->create_property( iv_property_name  = 'AirlineId'
                                                 iv_abap_fieldname = 'AIRLINEID' ).
  lo_property->set_is_key( ).
  lo_property = lo_entity_type->create_property( iv_property_name  = 'ConnectionNo'
                                                 iv_abap_fieldname = 'CONNECTIONNO' ).
  lo_property->set_is_key( ).
  lo_property = lo_entity_type->create_property( iv_property_name  = 'FlightDate'
                                                 iv_abap_fieldname = 'FLIGHTDATE' ).
  lo_property->set_is_key( ).
  lo_property = lo_entity_type->create_property( iv_property_name  = 'Price'
                                                 iv_abap_fieldname = 'PRICE' ).
  lo_property = lo_entity_type->create_property( iv_property_name  = 'CurrencyCode'
                                                 iv_abap_fieldname = 'CURRENCYCODE' ).
  lo_property = lo_entity_type->create_property( iv_property_name  = 'Distance'
                                                 iv_abap_fieldname = 'DISTANCE' ).
  lo_property = lo_entity_type->create_property( iv_property_name  = 'DistanceUnit'
                                                 iv_abap_fieldname = 'DISTANCEUNIT' ).
  lo_property = lo_entity_type->create_property( iv_property_name  = 'FlightTime'
                                                 iv_abap_fieldname = 'FLIGHTTIME' ).
  lo_property = lo_entity_type->create_property( iv_property_name  = 'PlaneType'
                                                 iv_abap_fieldname = 'PLANETYPE' ).
  lo_entity_type->create_complex_property( iv_property_name     = 'Departure'
                                           iv_complex_type_name = 'Location' ).
  lo_entity_type->create_complex_property( iv_property_name     = 'Arrival'
                                           iv_complex_type_name = 'Location' ).
  "Set property title as atom title and do not keep the property in the content
  lo_property = lo_entity_type->create_property( iv_property_name  = 'Title'
                                                 iv_abap_fieldname = 'TITLE' ).
  lo_property->set_as_title( iv_keep_in_content = abap_false ).

  lo_entity_type->bind_structure( 'Z_CL_MODEL_PROVIDER=>FLIGHT_S' ).

  "Name the EntitySet Flights instead of the default FlightCollection
  lo_entity_type->create_entity_set( 'Flights' ).
```

5. Save and activate your changes.
6. Go back to the browser and re-issue the `$metadata` URL.
7. You will now see that there are two extra `<EntityType>` elements – one for `Flight` and one for `Airport`.

4.12 Create Associations Between Entity Types

In order to create the real-life associations that exist between related pieces of information, OData provides a facility to create associations.

1. Open the ABAP editor for the redefinition of method `DEFINE`.
2. Add the following coding to the end the method.

```
*********************************************************************************
*   ASSOCIATIONS
*********************************************************************************
  "Define Association AirportFrom_AirportsTo between Airport and Airport
  lo_association = model->create_association(
                           iv_association_name = 'AirportFrom_AirportsTo'
                           iv_left_type        = 'Airport'
                           iv_right_type       = 'Airport'
                           iv_right_card       = cardinality_feed
                           iv_left_card        = cardinality_entity ).

  "Define Association AirportFrom_DepartingFlights between Airport and Flight
  lo_association = model->create_association(
                           iv_association_name = 'AirportFrom_DepartingFlights'
                           iv_left_type        = 'Airport'
                           iv_right_type       = 'Flight'
                           iv_right_card       = cardinality_feed
                           iv_left_card        = cardinality_entity ).

  "Define Association Flight_Bookings between Flight and Booking
  lo_association = model->create_association(
                           iv_association_name = 'Flight_Bookings'
                           iv_left_type        = 'Flight'
                           iv_right_type       = 'Booking'
                           iv_right_card       = cardinality_feed
                           iv_left_card        = cardinality_entity ).
```

Three associations are created here:

- An association showing which airports can be reached from any other airport.
- An association showing which flights are departing from a particular airport.
- An association showing the bookings for a particular flight.

The different sides of the association are labelled “left” and “right”. In this example, we are creating 1..n (one to many) relationships. The entity is the single item associated with the many items within the feed.

In the above examples, the left side of the association holds the entity and the right side holds the feed.

The first association relates one `AirportFrom` to many `AirportTos`. In other words, if you start at a particular airport, there are only certain other airports to which you can fly directly.

The second association relates one airport with many flights. In other words, for any given airport, there are only a certain number of flights going to or from that one airport.

Finally, the third association shows that for a given flight, there are multiple bookings.

3. Save and activate your changes.
4. Go back to the browser and re-issue the `$metadata` URL.

5. You will now see various extra elements in the XML – three `<Association>` elements, and within the `<EntityContainer>` element, three new `<AssociationSet>` elements.

