SAP NetWeaver
How-To Guide

How To Write an OData Channel Gateway Service. Part 2 - The Runtime Data Provider Class

Applicable Releases:

SAP NetWeaver 7.02 ≥SP7 + SAP NetWeaver Gateway 2.0 SP1 add-on

SAP ERP 6.0 or higher

IT Practice / Topic Area:

SAP NetWeaver Gateway

IT Scenario / Capability:

Development of Gateway Services using the OData Channel approach

Version 1.0

August 2011

THE BEST-RUN BUSINESSES RUN SAP™

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 second 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 the first document, we covered the creation of the Model Provider class.

In this second document, we will cover the creation of the Runtime Data Provider class.

Once configured to function together, these two classes form a Gateway Service.

0.1 Sample Code

Due to some problems with pasting code copied from a PDF document into the ABAP editor, each code fragment can be individually downloaded as a text file via a URL. The coding can then be copied from a text editor into the ABAP editor without corrupting the space and carriage return characters.

Alternatively, the code is available from the SAP Code Exchange.

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, the 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 coding. 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 of development 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.
- You have completed the first How To Guide in this series and have a working Model Provider class.

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, allow your NetWeaver ABAP server to function as a 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 must be a NetWeaver 7.02 SP7 or higher. This is a requirement of the `GW_CORE` component.

Such an installation will on a system that contains 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.

This installation scenario is recommended either for testing purposes only, or for situations 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.

The system containing the `IW_BEP` component can be as low a version as ERP 6.0.

This installation option allows you not only to isolate your backend business system behind a firewall, but also means that you can build a Gateway interface into much older SAP systems without the need for an upgrade.

The only system exposed to the public Internet is the Gateway Server.[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.

IMPORTANT

This is the second of two documents on the development of a Gateway Service using the OData Channel. Before working through this document, you must first have successfully completed all the coding and configuration in the first document.

[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 second part (the Runtime Data Provider class) of the code sample used during the hands-on exercises for SAP NetWeaver Gateway at TechEd 2011.

4.1 The Data Model

Upon completion of the coding and configuration in the first document in this series, you will find yourself with a complete data model for our business scenario.

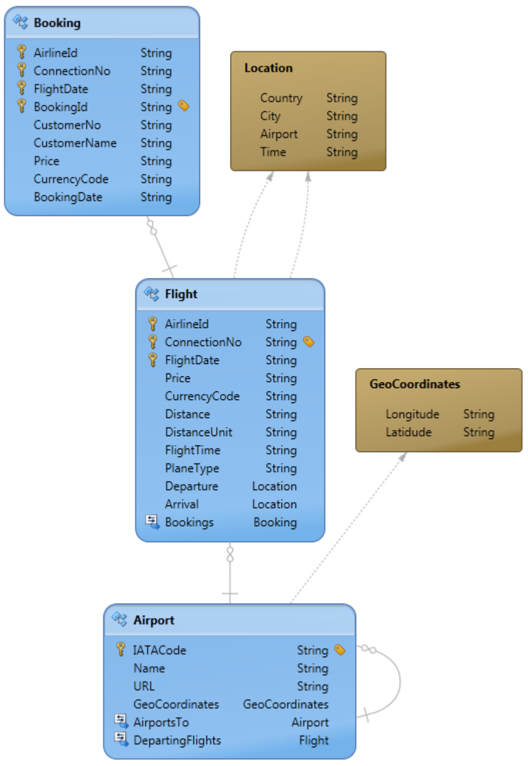
This data model has been implemented by writing an ABAP class known as a Model Provider. In our case, this is class `Z_CL_MODEL_PROVIDER`.

In order to make the Model Provider class accessible from a browser, we created a Runtime Data Provider class. In spite of the fact that this class contained no functionality whatsoever, we were still able to test the metadata structure of our Model Provider class.

Remember that from a coding perspective, there is no requirement to make a direct connection between the coding in the Model Provider class and the coding in Runtime Data Provider class.

The connection between these classes exists in the configuration, not in the coding.[2]

Now that the Model Provider class is functional, it is time to transform the Runtime Data Provider class from its current state (an empty shell), into a fully functional business object.



[2] There will be times when we reference public data types in the Model Provider from the Runtime Data Provider class. However, this is merely for convenience and is not a requirement.

4.2 Software Architecture

Before we start hammering out any code, we first need to think about the architecture of the Runtime Data Provider class `Z_CL_DATA_PROVIDER`.

The `get_entityset()` method is a good starting point for our implementation, because this method is called when in the URL, you specify a collection name without any parameters. However, `get_entityset()` is a general purpose method for delivering any number of entity sets.[3]

So how many entity sets need to be delivered in our case?

Look back at the coding in method `DEFINE` of class `Z_CL_MODEL_PROVIDER`. Here we called the `create_entity_set()` method a total of three times to aggregate the entity types `Airport`, `Flight` and `Booking` into their respective Entity Sets.

Therefore, the `get_entityset()` method in our Runtime Data Provider class will need to respond to requests for any one of these three entity sets.

Consequently, we should test what type of entity set has been requested and respond accordingly.

In addition to this, if we were to implement all the coding to handle each one of these entity sets within class `Z_CL_DATA_PROVIDER`, we would be heading down the road towards a monolithic design.[4] Therefore, we will create a separate data provider class for each type of entity set.

The data provider classes for each entity set will be called:

`Z_CL_DATA_PROVIDER_AIRPORT`

`Z_CL_DATA_PROVIDER_FLIGHT`

`Z_CL_DATA_PROVIDER_BOOKING`

And since these classes will behave exactly like `Z_CL_DATA_PROVIDER`, each will need to implement the standard Gateway interface `/IWBEP/IF_MGW_APPL_SRV_RUNTIME`. From a coding perspective, this is very convenient because the methods of these classes will now have the same signatures.

4.3 Understanding How Gateway Works with HTTP

Each time an HTTP client makes a request to a web server, that request must specify which HTTP method is being used. The commonest HTTP method is GET. This simply informs the web server that a particular resource is required. If extra parameters are required to identify that resource, then these are concatenated to the end of the URL forming what is called the “query string”.

URL refers to	HTTP Method	Operation	Method in Gateway Data Provider Class
Service Document	`GET`		Handled by Gateway Framework
Entity Set	`GET`	`QUERY`	`GET_ENTITYSET`
Entity Set + Parameters	`GET`	`READ`	`GET_ENTITY`
Entity Set	`POST + Parameters`	`CREATE`	`CREATE_ENTITY`
Entity Set + Parameters	`DELETE`	`DELETE`	`DELETE_ENTITY`
Entity Set + Parameters	PUT or POST	`UPDATE`	`UPDATE_ENTITY`
Action + Parameters	Variable	Custom	`EXECUTE_ACTION`

The `EXECUTE_ACTION` method is used to implement custom functionality and can be invoked by a variety of HTTP methods depending on whether the request is modifying or non-modifying. Typically, `GET` is used for non-modifying requests and `PUT` or `POST` for modifying requests.

[3] The terms “entity set” and “collection” can be used interchangeably.
[4] Bad news!

4.4 Create Entity Specific Data Provider Classes

1. Log on to NetWeaver ABAP system in which you previously created the Runtime Data Provider class `Z_CL_DATA_PROVIDER`. (This is the system that contains component `IW_BEP`).
2. Start transaction `se80` (ABAP Development Workbench)
3. Create a new class called `Z_CL_DATA_PROVIDER_AIRPORT`.
4. Assign the new class to the `$tmp` package.
5. This class must implement the interface `/IWBEP/IF_MGW_APPL_SRV_RUNTIME`.
6. Double click on method `/IWBEP/IF_MGW_APPL_SRV_RUNTIME~GET_ENTITYSET` and answer Yes to the question about creating an implementation.
7. Save and activate your changes.
8. Repeat steps 3 to 7 for `Z_CL_DATA_PROVIDER_FLIGHT` and `Z_CL_DATA_PROVIDER_BOOKING`.
9. You should now have three empty, data provider classes; one for each entity type `Airport`, `Flight` and `Booking`.

4.5 The Runtime Data Provider Class

Up till now, the Runtime Data Provider class has been nothing more than an empty shell. Now it is time to implement the functionality within that class.

1. In transaction `se80`, edit class `Z_CL_DATA_PROVIDER`.
2. Expand the tree `Methods` → `Inherited Methods` → `/IWBEP/IF_MGW_APPL_SRV_RUNTIME`
3. Right click on method `GET_ENTITYSET` and select Redefine from the context menu.
4. Replace the contents of the method with the following code.

```
METHOD /iwbep/if_mgw_appl_srv_runtime~get_entityset.

data:
* Reference the specific data provider class for each entity set
  lo_airport type ref to z_cl_data_provider_airport,
  lo_flight  type ref to z_cl_data_provider_flight,
  lo_booking type ref to z_cl_data_provider_booking.

* What entity set has been requested?
* Be careful! The entity set names are case-sensitive!
  case iv_entity_set_name.
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
*   AIRPORTS
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
    when 'Airports'.
      " Instantiate the data provider object for this entity set and the export
      " data structure.
      " Here, we just happen to know that the correct data type can be found in
      " the associated Model Provider class; however, such a reference is only
      " a convenience, not a requirement
      create:
        object lo_airport,
        data er_entityset type z_cl_model_provider=>airports_t.

      lo_airport->/iwbep/if_mgw_appl_srv_runtime~get_entityset(
        exporting iv_entity_name           = iv_entity_name
                  iv_source_name           = iv_source_name
                  iv_entity_set_name       = iv_entity_set_name
                  it_filter_select_options = it_filter_select_options
                  it_order                 = it_order
                  is_paging                = is_paging
                  it_navigation_path       = it_navigation_path
                  it_key_tab               = it_key_tab
                  iv_filter_string         = iv_filter_string
                  iv_search_string         = iv_search_string
        importing er_entityset             = er_entityset ).

* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
*   FLIGHTS
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
    when 'Flights'.
      " Instantiate the data provider object for this entity set and the export
      " data structure.
      create:
        object lo_flight,
        data er_entityset type z_cl_model_provider=>flights_t.

      lo_flight->/iwbep/if_mgw_appl_srv_runtime~get_entityset(
        exporting iv_entity_name           = iv_entity_name
                  iv_source_name           = iv_source_name
                  iv_entity_set_name       = iv_entity_set_name
                  it_filter_select_options = it_filter_select_options
                  it_order                 = it_order
                  is_paging                = is_paging
                  it_navigation_path       = it_navigation_path
                  it_key_tab               = it_key_tab
                  iv_filter_string         = iv_filter_string
                  iv_search_string         = iv_search_string
        importing er_entityset             = er_entityset ).
```

```
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
*   BOOKINGS
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
    when 'Bookings'.
      " Instantiate the data provider object for this entity set and the export
      " data structure.
      create:
        object lo_booking,
        data er_entityset type z_cl_model_provider=>bookings_t.

      lo_booking->/iwbep/if_mgw_appl_srv_runtime~get_entityset(
        exporting iv_entity_name          = iv_entity_name
                  iv_source_name          = iv_source_name
                  iv_entity_set_name      = iv_entity_set_name
                  it_filter_select_options = it_filter_select_options
                  it_order                = it_order
                  is_paging               = is_paging
                  it_navigation_path      = it_navigation_path
                  it_key_tab              = it_key_tab
                  iv_filter_string        = iv_filter_string
                  iv_search_string        = iv_search_string
        importing er_entityset            = er_entityset ).

* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
*   Something else...
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
    when others.
      " Add handler here
  endcase.
endmethod.
```

