Tutorial for Creating Resources in the ABAP-Server

This tutorial describes step-by-step how to implement a resource controller for reading a single record of the database table *SFLIGHT*.

Contents

1. Creating the resource controller

The resource controller class implements the access to resource information in the backend.

1.1 Create Resource Controller Class

Open Eclipse and create a new class with the name: *ZXX_CL_SFLIGHT_ADT_RES_FLIGHTS* - replace the *XX* with your own initials. Set *CL_ADT_REST_RESOURCE* as the super class and finish the creation wizard.

> **Remark:**
> You can use the example class ***CL_SFLIGHT_ADT_RES_FLIGHT*** for the single flight resource for further reference. Initially, you may ignore the implementations of PUT, POST and DELETE. The complete example also consists of the collection resource which is implemented in class ***CL_SFLIGHT_ADT_RES_FLIGHT_COLL***. The collection resource is needed when registering the resources for the *discovery service*. You may also skip this step at first but it is a good idea to take a look at the implementation of the POST method.

The abstract base class *CL_ADT_REST_RESOURCE* provides default implementations for the methods GET, POST, PUT, DELETE, which is a subset of the methods the HTTP protocol. For implementing read access of a resource, you have to override the GET method.

Class definition

```
class zxx_cl_sflight_adt_res_flights definition
  public
  inheriting from cl_adt_rest_resource
  final
  create public .

  public section.
    constants:
      co_class_name type seoclsname value 'ZXX_CL_SFLIGHT_ADT_RES_FLIGHTS' ##NO_TEXT.

    methods:
      get  redefinition.

  protected section.

  private section.

endclass.
```

1.2 Implementing the Controller Method GET

The GET method shall read flight information from database table SFLIGHT, serialize the data and send it back to the client. The method will receive three parameters: the carrier ID, the flight ID and the flight date. First, these parameters have to be extracted from the URI. They build the key for reading a single entry from the table SFLIGHT. If the select was successful, the return value is set by invoking the method *SET_BODY_DATA* on the *RESPONSE* object. Otherwise, an exception is raised.

Be aware that the framework already offers a wide range of resource related exceptions. They all follow the pattern: *CX_ADT_RES*_*; you can use the code completion to browse through them.

Use the following link to get more information about Guidelines for Response Status Codes and Resource Exceptions.

In the code snippet below, you can see how URI segment parameters can be accessed. The request method GET_URI_ATTRIBUTE will read a parameter value from the URI. If the parameter exists, the corresponding value is returned, if not, the result is an empty character field equivalent to space. If the existence of a parameter is obligatory, however, you can call the method with the parameter MANDATORY = ABAP_TRUE.

```
data: carrier_id               type s_carr_id,
          connection_id            type s_conn_id,
          flight_date              type s_date,
          wa_flight                type sflight,
          content_handler          type ref to if_adt_rest_content_handler.

*   Access resource id
    request->get_uri_attribute(
      exporting
        name      = 'carrier_id'
        mandatory = abap_true
       importing
        value     = carrier_id ).

    request->get_uri_attribute(
      exporting
        name      = 'connection_id'
        mandatory = abap_true
      importing
        value     = connection_id ).

    request->get_uri_attribute(
      exporting
         name      = 'flight_date'
         mandatory = abap_true
      importing
         value     = flight_date ).

*   Read the resource from the DB
    select single * from sflight into wa_flight where carrid = carrier_id and connid = connection_id
and fldate = flight_date.

    if sy-subrc = 4.
      raise exception type cx_adt_res_not_found.
    else.
*     Get appropriate content handler
      create object content_handler type lcl_flight_content_handler.
*     Set response content
      response->set_body_data( content_handler = content_handler
                               data            = wa_flight ).
    endif.
```

Note: this code still contains a syntax error, because the content handler implementation class is missing - this comes next.