```
<Association Name="Flight_Bookings" sap:content-version="1">
  <End Type="FLIGHTINFORMATION.Flight"  Multiplicity="1" Role="FromRole_Flight_Booking" />
  <End Type="FLIGHTINFORMATION.Booking" Multiplicity="*" Role="ToRole_Booking_Flight" />
</Association>

<Association Name="AirportFrom_AirportsTo" sap:content-version="1">
  <End Type="FLIGHTINFORMATION.Airport" Multiplicity="1" Role="FromRole_Airport_Airport" />
  <End Type="FLIGHTINFORMATION.Airport" Multiplicity="*" Role="ToRole_Airport_Airport" />
</Association>

<Association Name="AirportFrom_DepartingFlights" sap:content-version="1">
  <End Type="FLIGHTINFORMATION.Airport" Multiplicity="1" Role="FromRole_Airport_Flight" />
  <End Type="FLIGHTINFORMATION.Flight"  Multiplicity="*" Role="ToRole_Flight_Airport" />
</Association>

<EntityContainer Name="FLIGHTINFORMATION" m:IsDefaultEntityContainer="true">
  <EntitySet Name="Flights" EntityType="FLIGHTINFORMATION.Flight"
      sap:content-version="1" />
  <EntitySet Name="Airports" EntityType="FLIGHTINFORMATION.Airport"
      sap:content-version="1" />
   <EntitySet Name="Bookings" EntityType="FLIGHTINFORMATION.Booking"
      sap:content-version="1" />

  <AssociationSet Name="AssocSet_Flight_Bookings"
      Association="FLIGHTINFORMATION.Flight_Bookings" sap:content-version="1">
    <End EntitySet="Flights"  Role="FromRole_Flight_Booking" />
    <End EntitySet="Bookings" Role="ToRole_Booking_Flight" />
  </AssociationSet>

  <AssociationSet Name="AssocSet_AirportFrom_AirportsTo"
      Association="FLIGHTINFORMATION.AirportFrom_AirportsTo" sap:content-version="1">
    <End EntitySet="Airports" Role="FromRole_Airport_Airport" />
    <End EntitySet="Airports" Role="ToRole_Airport_Airport" />
  </AssociationSet>

  <AssociationSet Name="AssocSet_AirportFrom_DepartingFlights"
      Association="FLIGHTINFORMATION.AirportFrom_DepartingFlights" sap:content-version="1">
    <End EntitySet="Airports" Role="FromRole_Airport_Flight" />
    <End EntitySet="Flights"  Role="ToRole_Flight_Airport" />
  </AssociationSet>
</EntityContainer>
```

6. The `<Association>` element defines which entity types are related to each other, and the multiplicity of that relationship. This is the association's metadata.

```
<Association Name="Flight_Bookings" sap:content-version="1">
  <End Type="FLIGHTINFORMATION.Flight"  Multiplicity="1" Role="FromRole_Flight_Booking" />
  <End Type="FLIGHTINFORMATION.Booking" Multiplicity="*" Role="ToRole_Booking_Flight" />
</Association>
```

For instance, the association shown above defines the following:

- Assigns it the name `Flight_Bookings`
- Defines which entity types are found at either end of the association
- Defines the multiplicity of the association
- Assigns each end of the association a role name.

Then within the `<EntityContainer>` element, the association metadata is used to create a real association between the data found in the two entity sets.

Inside the `<EntityContainer>` element, the first things we find are the `<EntitySet>` elements. This is where each of the defined entity types is aggregated into an entity set.

```
<EntitySet Name="Flights"  EntityType="FLIGHTINFORMATION.Flight"
    sap:content-version="1" />
<EntitySet Name="Airports" EntityType="FLIGHTINFORMATION.Airport"
    sap:content-version="1" />
<EntitySet Name="Bookings" EntityType="FLIGHTINFORMATION.Booking"
    sap:content-version="1" />
```

Once these entity sets have been defined, it is then possible to define associations between them based on the `<Association>` metadata defined above.