5. Save and activate your changes.
6. Now open your browser and display the Service Document. If you can't remember how to do this, then:
   a. Log on to the ABAP system that contains the component `GW_CORE`.
   b. Run transaction `/iwfnd/reg_service`.
   c. Enter the system alias of the ABAP system that contains component `IW_BEP` and press enter.
   d. Select the row containing the External Service name `FLIGHTINFORMATION` and press the Call Browser button in the toolbar.
   e. The browser will now open, showing you the Service Document for this Gateway Service.
7. Notice that three collections are listed: `Airports`, `Flights` and `Bookings`.[5]
8. Insert the name of any one of these collections between the base URL's terminating slash, and the question mark that denotes the start of the query string. For example:

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

   IMPORTANT

   Remember that the Collection name (`Airports`, in this case) is case-sensitive, whereas the Gateway Service name (`FLIGHTINFORMATION`) is not!
9. Press enter.

[5] The order in which these collections are listed is not important.

10. You should now see XML similar to the following:

```
<?xml version="1.0" encoding="utf-8" ?>
<atom:feed
    xml:base="http://myserver:50000/sap/opu/sdata/sap/FlightInformation/"
    xmlns:atom="http://www.w3.org/2005/Atom"
    xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices"
    xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata"
    xmlns:sap="http://www.sap.com/Protocols/SAPData">
  <atom:id>http://myserver:50000/sap/opu/sdata/sap/FlightInformation/Airports</atom:id>
  <atom:link href="Airports" rel="self" type="application/atom+xml;type=feed" title="Airports" />
  <atom:title>Airports</atom:title>
  <atom:updated>2011-08-05T17:19:02Z</atom:updated>
</atom:feed>
```

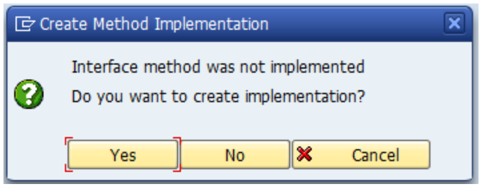
11. This is to be expected because we have implemented the `GET_ENTITYSET` method in the Runtime Data Provider class that in turn, calls the `GET_ENTITYSET` method in an entity type-specific Data Provider class.

    However, the entity type-specific class is just an empty shell, and consequently returns a null collection.

    The above XML contains the `<atom:feed>` element with the entity set header information; however, the lack of `<atom:entry>` elements tells us that this is an empty collection.

4.6 Implement the `GET_ENTITYSET` Method for the Airport Provider Class

1. In transaction `se80`, edit class `Z_CL_DATA_PROVIDER_AIRPORT`.
2. Double click on method `GET_ENTITYSET` and answer Yes to the pop-up dialogue.


Create Method Implementation

Interface method was not implemented

Do you want to create implementation?

3. Replace the method's implementation with the following code.

```
method /iwbep/if_mgw_appl_srv_runtime~get_entityset.

data:
  " Structure for one airport
  ls_airport  type z_cl_model_provider=>airport_s,
  " Table of airports
  lt_airports type standard table of spfli,

  " Single values needed in output structure
  lv_airport  type spfli-airpfrom,
  lv_country  type spfli-countryfr.

field-symbols:
  <airport>     type spfli,
  <key>         type /iwbep/s_mgw_name_value_pair,
  <fs_airports> type z_cl_model_provider=>airports_t.

* Create the output data structure and assign its fields to
* the field symbol
  create data er_entityset type z_cl_model_provider=>airports_t.
  assign er_entityset->* to <fs_airports>.

* Is there a navigation path that needs to be followed?
  if it_navigation_path is initial.
    " Nope, so in this case, we are simply obtaining a
    " list of all available airports
    select distinct airpfrom cityfrom countryfr
      into corresponding fields of table lt_airports
      from spfli
     up to is_paging-top rows.
  else.
    " Yup, so now we need to read only those airports
    " that can be reached directly from the current airport.
    " The current airport is supplied in the it_key_tab
    " table
    read table it_key_tab assigning <key> index 1.

    select distinct airpto cityto countryto
      into corresponding fields of table lt_airports
      from spfli
     up to is_paging-top rows
     where airpfrom = <key>-value.
  endif.

  " Transfer the list of airports to the output structure
  loop at lt_airports assigning <airport>.
    " Are we following a navigation path?
    if it_navigation_path is initial.
      " Nope, so use the 'from' values
      lv_airport      = <airport>-airpfrom.
      lv_country      = <airport>-countryfr.
      ls_airport-name = <airport>-cityfrom.
    else.
      " Yup, so use the 'to' values
      lv_airport      = <airport>-airpto.
      lv_country      = <airport>-countryto.
      ls_airport-name = <airport>-cityto.
    endif.

    " Populate the output structure
    ls_airport-iatacode = lv_airport.
    ls_airport-url      = z_cl_helper=>get_url_for_airport( airport = lv_airport ).

    concatenate ls_airport-name '-' lv_airport
           into ls_airport-title
     separated by space.

    ls_airport-geocoordinates-latitude =
      z_cl_helper=>get_latitude_for_airport( city    = ls_airport-name
                                             country = lv_country ).
    ls_airport-geocoordinates-longitude =
      z_cl_helper=>get_longitude_for_airport( city    = ls_airport-name
                                              country = lv_country ).

    append ls_airport to <fs_airports>.
  endloop.
endmethod.
```

4. Save your changes.
5. You may have noticed that the above coding references a class called `Z_CL_HELPER`. The purpose of this class is to make some basic geographic information retrieval reusable.
6. Create an ABAP class called `Z_CL_HELPER`.
7. Create three static, public methods having the following parameters:

Method name	Import Parameter	Type	Returning Parameter	Type
GET_LATITUDE_FOR_AIRPORT	CITY	SGEOCITY-CITY	LATITUDE	SGEOCITY-LATITUDE
	COUNTRY	SGEOCITY-COUNTRY		
GET_LONGITUDE_FOR_AIRPORT	CITY	SGEOCITY-CITY	LONGITUDE	SGEOCITY-LONGITUDE
	COUNTRY	SGEOCITY-COUNTRY		
GET_URL_FOR_AIRPORT	AIRPORT	CHAR3	URL	STRING

8. The coding for each method is as follows:

Download get_latitude_for_airport.

```
method get_latitude_for_airport.
  select single latitude
    into latitude
    from sgeocity
   where city    = city and
         country = country.
endmethod.
```

Download get_longitude_for_airport.

```
method get_longitude_for_airport.
  select single longitude
    into longitude
    from sgeocity
   where city    = city and
         country = country.
endmethod.
```

Download get_url_for_airport.

```
method get_url_for_airport.
  case airport.
    when 'BOS'. "Boston
      url ='http://www.massport.com/logan-airport/Pages/Default.aspx'.
    when 'DEN'. " Denver
      url ='http://flydenver.com/'.
    when 'EWR'. "Atlanta
      url ='http://www.panynj.gov/airports/newark-liberty.html'.
    when 'FCO'. "Rome
      url ='http://www.adr.it/fiumicino'.
    when 'JFK'. "New York
      url ='http://www.jfk-airport.net/'.
    when 'FRA'. "Frankfurt
      url = 'http://www.frankfurt-airport.com/content/frankfurt_airport/en.html'.
    when 'NRT'. "Tokyo Narita
      url ='http://www.narita-airport.jp/en/'.
    when 'SFO'. "San Francisco
      url ='http://www.flysfo.com/web/page/index.jsp'.
    when 'SIN'. "Singapore
      url = 'www.changiairport.com'.
    when 'TYO'. "Tokyo
      url ='http://www.narita-airport.jp/en/'.
  endcase.
endmethod.
```

8. Save and activate ***all*** your changes.

   If you recall, when last we attempted to list the `Airports` collection from the browser, we were attempting to see this information ***before*** we had implemented any functionality in the `GET_ENTITYSET` method of class `Z_CL_DATA_PROVIDER_AIRPORT`.