1.4 Implement the Content Handler

Data exchange with the client requires serialization and deserialization of the data. This is accomplished by a content handler, which implements both, serialization and deserialization for a given content type. In practice, almost all content handlers make use of simple transformations. There is already an existing simple transformation SFLIGHT_ADT_ST_FLIGHT that shall be used for our example. In order to use it, a class which implements the content handler interface *IF_ADT_REST_CONTENT_HANDLER* needs to be implemented. Fortunately, there is already a generic content handler which implements the interface of the class and which needs to be initialized with the name of the simple transformation, the name of the data root and the name of the content type.

Create a local class LCL_FLIGHT_CONTENT_HANDLER for the content handler. Go to the tab 'local definitions and implementations' and paste the following code:

```
class lcl_flight_content_handler definition inheriting from cl_adt_rest_st_handler create public.

    public section.

        constants co_content_type type string value if_rest_media_type=>gc_appl_xml.

        methods constructor.

    private section.
        constants:
            co_st_name type string value 'SFLIGHT_ADT_ST_FLIGHT',
            co_root_name type string value 'FLIGHT_DATA'.

endclass.

class lcl_flight_content_handler implementation.

    method constructor.
        super->constructor( st_name = co_st_name root_name = co_root_name content_type =
co_content_type ).
    endmethod.

endclass.
```

1.5 Implementing a Unit Test

Before you proceed, you may create a unit test for the resource controller, i.e. for verifying the correctness of the GET method implemented before. The following link leads you to a tutorial with an implementation example: Tutorial - Unit Test for the Resource Controller GET Method

Start the unit test by pressing SHIFT-CTRL-F10!

Congratulations, you have sucessfully implemented the resource controller! Let's now proceed and register the class, so that it can be invoked by an external REST client.

2. Registering the resource controller

Requests from the client have to be delegated to the appropriate resource controller class. The registration is done by use of a BAdI (see 2.2) with a filter value for the static URI path. The implementations of this BAdI are called *resource application*. A resource application is a domain-specific router for requests. For our tutorial, we will choose a separate name space for our controller and thus have to create an own resource application.

2.1 Create a Resource Application Class

Create a new class *ZXX_CL_SFLIGHT_ADT_RES_APP* by using the creation wizard in eclipse. Set class *CL_ADT_DISC_RES_APP_BASE* as superclass.
It is recommended to use CL_ADT_DISC_RES_APP_BASE as a basis class instead of CL_ADT_RES_APP_BASE since this class supports the implementation of the discovery service for discoverable resources. Our example resource is not discoverable (find more information at Publishing Resources - Discovery).

2.2 Implement a Resource Application Class

- Redefine and implement the methods REGISTER_RESOURCES and GET_APPLICATION_TITLE, as shown below. Replace XX by your initials.
- Please note that the first call of REGISTER_DISCOVERABLE_RESOURCE is shown as an example for a discoverable resource here. You may uncomment the code at first since there is not yet any description in the tutorial how to implement the collection resource ZXX_CL_SFLIGHT_ADT_RES_COLL. If you need to add your resource to the discovery service, please implement the method.