```
<AssociationSet Name="AssocSet_Flight_Bookings"
    Association="FLIGHTINFORMATION.Flight_Bookings" sap:content-version="1">
  <End EntitySet="Flights"  Role="FromRole_Flight_Booking" />
  <End EntitySet="Bookings" Role="ToRole_Booking_Flight" />
</AssociationSet>
```

In this example, the data in the `Flights` entity set is associated with the data in the `Bookings` entity set. The multiplicity of this association is determined by means of the association role names. In this case, the role `FromRole_Flight_Booking` determines that a single flight is associated with multiple bookings (through the role `ToRole_Booking_Flight`).

4.13 Create Navigation Properties Between Associations

The Associations we've just created declare nothing more than the existence of an association between certain entity sets. In order to make use of these associations we need to provide the user with a means of navigating from one side of an association to the other.

This is where Navigation Properties are used.

1. Open the ABAP editor for method `DEFINE`.
2. Add the following coding to the end the method.

```
**********************************************************************
*   NAVIGATION PROPERTIES
**********************************************************************
  "Navigation Properties for entity Airport
  lo_entity_type = model->get_entity_type( iv_entity_name = 'Airport' ).

  "Add Navigation Property from role AirportFrom to role AirportsTo
  lo_entity_type->create_navigation_property(
                    iv_property_name    = 'AirportsTo'
                    iv_association_name = 'AirportFrom_AirportsTo' ).

  "Add Navigation Property from role AirportFrom to role DepartingFlights
  lo_entity_type->create_navigation_property(
                    iv_property_name    = 'DepartingFlights'
                    iv_association_name = 'AirportFrom_DepartingFlights' ).

  "Navigation Properties for entity Flight
  lo_entity_type = model->get_entity_type( iv_entity_name = 'Flight' ).

  "Add Navigation Property from role Flight to role Bookings
  lo_entity_type->create_navigation_property(
                    iv_property_name    = 'Bookings'
                    iv_association_name = 'Flight_Bookings' ).
```

3. Notice that these statements operate on the entity types for `Airport` and `Booking`, not the entity sets `Airports` and `Bookings`.

   A navigation property is identified by the value of the `iv_property_name` parameter, and then given an association on which to operate.
4. Save and activate your changes.
5. Go back to the browser and re-issue the `$metadata` URL.
6. You will now see some new `<NavigationProperty>` elements within the `<EntityType>` elements for `Flight` and `Airport`.

```
<EntityType Name="Airport" sap:content-version="1">
  ...snip...

  <NavigationProperty Name="AirportsTo"
      Relationship="FLIGHTINFORMATION.AirportFrom_AirportsTo"
      FromRole="FromRole_Airport_Airport"
      ToRole="ToRole_Airport_Airport" />

  <NavigationProperty Name="DepartingFlights"
      Relationship="FLIGHTINFORMATION.AirportFrom_DepartingFlights"
      FromRole="FromRole_Airport_Flight"
      ToRole="ToRole_Flight_Airport" />
</EntityType>

<EntityType Name="Flight" sap:content-version="1">
  ...snip...

  <NavigationProperty Name="Bookings"
      Relationship="FLIGHTINFORMATION.Flight_Bookings"
      FromRole="FromRole_Flight_Booking"
      ToRole="ToRole_Booking_Flight" />
</EntityType>
```

This completes the definition of the Gateway Service's Model Provider class `Z_CL_MODEL_PROVIDER`.

In part 2 of this guide, we will look at implementing the functionality in the Gateway Service's Runtime Data Provider class `Z_CL_DATA_PROVIDER`.

5. Appendix

5.1 A Gateway Service's Base URL

At the end of the Gateway Service's URL (known as the "base URL"), just before the start of the query string is a forward slash character. This seemingly innocuous character is important for two reasons, and if you are constructing Gateway Service URLs manually, should always be included.

1. **URL validity**
   Without the terminating slash character, the relative URL supplied in the `href` parameter of the `app:collection` element could not be appended to this base URL value without generating an invalid URL.
2. **Network Traffic Efficiency**
   Using an analysis program such as Fiddler or HTTPWatch or Inspect Element, remove the terminating slash from the base URL and issue that value directly from the address line of your browser. You'll see that the request works perfectly well…
   But now look at the request traffic in your analysis tool. ***Two*** round trips to the Gateway server took place. This is because the Gateway server detects that the URL is missing its terminating slash and issues an HTTP 307 (Temporary Redirect).
   The temporary redirect carries with it a `location` parameter that the browser then follows. The `location` parameter's value is simply the base URL to which the terminating slash has been added.

5.2 Case sensitivity of URL values

The Gateway Service name (in our case FLIGHTINFORMATION) is not case sensitive. This value can be specified in either upper or lower case. `FlightInformation`, `flightInformation` and `fLiGhTiNfOrMaTiOn` are all considered to be identical.

However, the collection names that follow the terminating slash of the Gateway Service's base URL ***are case sensitive***. They must be entered in the correct case; otherwise, you will encounter an error.

www.sdn.sap.com/irj/sdn/howtoguides