   Not surprisingly, our first attempt at listing airports resulted in an empty collection being returned to the browser. Now open your browser again and reissue the URL we tried in section 4.5.8. You should see some XML like this:

```
<?xml version="1.0" encoding="utf-8" ?>
<atom:feed
    xml:base="http://myserver:50000/sap/opu/sdata/sap/FlightInformation/"
    xmlns:atom="http://www.w3.org/2005/Atom"
    xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices"
    xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata"
    xmlns:sap="http://www.sap.com/Protocols/SAPData">
  <atom:id>http://myserver:50000/sap/opu/sdata/sap/FlightInformation/Airports</atom:id>
  <atom:link href="Airports" rel="self" type="application/atom+xml;type=feed" title="Airports" />
  <atom:title>Airports</atom:title>
  <atom:updated>2011-08-05T17:19:02Z</atom:updated>
  <atom:entry>
    <atom:author />
    <atom:content type="application/xml">
      <m:properties>
        <d:IATACode>FCO</d:IATACode>
        <d:Name>ROME</d:Name>
        <d:URL>http://www.adr.it/fiumicino</d:URL>
        <d:GeoCoordinates>
          <d:Longitude m:Type="Edm.Double">-1.2616666666666667E+01</d:Longitude>
          <d:Latitude m:Type="Edm.Double">4.1866666666666667E+01</d:Latitude>
        </d:GeoCoordinates>
      </m:properties>
    </atom:content>
    <atom:id>
      http://myserver:50000/sap/opu/sdata/sap/FLIGHTINFORMATION/Airports(IATACode='FCO')
    </atom:id>
    <atom:link href="Airports(IATACode='FCO')" rel="edit" type="application/atom+xml;type=entry" />
    <atom:link href="Airports(IATACode='FCO')/AirportsTo"
        rel="http://schemas.microsoft.com/ado/2007/08/dataservices/related/AirportsTo"
        type="application/atom+xml;type=feed"
        title="FLIGHTINFORMATION.AirportFrom_AirportsTo" />
    <atom:link href="Airports(IATACode='FCO')/DepartingFlights"
        rel="http://schemas.microsoft.com/ado/2007/08/dataservices/related/DepartingFlights"
        type="application/atom+xml;type=feed"
        title="FLIGHTINFORMATION.AirportFrom_DepartingFlights" />
    <atom:title>ROME - FCO</atom:title>
    <atom:updated>2011-08-08T11:57:16Z</atom:updated>
  </atom:entry>

  ...snip...

</atom:feed>
```

9. The important point now is that we have some `<atom:entry>` elements. Each element corresponds to a single entity type (in this case, an `Airport`).
10. The <atom:entry> element contains the following child elements:

`<atom:author>`	The Atom syndication author
`<atom:content>`	The actual information for this instance of the entity type
`<atom:id>`	The full URL to list this particular entity type (an airport in this case)
`<atom:link>`	There are two `<atom:link>` elements; one for each navigation link `AirportsTo` and `DepartingFlights`.
`<atom:title>`	The title of this particular entity.
`<atom:updated>`	The timestamp showing when this entry was generated.

11. The `<atom:content>` tag is where the information about this specific entity can be found. In the case of an `Airport`, we have (for instance):

```
<atom:content type="application/xml">
  <m:properties>
    <d:IATACode>FCO</d:IATACode>
    <d:Name>ROME</d:Name>
    <d:URL>http://www.adr.it/fiumicino</d:URL>
    <d:GeoCoordinates>
      <d:Longitude m:Type="Edm.Double">-1.2616666666666667E+01</d:Longitude>
      <d:Latitude m:Type="Edm.Double">4.1866666666666667E+01</d:Latitude>
    </d:GeoCoordinates>
  </m:properties>
</atom:content>
```

The `<m:properties>` element holds the individual fields, as either primitive or complex data types.

Here you can see that `IATACode`, `Name` and `URL` are primitive data types, but `GeoCoordinates` is a complex data type.

Now that we can get a list of airports, the next task is to implement the `GET_ENTITY` method so that we can request a single airport.

Before we can request the details of a single airport however, we must first implement the general purpose `GET_ENTITY` method in our Runtime Data Provider class that handles all entity requests.

4.7 Implement the `GET_ENTITY` Method for the Runtime Data Provider Class

1. In transaction `se80`, edit the Runtime Data Provider class `Z_CL_DATA_PROVIDER`.
2. Expand the tree `Methods` → `Inherited Methods` → `/IWBEP/IF_MGW_APPL_SRV_RUNTIME`
3. Right click on method `GET_ENTITY` and select Redefine from the context menu.
4. Replace the implementation with the following code.

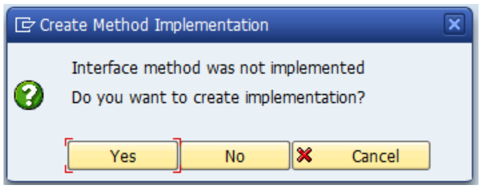
```
method /iwbep/if_mgw_appl_srv_runtime~get_entity.
data:
  lo_airport type ref to z_cl_data_provider_airport,
  lo_flight  type ref to z_cl_data_provider_flight,
  lo_booking type ref to z_cl_data_provider_booking.

* Implement call-throughs to the specific entity type data provider classes.
* What entity has been requested?  Be careful! Entity names are case sensitive!
  case iv_entity_name.
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
*   AIRPORT
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
    when 'Airport'.
      create:
        object lo_airport,
        data   er_entity type z_cl_model_provider=>airport_s.
      lo_airport->/iwbep/if_mgw_appl_srv_runtime~get_entity(
        exporting iv_entity_name     = iv_entity_name
                  iv_entity_set_name = iv_entity_set_name
                  iv_source_name     = iv_source_name
                  it_key_tab         = it_key_tab
                  it_navigation_path = it_navigation_path
        importing er_entity          = er_entity ).

* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
*   FLIGHT
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
    when 'Flight'.
      create:
        object lo_flight,
        data   er_entity type z_cl_model_provider=>flight_s.
      lo_flight->/iwbep/if_mgw_appl_srv_runtime~get_entity(
        exporting iv_entity_name     = iv_entity_name
                  iv_entity_set_name = iv_entity_set_name
                  iv_source_name     = iv_source_name
                  it_key_tab         = it_key_tab
                  it_navigation_path = it_navigation_path
        importing er_entity          = er_entity ).

* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
*  BOOKING
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
    when 'Booking'.
      create:
        object lo_booking,
        data   er_entity type z_cl_model_provider=>booking_s.
      lo_booking->/iwbep/if_mgw_appl_srv_runtime~get_entity(
        exporting iv_entity_name     = iv_entity_name
                  iv_entity_set_name = iv_entity_set_name
                  iv_source_name     = iv_source_name
                  it_key_tab         = it_key_tab
                  it_navigation_path = it_navigation_path
        importing er_entity          = er_entity ).

    when others.
      " Add some other entity handler here
  endcase.
endmethod.
```

5. Save and activate your changes.

Our Runtime Data Provider class can now respond to requests for single entities called `Airport`, `Flight` or `Booking`. We now need to implement the coding in the `GET_ENTITY` methods in each of the individual Data Provider classes.

4.8 Implement the `GET_ENTITY` Method for the Airport Provider Class

1. Edit class `Z_CL_DATA_PROVIDER_AIRPORT`.
2. Double click on the interface method `GET_ENTITY` and answer Yes to the pop-up dialogue.
3. Replace the method's implementation with the following code:


Create Method Implementation

Interface method was not implemented

Do you want to create implementation?

```
method /iwbep/if_mgw_appl_srv_runtime~get_entity.
data:
  lv_country type spfli-countryto.

field-symbols:
  <key>        type /iwbep/s_mgw_name_value_pair,
  <fs_airport> type z_cl_model_provider=>airport_s.

* Create the output data structure and assign its fields to
* the field symbol
  create data er_entity type z_cl_model_provider=>airport_s.
  assign er_entity->* to <fs_airport>.

  " In which airport are we interested?
  read table it_key_tab assigning <key> index 1.

  " Read the details of that particular airport from SPFLI
  select distinct airpfrom cityfrom countryfr
    into (<fs_airport>-iatacode, <fs_airport>-name, lv_country)
    from spfli
   where airpfrom = <key>-value.
  endselect.

  " Did we find what we're looking for?
  if sy-subrc = 0.
    " Yup
    <fs_airport>-url = z_cl_helper=>get_url_for_airport( airport = <fs_airport>-iatacode ).

    concatenate <fs_airport>-name '-' <fs_airport>-iatacode
           into <fs_airport>-title
      separated by space.

    <fs_airport>-geocoordinates-latitude =
      z_cl_helper=>get_latitude_for_airport( city    = <fs_airport>-name
                                             country = lv_country ).
    <fs_airport>-geocoordinates-longitude =
      z_cl_helper=>get_longitude_for_airport( city    = <fs_airport>-name
                                              country = lv_country ).
  endif.
endmethod.
```

4. Save and activate your changes.
5. Assuming you still have your browser open that displayed the `Airports` collection, modify the URL to insert the IATA code for an airport as follows:

   `.../FLIGHTINFORMATION/Airports(IATACode='SFO')?sap-client=100&$format=xml`

   Notice that the syntax here is not the normal query string syntax. Instead, the parameters are placed in parentheses as a comma separated list of name value pairs. The values need to be enclosed in single quote characters.

6. This asks the Gateway server to send you, not a collection of all known airports, but the details of the single airport having the IATA code of ‘SFO’ (San Francisco). After issuing this URL, you should get back XML similar to the following:

```
<?xml version="1.0" encoding="utf-8" ?>
<atom:entry xml:base="http://myserver.com:50000/sap/opu/sdata/sap/flightinformation/"
    xmlns:atom="http://www.w3.org/2005/Atom"
    xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices"
    xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata"
    xmlns:sap="http://www.sap.com/Protocols/SAPData">
  <atom:author />
  <atom:content type="application/xml">
    <m:properties>
      <d:IATACode>SFO</d:IATACode>
      <d:Name>SAN FRANCISCO</d:Name>
      <d:URL>http://www.flysfo.com/web/page/index.jsp</d:URL>
      <d:GeoCoordinates>
        <d:Longitude m:Type="Edm.Double">1.2241666666666667E+02</d:Longitude>
        <d:Latitude m:Type="Edm.Double">3.7766666666666666E+01</d:Latitude>
      </d:GeoCoordinates>
    </m:properties>
  </atom:content>
  <atom:id>
    http://myserver.com:50009/sap/opu/sdata/sap/flightinformation/Airports(IATACode='SFO')
  </atom:id>
  <atom:link href="Airports(IATACode='SFO')"
      rel="edit"
      type="application/atom+xml;type=entry" />
  <atom:link href="Airports(IATACode='SFO')/AirportsTo"
      rel="http://schemas.microsoft.com/ado/2007/08/dataservices/related/AirportsTo"
      type="application/atom+xml;type=feed"
      title="flightinformation.AirportFrom_AirportsTo" />
  <atom:link href="Airports(IATACode='SFO')/DepartingFlights"
      rel="http://schemas.microsoft.com/ado/2007/08/dataservices/related/DepartingFlights"
      type="application/atom+xml;type=feed"
      title="flightinformation.AirportFrom_DepartingFlights" />
  <atom:title>SAN FRANCISCO - SFO</atom:title>
  <atom:updated>2011-08-08T16:04:49Z</atom:updated>
</atom:entry>
```

Notice what XML element is missing here.

There is no `<atom:feed>` element acting as the root element. There is simply the `<atom:entry>` element. This is because we have constructed a request that calls the `GET_ENTITY` method, not the `GET_ENTITYSET` method. Consequently, it can only return a single entity, and thus there can be no Atom feed, only a single Atom entry.

4.9 Implement Methods `GET_ENTITYSET` & `GET_ENTITY` for the Flight Provider Classes

Since we have already created the `GET_ENTITYSET` and `GET_ENTITY` methods for the Runtime Data Provider class `Z_CL_DATA_PROVIDER`, the only coding that remains to be implemented is the entity-specific coding in the classes `Z_CL_DATA_PROVIDER_FLIGHT` and `Z_CL_DATA_PROVIDER_BOOKING`.

By now, you should be getting the idea of how a Gateway Service is implemented. So when we implement the `GET_ENTITYSET` and `GET_ENTITY` methods for the Flight and Booking Entities, you will only be provided with the coding.

1. In class `Z_CL_DATA_PROVIDER_FLIGHT` create a private data type called `FLIGHT_S` and give it the following definition:

```
types:
 begin of flight_s.
   include type spfli.

   types:
     fldate    type sflight-fldate,
     price     type sflight-price,
     currency  type sflight-currency,
     planetype type sflight-planetype,
end of flight_s .
```

2. Edit the method `GET_ENTITYSET`.
3. Replace the implementation with the following code.

```
method /iwbep/if_mgw_appl_srv_runtime~get_entityset.
data:
  lt_flights       type standard table of flight_s,
  ls_flight        type z_cl_model_provider=>flight_s,
  ls_filter_airpto type /iwbep/s_mgw_select_option,
  ls_filter_fldate type /iwbep/s_mgw_select_option.

field-symbols:
  <flight>     type flight_s,
  <key>        type /iwbep/s_mgw_name_value_pair,
  <so_airpto>  type /iwbep/s_mgw_select_option,
  <so_fldate>  type /iwbep/s_mgw_select_option,
  <fs_flights> type z_cl_model_provider=>flights_t.

  create data er_entityset type z_cl_model_provider=>flights_t.

  assign:
    ls_filter_airpto to <so_airpto>,
    ls_filter_fldate to <so_fldate>,
    er_entityset->*  to <fs_flights>.

  " Are we following a navigation path?
  if it_navigation_path is initial.
    " Nope, so read all the available flights
    select *
      into corresponding fields of table lt_flights
     up to is_paging-top rows
      from ( spfli as c
             inner join sflight as f on f~carrid = c~carrid and
                                        f~connid = c~connid ).
  else.
    " Yup, so apply key field and select options
    read:
      table it_key_tab              assigning <key>       index 1,
      table it_filter_select_options assigning <so_airpto> with key property = 'AirportTo',
      table it_filter_select_options assigning <so_fldate> with key property = 'FlightDate'.

    select *
      into corresponding fields of table lt_flights
     up to is_paging-top rows
      from ( spfli as c
              inner join sflight as f on f~carrid = c~carrid and
                                         f~connid = c~connid )
        where c~airpfrom = <key>-value and
              c~airpto in <so_airpto>-select_options and
              f~fldate in <so_fldate>-select_options.
  endif.

  " Transfer all the flight information from the SPFLI structured table,
  " to the output structured table.
  loop at lt_flights assigning <flight>.
    clear ls_flight.

    ls_flight-airlineid          = <flight>-carrid.
    ls_flight-connectionno       = <flight>-connid.
    ls_flight-flightdate         = <flight>-fldate.
    ls_flight-price              = <flight>-price.
    ls_flight-currencycode       = <flight>-currency.
    ls_flight-distance           = <flight>-distance.
    ls_flight-distanceunit       = <flight>-distid.
    ls_flight-flighttime         = <flight>-fltime.
    ls_flight-planetype          = <flight>-planetype.
    ls_flight-departure-airport = <flight>-airpfrom.
    ls_flight-departure-city     = <flight>-cityfrom.
    ls_flight-departure-country = <flight>-countryfr.
    ls_flight-departure-time     = <flight>-deptime.
    ls_flight-arrival-airport    = <flight>-airpto.
    ls_flight-arrival-city       = <flight>-cityto.
    ls_flight-arrival-country    = <flight>-countryto.
    ls_flight-arrival-time       = <flight>-arrtime.

    concatenate `Flight ` ls_flight-airlineid ls_flight-connectionno
                ` on ` ls_flight-flightdate
           into ls_flight-title.

    append ls_flight to <fs_flights>.
  endloop.
endmethod.
```

4. Save and activate your changes.
5. Test the `Flights` collection from your browser.
6. Edit method `GET_ENTITY`.
7. Replace the implementation with the following code:

```
method /iwbep/if_mgw_appl_srv_runtime~get_entity.
data:
  ls_flight type flight_s,
  lv_carrid type spfli-carrid,
  lv_connid type spfli-connid,
  lv_fldate type sflight-fldate,
  lv_kline  type /iwbep/s_mgw_name_value_pair,
  lv_lines  type i.

field-symbols:
  <fs_flight> type z_cl_model_provider=>flight_s.

  create data er_entity type z_cl_model_provider=>flight_s.
  assign er_entity->* to <fs_flight>.

  " Is the information for a specific flight required?
  if it_key_tab is not initial.
    describe table it_key_tab lines lv_lines.
    " In order for a flight to be uniquely identified, all
    " three key fields must be supplied.
    if lv_lines eq 3.
      read table it_key_tab into lv_kline with key name = 'AirlineId'.
      if sy-subrc = 0.
        lv_carrid = lv_kline-value.
      endif.

      read table it_key_tab into lv_kline with key name = 'ConnectionNo'.
      if sy-subrc = 0.
```

```
        lv_connid = lv_kline-value.
      endif.

      read table it_key_tab into lv_kline with key name = 'FlightDate'.
      if sy-subrc = 0.
        lv_fldate = lv_kline-value.
      endif.
    endif.
  endif.

  " Try to find the required flight
  select single *
    into corresponding fields of ls_flight
    from ( spfli as c
           inner join sflight as f on f~carrid = c~carrid and
                                      f~connid = c~connid )
   where f~carrid = lv_carrid and
         f~connid = lv_connid and
         f~fldate = lv_fldate.

  " Did we find it?
  if sy-subrc = 0.
    " Yup, transfer fields to output structure
    <fs_flight>-airlineid         = ls_flight-carrid.
    <fs_flight>-connectionno      = ls_flight-connid.
    <fs_flight>-flightdate        = ls_flight-fldate.
    <fs_flight>-price             = ls_flight-price.
    <fs_flight>-currencycode      = ls_flight-currency.
    <fs_flight>-distance          = ls_flight-distance.
    <fs_flight>-distanceunit      = ls_flight-distid.
    <fs_flight>-flighttime        = ls_flight-fltime.
    <fs_flight>-planetype         = ls_flight-planetype.
    <fs_flight>-departure-airport = ls_flight-airpfrom.
    <fs_flight>-departure-city    = ls_flight-cityfrom.
    <fs_flight>-departure-country = ls_flight-countryfr.
    <fs_flight>-departure-time    = ls_flight-deptime.
    <fs_flight>-arrival-airport   = ls_flight-airpto.
    <fs_flight>-arrival-city      = ls_flight-cityto.
    <fs_flight>-arrival-country   = ls_flight-countryto.
    <fs_flight>-arrival-time      = ls_flight-arrtime.

    concatenate `Flight ` <fs_flight>-airlineid <fs_flight>-connectionno
                ` on ` <fs_flight>-flightdate
           into <fs_flight>-title.
  endif.
endmethod.
```

8. Save and activate your changes.
9. Test a `Flight` from your browser. The easiest way to find the key syntax is to look at an `<atom:link>` element within an `<atom:entry>` element in the `Flights` collection. Copy and paste the relative URL to the end of the Gateway Service’s Base URL.

4.10 Implement the Booking Provider Class

Of the three entities with which we have been working, a `Booking` is the only one that can be updated by the end user. The other two entities are read-only.

Therefore, in addition to the `GET_ENTITYSET` and `GET_ENTITY` methods for `Booking`, we will also need to implement a `CREATE` and `CancelBooking` operation.

There isn't a standard CRUD operation for "Cancel Booking"; therefore, we will make use of the generic method called `EXECUTE_ACTION`. This method allows us to add custom functionality into a Gateway object and extend its range of operations beyond the standard Create, Read, Update and Delete (CRUD) operations.

1. Edit class `Z_CL_DATA_PROVIDER_BOOKING` and create an implementation for method `GET_ENTITYSET`.
2. Replace the implementation with the following code:

```
method /IWBEP/IF_MGW_APPL_SRV_RUNTIME~GET_ENTITYSET.
data:
  lt_bookings type standard table of booking_s,
  ls_booking  type z_cl_model_provider=>booking_s,
  lv_carrid   type sbook-carrid,
  lv_connid   type sbook-connid,
  lv_fldate   type sbook-fldate,
  lv_bookid   type sbook-bookid,
  lv_kline    type /iwbep/s_mgw_name_value_pair,
  lv_lines    type i.

field-symbols:
  <booking>     type booking_s,
  <fs_bookings> type z_cl_model_provider=>bookings_t.

  create data er_entityset type z_cl_model_provider=>bookings_t.
  assign er_entityset->* to <fs_bookings>.

  " Are we following a navigation path?
  if it_navigation_path is initial.
    " Nope, so read all available bookings
    select *
      into corresponding fields of table lt_bookings
     up to is_paging-top rows
     from sbook.
  else.
    " Yup, so read only the booking identified by the key values
    " Were we passed any key values
    if it_key_tab is not initial.
      " Yup, so see if the key table contains the correct number of rows
      describe table it_key_tab lines lv_lines.

      " Did we find three rows
      if lv_lines eq 3.
        " Yup, so search for required key values
        read table it_key_tab into lv_kline with key name = 'AirlineId'.
        if sy-subrc = 0.
          lv_carrid = lv_kline-value.
        endif.

        read table it_key_tab into lv_kline with key name = 'ConnectionNo'.
        if sy-subrc = 0.
          lv_connid = lv_kline-value.
        endif.

        read table it_key_tab into lv_kline with key name = 'FlightDate'.
        if sy-subrc = 0.
          lv_fldate = lv_kline-value.
        endif.
      endif.
    endif.

    " If the required key values are not supplied, then the read will fail
```

```
     " and a null result will be passed back to the client.
     select *
       into corresponding fields of table lt_bookings
      up to is_paging-top rows
       from sbook
      where carrid = lv_carrid and
            connid = lv_connid and
            fldate = lv_fldate.
   endif.

   " Transfer the data from the SBOOK data structure to the export data structure
   loop at lt_bookings assigning <booking>.
     clear ls_booking.

     ls_booking-airlineid      = <booking>-carrid.
     ls_booking-connectionno   = <booking>-connid.
     ls_booking-flightdate     = <booking>-fldate.
     ls_booking-bookingid      = <booking>-bookid.
     ls_booking-customerno     = <booking>-customid.
     ls_booking-customername   = <booking>-passname.
     ls_booking-price          = <booking>-forcuram.
     ls_booking-currencycode   = <booking>-forcurkey.
     ls_booking-bookingdate    = <booking>-order_date.

     concatenate `Booking for ` ls_booking-customername
                 ` on flight ` ls_booking-airlineid ls_booking-connectionno
                 ` on ` ls_booking-flightdate
            into ls_booking-title.

     append ls_booking to <fs_bookings>.
   endloop.
 endmethod.
```

3. Save and activate your changes.
4. Test that the `Bookings` collection is delivered correctly to the browser.
5. Create the implementation for method `GET_ENTITY`.
6. Replace the implementation with the following code:

```
 method /iwbep/if_mgw_appl_srv_runtime~get_entity.
 data:
   ls_booking type booking_s,
   lv_carrid  type sbook-carrid,
   lv_connid  type sbook-connid,
   lv_fldate  type sbook-fldate,
   lv_bookid  type sbook-bookid,
   lv_kline   type /iwbep/s_mgw_name_value_pair,
   lv_lines   type i.

 field-symbols:
   <fs_booking> type z_cl_model_provider=>booking_s.

   create data er_entity type z_cl_model_provider=>booking_s.
   assign er_entity->* to <fs_booking>.

   " Have we been passed any key values?
   if it_key_tab is not initial.
     " Yup, how many key values are there?
     describe table it_key_tab lines lv_lines.

     " 4 key values are needed to locate a booking
     if lv_lines eq 4.
       read table it_key_tab into lv_kline with key name = 'AirlineId'.
       if sy-subrc = 0.
         lv_carrid = lv_kline-value.
       endif.

       read table it_key_tab into lv_kline with key name = 'ConnectionNo'.
       if sy-subrc = 0.
         lv_connid = lv_kline-value.
       endif.
```

```
      read table it_key_tab into lv_kline with key name = 'FlightDate'.
      if sy-subrc = 0.
        lv_fldate = lv_kline-value.
      endif.

      read table it_key_tab into lv_kline with key name = 'BookingId'.
      if sy-subrc = 0.
        lv_bookid = lv_kline-value.
      endif.
    endif.
  endif.

  " Try reading the booking
  select single *
    into corresponding fields of ls_booking
    from sbook
    where carrid = lv_carrid and
          connid = lv_connid and
          fldate = lv_fldate and
          bookid = lv_bookid.

  " Did we find the booking?
  if sy-subrc = 0.
    " Transfer the data from the SBOOK structure to the output structure
    <fs_booking>-airlineid    = ls_booking-carrid.
    <fs_booking>-connectionno = ls_booking-connid.
    <fs_booking>-flightdate   = ls_booking-fldate.
    <fs_booking>-bookingid    = ls_booking-bookid.
    <fs_booking>-customerno   = ls_booking-customid.
    <fs_booking>-customername = ls_booking-passname.
    <fs_booking>-price        = ls_booking-forcuram.
    <fs_booking>-currencycode = ls_booking-forcurkey.
    <fs_booking>-bookingdate  = ls_booking-order_date.

    concatenate `Booking for ` <fs_booking>-customername
                ` on flight ` <fs_booking>-airlineid <fs_booking>-connectionno
                ` on ` <fs_booking>-flightdate
           into <fs_booking>-title.
  endif.
endmethod.
```

7. Save and activate your changes.
8. Test that you can read a single booking from your browser.

4.11 Implement the Create Operation in the Booking Provider Class

As was mentioned at the start of the previous section, the `Flight` and `Airport` entities are read only, but the `Booking` entity can be updated (created and cancelled, in our case). So now we need to look at how such functionality is implemented.

1. In order to create a `Booking`, five values must be supplied:
   a. AirlineId
   b. ConnectionNo
   c. FlightDate
   d. BookingId
   e. Agency
2. At the moment, our data model for a booking does not include an `Agency` property, so this needs to be added. Also, we will be adding cancellation functionality to the Booking entity, so we should also add a `Cancelled` field.
3. Edit type `BOOKING_S` in class `Z_CL_MODEL_PROVIDER` and insert the new properties. Remember that the field names in this data type are internal to the Gateway Data Provider class. They do ***not*** need to be the same as the properties exposed through the Gateway interface.

   Download for code fragment.

```
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,
    agencynum     type sbook-agencynum,
    cancelled     type sbook-cancelled,
    title         type string,
  end of booking_s .
types:
  bookings_t type standard table of booking_s .
```

4. Edit method `DEFINE` in the class `Z_CL_MODEL_PROVIDER` and insert the statements to define the `Agency` and `Cancelled` fields:

   Download for code fragment.

```
  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' ).

  lo_property = lo_entity_type->create_property( iv_property_name  = 'Agency'
                                                 iv_abap_fieldname = 'AGENCYNUM' ).
  lo_property = lo_entity_type->create_property( iv_property_name  = 'Cancelled'
                                                 iv_abap_fieldname = 'CANCELLED' ).

  "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( abap_true ).

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

Also change the Boolean parameter to `abap_true` in the `set_as_title()` method for the `Title` property. This is because in the event of an exception being thrown, we will hijack this field in order to communicate the error message back to the client.

IMPORTANT Remember that property names are case sensitive!

5. Save and activate the Model Provider class.
6. From your browser, reissue the `$metadata` command and check the new properties are visible.
7. Edit class `Z_CL_DATA_PROVIDER_BOOKING` and implement method `CREATE_ENTITY`.
8. Replace the implementation with the following code:

```
method /iwbep/if_mgw_appl_srv_runtime~create_entity.
data:
  ls_booking             type z_cl_model_provider=>booking_s,
  ls_sbook_create        type bapisbonew,
  lt_sbook_create_return type bapirettab,
  lr_sbook_create_return type ref to bapiret2,
  lx_busi_exc            type ref to /iwbep/cx_mgw_busi_exception.

field-symbols:
  <fs_booking> type z_cl_model_provider=>booking_s.

  create data er_entity type z_cl_model_provider=>booking_s.
  assign er_entity->* to <fs_booking>.

  " Read the data supplied by the client
  call method io_data_provider->read_entry_data
    importing es_data = ls_booking.

  " Transfer the data to the input structure for the CREATE BAPI
  ls_sbook_create-airlineid  = ls_booking-airlineid.
  ls_sbook_create-connectid  = ls_booking-connectionno.
  ls_sbook_create-flightdate = ls_booking-flightdate.
  ls_sbook_create-passname   = ls_booking-customername.
  ls_sbook_create-agencynum  = ls_booking-agencynum.
  ls_sbook_create-customerid = ls_booking-customerno.

  " Create the new booking
  call function 'BAPI_FLBOOKING_CREATEFROMDATA'
      exporting booking_data  = ls_sbook_create
      importing airlineid     = ls_booking-airlineid
                bookingnumber = ls_booking-bookingid
         tables return        = lt_sbook_create_return.

  " Read the first row from the return table
  read table lt_sbook_create_return
       index 1
   reference into lr_sbook_create_return.

  " Did the booking creation work?
  if lr_sbook_create_return->*-type ne 'S'.
    " Nope, so create an exception object
    create object lx_busi_exc.
    " Transfer the error message(s) into the exception object
    lx_busi_exc->get_msg_container( )->add_messages_from_bapi(
      it_bapi_messages = lt_sbook_create_return ).
    " Throw toys out of pram
    raise exception lx_busi_exc.
  else.
    " Yup. Whether we do an explicit 'commit work' here depends on
    " whether this functionality is performed in the update task.
    " In update task   => Do an explicit commit work
    " In dialogue task => Gateway framework does the commit for you
    " commit work.

    " Send results back to client
    <fs_booking> = ls_booking.
  endif.
endmethod.
```

9. Save and activate your changes.

10. At the moment, it is not possible to test this functionality because the Runtime Data Provider class `Z_CL_DATA_PROVIDER` has not yet had its `CREATE_ENTITY` method implemented.
11. Edit class `Z_CL_DATA_PROVIDER`.
12. Redefine the inherited method `/IWBEP/IF_MGW_APPL_SRV_RUNTIME~CREATE_ENTITY`.
13. Replace the implementation with the following code:

```
method /iwbep/if_mgw_appl_srv_runtime~create_entity.
data:
  lo_booking type ref to /iwbep/if_mgw_appl_srv_runtime,
  ex_ref     type ref to /iwbep/cx_mgw_busi_exception.

  " Check to which entity type the request belongs
  " In this case, we will only respond to create requests for
  " Booking entity types
  case iv_entity_name.
    when 'Booking'.
      " Create the Booking object and the data structure in
      " which the booking will be returned
      create:
        object lo_booking type z_cl_data_provider_booking,
        data   er_entity  type z_cl_model_provider=>booking_s.

      " The booking creation will throw an exception if it fails.
      " This needs to be caught and handled correctly otherwise
      " the user will get the very unhelpful error message on their
      " browser 'An exception was thrown'
      try.
        " Try to create the booking
        lo_booking->create_entity(
          exporting iv_entity_name     = iv_entity_name
                    iv_entity_set_name = iv_entity_set_name
                    iv_source_name     = iv_source_name
                    io_data_provider   = io_data_provider
                    it_key_tab         = it_key_tab
                    it_navigation_path = it_navigation_path
          importing er_entity          = er_entity ).

        " Has an exception been caught?
        catch /iwbep/cx_mgw_busi_exception into ex_ref.
          " Yup, so hijack the TITLE property to use as a carrier
          " for the error message
          assign er_entity->* to <fs_booking>.
          <fs_booking>-title = ex_ref->get_msg_container( )->get_leading_message_text( ).
      endtry.
  endcase.
endmethod.
```

14. Save and activate your changes.

    To test the creation of a booking, we cannot simply issue a URL from the browser's address line.[6] Instead, you will need to use some type of REST client to supply the required values in OData XML format over an HTTP POST request. Examples of free REST clients are WFetch or the Firefox REST client (installed as an add-on to FireFox).

    The following section on testing the create operation of your Gateway Service assumes that you have installed the Firefox REST client.

15. During the testing of the booking creation process, we will take the "trial and error" approach. This is because in reality, you will very often not be the person who wrote the software that implements the particular business functionality in question. Instead, you will simply be calling someone else's program.

    In this situation, you will often not have a detailed understanding of how the business process works or with what values it should be supplied in order to function correctly.

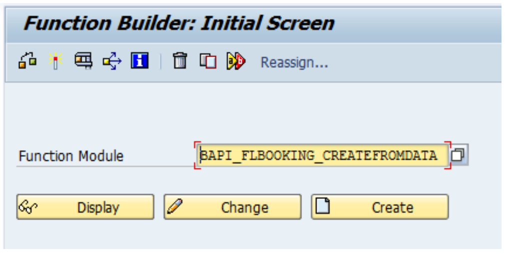
[6] Because this would send the HTTP request using the GET method.

16. First, we need to have a good idea of what fields should be supplied in order to create a booking. After determining what fields are needed, we then need to ensure we put valid data into those fields. This is preparatory work that must be completed before we jump into the creation of the OData XML in the Firefox REST client.
17. Have a look at the `DEFINE` method in class `Z_CL_MODEL_PROVIDER`. This is where we created the metadata for the `Booking` entity type. Here, we created a total of 12 properties of which 4 are marked as keys:

`AirlineId`	key
`ConnectionNo`	key
`FlightDate`	key
`BookingId`	key
`CustomerNo`	
`CustomerName`	
`Price`	
`CurrencyCode`	
`BookingDate`	
`Agency`	
`Cancelled`	
`Title`	

$\}$ So we must supply information at least these key fields in the create request.

18. In order to place meaningful values into the key fields, we must first know which tables the booking creation program will check during the validation process. To find this out, we need to look at the program that creates a booking. This is the ABAP function module `BAPI_FLBOOKING_CREATEFROMDATA`.
19. The simplest way to test how a booking is created is to run this function module.
20. Start transaction `se37`, enter the name `BAPI_FLBOOKING_CREATEFROMDATA` and press F8
21. In the selection screen, place an X in the "Test Run" field because at this point in time, we only want to check that we are supplying valid data.

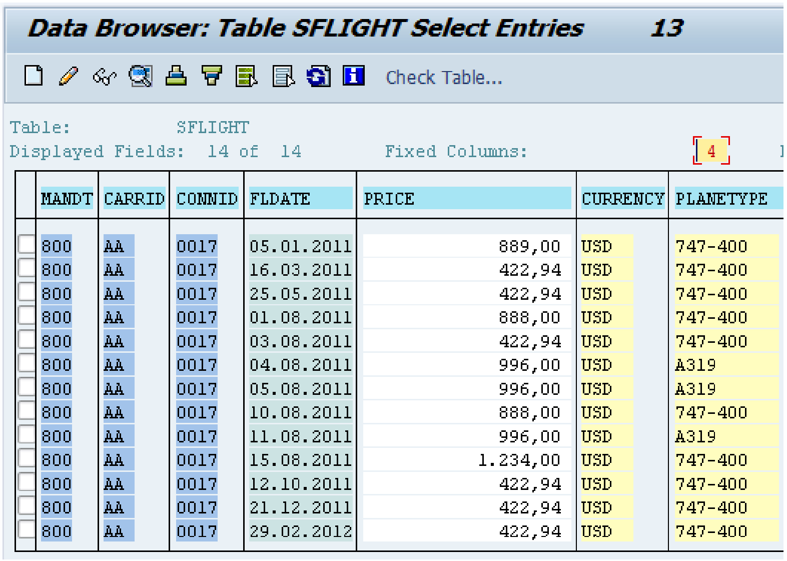

Function Builder: Initial Screen

Function Module: BAPI_FLBOOKING_CREATEFROMDATA

22. Click on the structure icon next to `BOOKING_DATA`. This is the data structure into which we must place our input data. We now need to find out what values to enter.
23. Run transaction `se16` in a new SAPGUI session.

    Look at the contents of table `SFLIGHT`.
24. For the sake of simplicity, we will only look at a single specific flight by a single carrier. In this example, American Airlines flight 17 is chosen, but you can choose any flight you like.

    Here you can see that this flight is being operated on 13 different dates. We need to pick one that's in the future, so we'll go for the flight on December 21st, 2011.

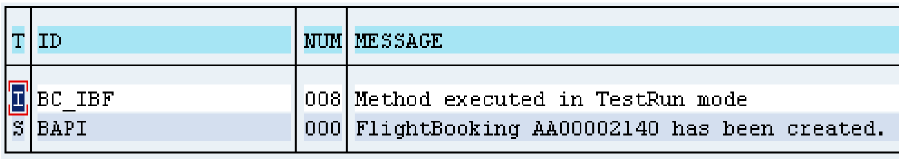

Data Browser: Table SFLIGHT Select Entries 13

```
Table:           SFLIGHT
Displayed Fields:  14 of  14        Fixed Columns:        4
```

MANDT	CARRID	CONNID	FLDATE	PRICE	CURRENCY	PLANETYPE
800	AA	0017	05.01.2011	889,00	USD	747-400
800	AA	0017	16.03.2011	422,94	USD	747-400
800	AA	0017	25.05.2011	422,94	USD	747-400
800	AA	0017	01.08.2011	888,00	USD	747-400
800	AA	0017	03.08.2011	422,94	USD	747-400
800	AA	0017	04.08.2011	996,00	USD	A319
800	AA	0017	05.08.2011	996,00	USD	A319
800	AA	0017	10.08.2011	888,00	USD	747-400
800	AA	0017	11.08.2011	996,00	USD	A319
800	AA	0017	15.08.2011	1.234,00	USD	747-400
800	AA	0017	12.10.2011	422,94	USD	747-400
800	AA	0017	21.12.2011	422,94	USD	747-400
800	AA	0017	29.02.2012	422,94	USD	747-400

25. Switch back to your `se37` screen and enter the carrier id, connection id and flight date for your chosen flight. Press F3 to go back to the selection screen.

26. Lets run the function module with just these values. Press F8 to execute it.
27. Hmmm, nothing – except three rows in the RETURN table.
28. Click the table icon next to the “3 Entries” text and look at the error messages.
29. We have two error messages: the first says that customer number 00000000 is unknown and the second that no counter or agency number has been entered.
30. Switch back to your se16 screen and look at table SCUSTOM for customer numbers.
31. Pick a customer number from this table.
32. Now look at table STRAVELAG to see the travel agencies.
33. Pick a travel agency from this table.
34. Switch back to your se37 screen and enter the customer and agency numbers into the correct fields in structure BOOKING_DATA.
35. Repeat the execution of the function module.
36. You should now see that the RETURN table contains 2 rows:



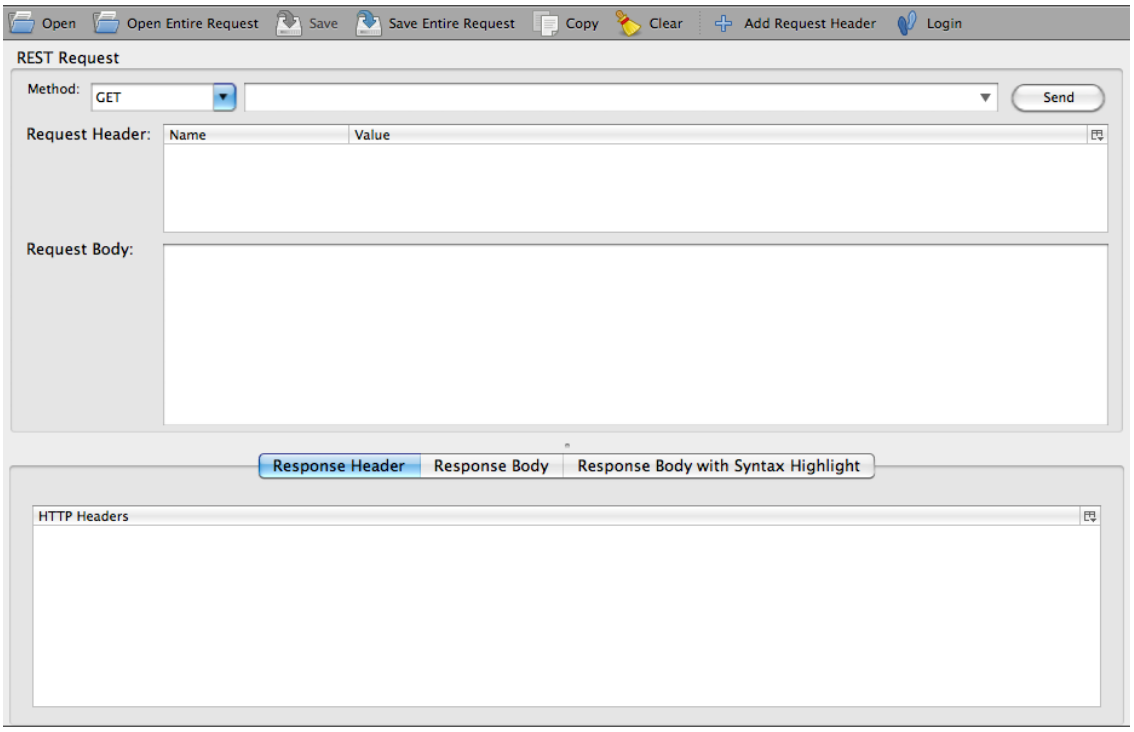
T	ID	NUM	MESSAGE
I	BC_IBF	008	Method executed in TestRun mode
S	BAPI	000	FlightBooking AA00002140 has been created.

37. This tells us that we have supplied the minimum number of required fields in order to create a flight booking. Any other fields we populate will be accepted, but are not required.
38. Now lets build the OData XML needed to run this test from the REST client in Firefox.

4.12 Testing the Create Operation in the Booking Provider Class

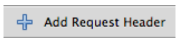
The following instructions assume that you have the Firefox browser installed into which you have added the REST Client add-on.

1. Open the Firefox REST Client



2. The first thing to change is the Method drop down. Change this from `GET` to `POST`.
3. Go back to your browser and copy the URL for displaying the `Bookings` collection into the address line of the REST Client.
4. **<u>IMPORTANT</u>**

   The request sent to the Gateway server ***must*** include the following HTTP header – otherwise the server will just make rude binary noises at you.[7]

   So click on the Add request Header button  in the REST Client toolbar.

Name	`X-Requested-With`
Value	`XMLHttpRequest`

[7] "HTTP 403 Forbidden" to be exact.

5. Next you need to enter the OData XML for the create request. You will need to change the actual data to match the information you used in `se37` when testing `BAPI_FLBOOKING_CREATEFROMDATA`.

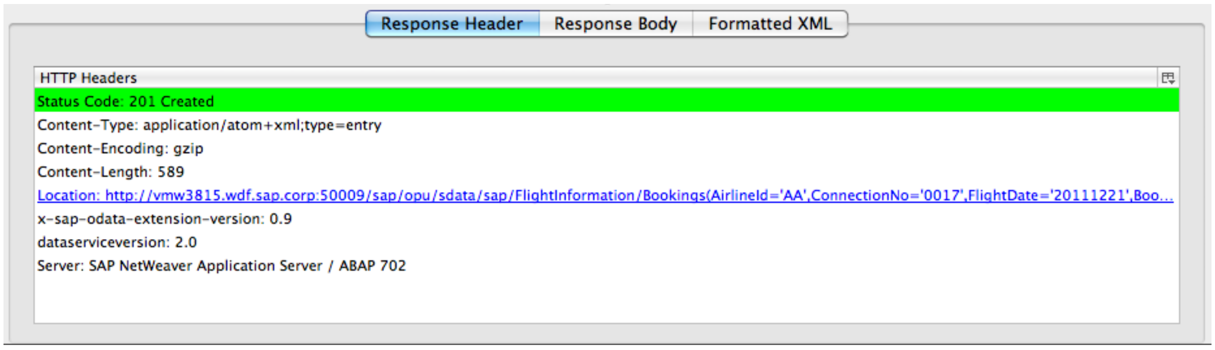
   The data you will need to change is highlighted in yellow.

```
<?xml version="1.0" encoding="utf-8"?>
<atom:entry
   xmlns:atom="http://www.w3.org/2005/Atom"
   xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices"
   xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata">
  <atom:content type="application/xml">
    <m:properties>
      <d:AirlineId>AA</d:AirlineId>
      <d:ConnectionNo>17</d:ConnectionNo>
      <d:FlightDate>2011-12-21T00:00:00</d:FlightDate>
      <d:CustomerNo>00000276</d:CustomerNo>
      <d:Agency>00000055</d:Agency>
    </m:properties>
  </atom:content>
</atom:entry>
```

   Notice the date format must be `YYYY-MM-DDTHH:MM:SS` – where the **-**, **T** and **:** characters are fixed separators.

6. The above XML must start with the standard XML header line
   `<?xml version="1.0" encoding="utf-8"?>`
7. Then comes the `<atom:entry>` element. Since only one booking can be created at a time, the OData document can contain only a single `<atom:entry>` element.

   Within the `<atom:entry>` element, we will be using elements from different namespaces; therefore these namespaces should be defined.
8. Within the `<atom:entry>` element is an `<atom:content>` element.
9. Within the `<atom:content>` element is an `<m:properties>` element.
10. Within the `<m:properties>` element are the `<d:property>` elements that hold the actual information from which the booking will be created.
11. When you have carefully completed the XML in the body section of REST Client, press the Send button. (Watch out for little issues like case-sensitivity in XML element names. If you're not careful, this can completely break the CREATE operation!)
12. You should now see an "HTTP 201 Created" success code in the Response Header section.[8]



```
Status Code: 201 Created
Content-Type: application/atom+xml;type=entry
Content-Encoding: gzip
Content-Length: 589
Location: http://vmw3815.wdf.sap.corp:50009/sap/opu/sdata/sap/FlightInformation/Bookings(AirlineId='AA',ConnectionNo='0017',FlightDate='20111221',Boo...
x-sap-odata-extension-version: 0.9
dataserviceversion: 2.0
Server: SAP NetWeaver Application Server / ABAP 702
```

13. An OData standard is that after every CREATE operation has completed; the client should automatically perform a READ operation. In the "Response Header" section, look at the `Location` parameter in the HTTP header returned to the client. This is the URL to perform the READ operation.

[8] If you see an "HTTP 200 OK", then this is because you forgot to change the HTTP method from GET to POST, and in that case, you will have done nothing more than request the list of Booking. Nothing will have been created.

14. Now select the Formatted XML section. In this section, you see the OData XML returned after the client has performed the automatic READ.

4.13 Implement the Cancel Operation in the Booking Provider Class

The `CancelBooking` action does not fit into the standard Create, Read, Update and Delete (CRUD) operations that people associate with RESTful interfaces. However, there is a general-purpose method called `EXECUTE_ACTION` that can be used to implement custom functionality.

1. In order to implement the `CancelBooking` action, we first need to extend the Gateway Service's Model Provider class. This will declare the existence of a custom action called `CancelBooking`.
2. Edit method `DEFINE` in class `Z_CL_MODEL_PROVIDER`.
3. Add two new declarations at the start of the method:

   Download for code fragment.

```
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,
    lo_association  type ref to /iwbep/if_mgw_odata_assoc,
    lo_action       type ref to /iwbep/if_mgw_odata_action,
    lo_parameter    type ref to /iwbep/if_mgw_odata_parameter.
```

4. Add the following code to the end of the method:

```
************************************************************************************************
* CREATE THE CancelBooking ACTION
************************************************************************************************
  lo_action = model->create_action( 'CancelBooking' ).

  lo_action->set_return_entity_type( 'Booking' ).
  lo_action->set_http_method( 'POST' ).
  lo_action->set_return_multiplicity( cardinality_entity ).

  lo_parameter = lo_action->create_input_parameter( iv_parameter_name = 'AirlineId'
                                                    iv_abap_fieldname = 'AIRLINEID' ).
  lo_parameter->bind_data_element( 'S_CARR_ID' ).

  lo_parameter = lo_action->create_input_parameter( iv_parameter_name = 'ConnectionNo'
                                                    iv_abap_fieldname = 'CONNECTIONNO' ).
  lo_parameter->bind_data_element( 'S_CONN_ID' ).

  lo_parameter = lo_action->create_input_parameter( iv_parameter_name = 'FlightDate'
                                                    iv_abap_fieldname = 'FLIGHTDATE' ).
  lo_parameter->bind_data_element( 'S_DATE' ).

  lo_parameter = lo_action->create_input_parameter( iv_parameter_name = 'BookingId'
                                                    iv_abap_fieldname = 'BOOKINGID' ).
  lo_parameter->bind_data_element( 'S_BOOK_ID' ).
endmethod.
```

5. The above coding does the following:
   a. Creates a new action called `CancelBooking`.
   b. Defines that this action returns a `Booking` entity type.
   c. Defines that this action is initiated by an HTTP POST request (I.E. we are going to perform some type of update).
   d. Defines that a single entity will be returned instead of an entity set.[9]

[9] Here, the terms multiplicity and cardinality are used interchangeably.

e. Next, the properties that define the key of `Booking` are defined as input parameters.

f. Since these input properties do not belong to a distinct entity type, we cannot define the property metadata as we did before by binding the entire entity type to a data type. Instead, we need to bind each input parameter individually to a suitable data type.

6. Save and activate your changes.
7. Test that your metadata changes are correct. Return to your browser and issue the Service Document URL for `FLIGHTINFORMATION` with the additional `$metadata` command. You should see additional XML at the end of the document (inside the `<EntityContainer>` element) similar to this:

```
<FunctionImport
    Name="CancelBooking"
    ReturnType="flightinformation.Booking"
    EntitySet="Bookings"
    m:HttpMethod="POST">
  <Parameter Name="AirlineId" Type="Edm.String" Mode="In">
    <Documentation>
      <Summary />
      <LongDescription>Airline</LongDescription>
    </Documentation>
  </Parameter>
  <Parameter Name="ConnectionNo" Type="Edm.String" Mode="In">
    <Documentation>
      <Summary />
      <LongDescription>Flight Number</LongDescription>
    </Documentation>
  </Parameter>
  <Parameter Name="FlightDate" Type="Edm.DateTime" Mode="In">
    <Documentation>
      <Summary />
      <LongDescription>Date</LongDescription>
    </Documentation>
  </Parameter>
  <Parameter Name="BookingId" Type="Edm.String" Mode="In">
    <Documentation>
      <Summary />
      <LongDescription>Booking number</LongDescription>
    </Documentation>
  </Parameter>
</FunctionImport>
```

8. In transaction `se80`, edit class `Z_CL_DATA_PROVIDER` and redefine the inherited method `/IWBEP/IF_MGW_APPL_SRV_RUNTIME~EXECUTE_ACTION`.
9. Replace the implementation with the following code:

```
method /IWBEP/IF_MGW_APPL_SRV_RUNTIME~EXECUTE_ACTION.
data:
  lo_booking     type ref to /iwbep/if_mgw_appl_srv_runtime,
  ls_rt_booking type z_cl_model_provider=>booking_s.

" The only action that is responded to is CancelBooking
  case iv_action_name.
    when 'CancelBooking'.
      create:
        object lo_booking type z_cl_data_provider_booking,
        data    er_data    type z_cl_model_provider=>booking_s.

      lo_booking->execute_action(
        exporting iv_action_name = iv_action_name
                  it_parameter   = it_parameter
        importing er_data        = er_data ).
  endcase.
endmethod.
```

10. Save your changes. This class cannot yet be activated, as we have not yet implemented the `EXECUTE_ACTION` method in class `Z_CL_DATA_PROVIDER_BOOKING`.
11. In class `Z_CL_DATA_PROVIDER_BOOKING`, edit method `EXECUTE_ACTION`.

12. Replace the implementation with the following code:

```
method /iwbep/if_mgw_appl_srv_runtime~execute_action.
data:
  wa_booking type sbook,
  lv_carrid  type sbook-carrid,
  lv_connid  type sbook-connid,
  lv_fldate  type sbook-fldate,
  lv_bookid  type sbook-bookid,
  lv_kline   type /iwbep/s_mgw_name_value_pair,
  lv_lines   type i.

field-symbols:
  <fs_booking> type z_cl_model_provider=>booking_s.

  " Check what action is being requested
  if iv_action_name = 'CancelBooking'.
    create data er_data type z_cl_model_provider=>booking_s.
    assign er_data->* to <fs_booking>.

    " Were we passed any parameters?
    if it_parameter is not initial.
      " Yup, how many?
      describe table it_parameter lines lv_lines.

      " Must have exactly 4 parameters
      if lv_lines eq 4.
        read table it_parameter into lv_kline with key name = 'AirlineId'.
        if sy-subrc = 0.
          lv_carrid = lv_kline-value.
        endif.

        read table it_parameter into lv_kline with key name = 'ConnectionNo'.
        if sy-subrc = 0.
          lv_connid = lv_kline-value.
        endif.

        read table it_parameter into lv_kline with key name = 'FlightDate'.
        if sy-subrc = 0.
          lv_fldate = lv_kline-value.
        endif.

        read table it_parameter into lv_kline with key name = 'BookingId'.
        if sy-subrc = 0.
          lv_bookid = lv_kline-value.
        endif.
      endif.
    endif.

    " Find the booking to be cancelled
    select single *
      from sbook
      into wa_booking
     where carrid = lv_carrid and
           connid = lv_connid and
           fldate = lv_fldate and
           bookid = lv_bookid.

    " Was the booking found
    if sy-subrc = 0.
      " Set the cancelled flag and update the database
      wa_booking-cancelled = 'X'.
      update sbook from wa_booking.

      " Return the details of the cancelled booking to the client.
      <fs_booking>-airlineid    = wa_booking-carrid.
      <fs_booking>-connectionno = wa_booking-connid.
      <fs_booking>-flightdate   = wa_booking-fldate.
      <fs_booking>-bookingid    = wa_booking-bookid.
      <fs_booking>-customerno   = wa_booking-customid.
      <fs_booking>-customername = wa_booking-passname.
      <fs_booking>-agencynum    = wa_booking-agencynum.
      <fs_booking>-price        = wa_booking-forcuram.
      <fs_booking>-currencycode = wa_booking-forcurkey.
      <fs_booking>-bookingdate  = wa_booking-order_date.
      <fs_booking>-cancelled    = wa_booking-cancelled.

      concatenate `Booking for ` <fs_booking>-customername
                  ` on Flight `  <fs_booking>-airlineid <fs_booking>-connectionno
                  ` on `         <fs_booking>-flightdate
                  ` has been cancelled.`
             into <fs_booking>-title.
    else.
      concatenate `Booking `     lv_bookid
                  ` for flight ` lv_carrid lv_connid
                  ` on `         lv_fldate
                  ` cannot be found.`
             into <fs_booking>-title.
    endif.
  endif.
endmethod.
```

13. Save and activate ***all*** your changes.

4.14 Testing the `CancelBooking` Action in the Booking Provider Class

Since this action has been defined to use the HTTP POST method, we will need to use the Firefox REST Client to perform this test.

1. Before we can cancel a booking, we need to know the `BookingId` of the booking to be cancelled. So, go back to the Firefox REST Client.
2. If the results of your successful booking creation are no longer present, rerun the create process to create a new booking. Look in the Formatted XML and locate the XML element `<d:BookingId>`. Remember the booking id number in this element.
3. Check that the HTTP method is set to POST and that the HTTP header `X-Requested-With` is still present and set to `XMLHttpRequest`.
4. Edit the Base URL of your `FlightInformation` service in the address line and replace the collection name `Bookings` with the action `CancelBooking`:

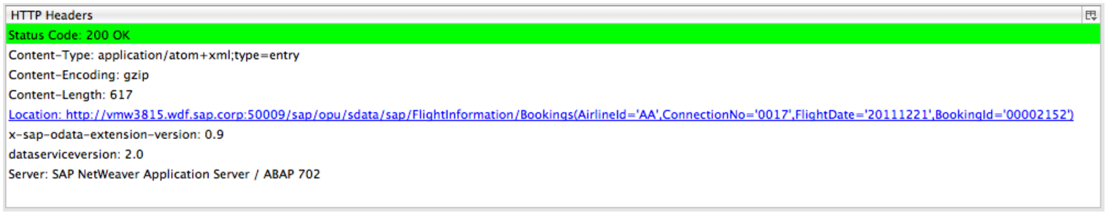
   `.../FlightInformation/CancelBooking`
5. Parameter values for custom actions can only be passed using the query string. Therefore, the values to identify the booking must be supplied as a query string appended to the URL.
6. In the URL, after the `CancelBooking` action, add the property values in query string format:

   `.../CancelBooking?AirlineId=AA&ConnectionNo=0017&FlightDate=20111221&BookingId=00002152`

   The values highlighted in yellow are specific to your particular situation and will need to match the details of the booking you created earlier.

   Notice the following important differences compared to the format of the READ operation:

   - The parameter list is ***not*** enclosed in parentheses
   - The parameter values are ***not*** enclosed in single quotes
   - The date format is YYYYMMDD
7. Press the Send button. You should now see an HTTP 200 OK status code.



```
HTTP Headers
Status Code: 200 OK
Content-Type: application/atom+xml;type=entry
Content-Encoding: gzip
Content-Length: 617
Location: http://vmw3815.wdf.sap.corp:50009/sap/opu/sdata/sap/FlightInformation/Bookings(AirlineId='AA',ConnectionNo='0017',FlightDate='20111221',BookingId='00002152')
x-sap-odata-extension-version: 0.9
dataserviceversion: 2.0
Server: SAP NetWeaver Application Server / ABAP 702
```

5. Summary

By now, you should have a good understanding of how a Gateway Service is defined and implemented using the OData Channel approach. In summary, the development steps are as follows:

1. Understand how your Gateway system has been installed. The SAP NetWeaver Gateway system has been implemented as a set of ABAP add-ons. For the purposes of most development scenarios, the two most important add-ons are `IW_BEP` and `GW_CORE`.
    a. The system in which `IW_BEP` is installed is the one in which the Model Provider and Runtime Data Provider classes are developed.
    b. The system in which `GW_CORE` is installed is the one to which the HTTP client will connect when running the Gateway Service.
    c. The above two components ***do not*** have to be installed on the same ABAP system.
2. Understand exactly what functionality needs to be provided through the Gateway interface.
3. Build a data model that can supply this information. This is implemented as an ABAP class known as a Model Provider class that inherits from class `/IWBEP/CL_MGW_ABS_MODEL`.
4. Within the Model Provider class, you will typically declare the metadata for:
    a. A number of Entities to represent the scalar data types found in your data model. The fields in an Entity can be either primitive data types or complex data types.
    b. A number of entity aggregations known as Entity Sets.
    c. The relationship between entities by the creation of Associations.
    d. Navigation properties based on the above associations. This permits a user to traverse from one side of an association to the other without needing to construct a specific query.
    e. Any number of custom actions to implement functionality over and above the standard Create, Read, Update and Delete (CRUD) operations.
5. Create an ABAP class known as a Runtime Data Provider class. This class must inherit from class `/IWBEP/CL_MGW_ABS_DATA`.
6. Within the Runtime Data Provider class, various standard methods are available within which the required business functionality can be implemented:

a.	`GET_ENTITYSET`	Returns a collection of entities
b.	`GET_ENTITY`	Returns a single entity matching the supplied parameters
c.	`CREATE_ENTITY`	Create an entity based on the supplied parameters
d.	`DELETE_ENTITY`	Delete an entity based on the supplied parameters
e.	`UPDATE_ENTITY`	Update an entity based on the supplied parameters
f.	`EXECUTE_ACTION`	Perform some custom (non-CRUD) action

7. The Model Provider class and the Runtime Data Provider class are not required have any direct reference to each other at the ABAP coding level.[10]
8. In the ABAP system that contains `IW_BEP`, create the configuration that associates the Model Provider and Runtime Data Provider classes. Until this configuration is performed, the above two classes will (from a Gateway perspective) remain non-functional.
9. In the ABAP system that contains `GW_CORE`, activate the Gateway Service. Until this step is been performed, the Gateway Service will remain inaccessible from an HTTP client.

This completes the series of How-To guides on OData Channel programming.

[10] However when declaring data structures in the Runtime Data Provider class, you will probably find it convenient to reference the public data types declared in the Model Provider class.