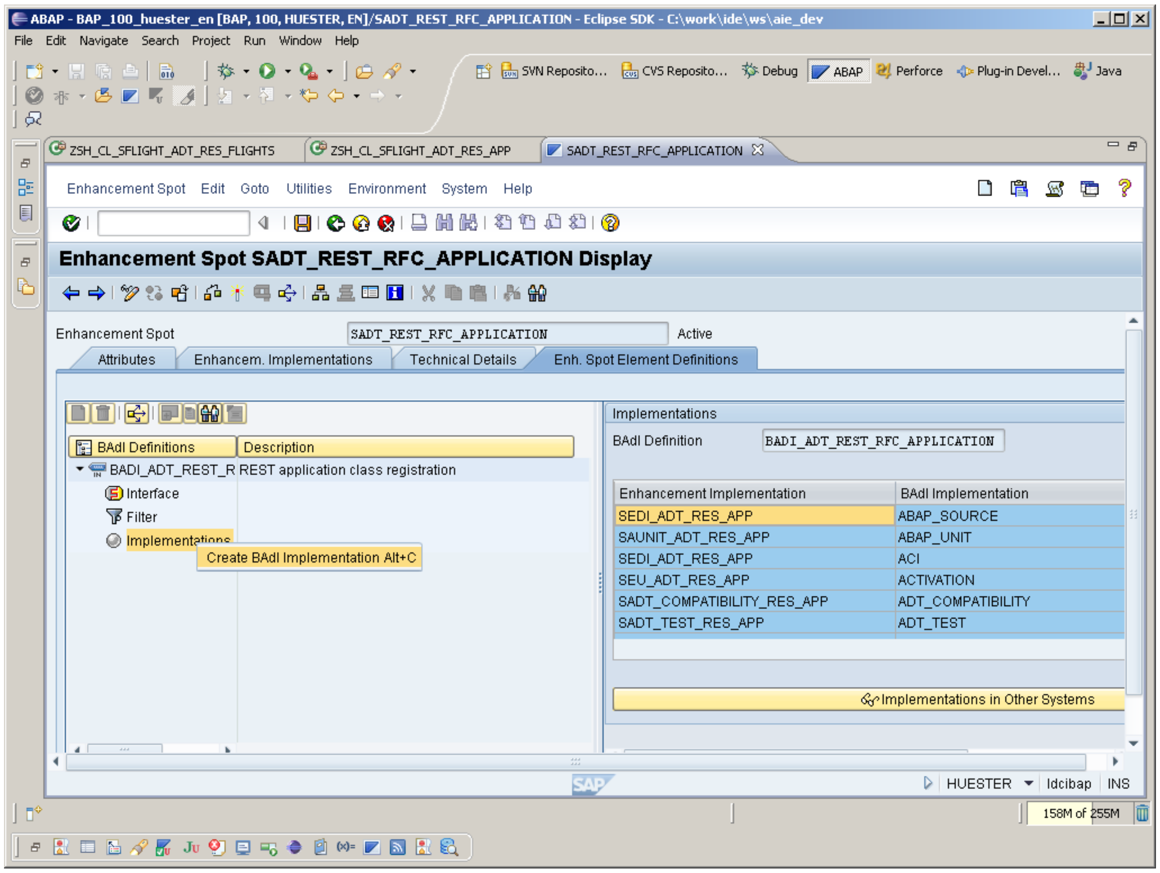
```
class zxx_cl_sflight_adt_res_app definition
  public
  inheriting from cl_adt_disc_res_app_base
  final
  create public .

  public section.
  protected section.
    methods:
      get_application_title redefinition,
      register_resources    redefinition.
  private section.
endclass.


class zxx_cl_sflight_adt_res_app implementation.

  method register_resources.
    registry->register_discoverable_resource(
      exporting
        url             = '/restflights/xx'
        handler_class   = 'zxx_cl_sflight_adt_res_coll=>co_class_name'
        description     = 'Flights'
        category_scheme = 'http://www.sap.com/adt/categories/examples/flights'
category_term   = 'services' ).
    registry->register_resource(
      template      = '/restflights/xx/flights/{carrier_id}/{connection_id}/{flight_date}'
      handler_class = zxx_cl_sflight_adt_res_flights=>co_class_name ).
  endmethod.
  method get_application_title.
    result = 'Flights'.
  endmethod.
endclass.
```

- Check the syntax, activate the class, but **don't leave any kind of syntax errors in the class!**

2.3 Create an Enhancement Implementation

Caution
Be **careful** as this step may have an impact on existing registrations which are used productively! Do **not** activate the implementation before it is explicitly mentioned in the tutorial.

- Open enhancement spot SADT_REST_RFC_APPLICATION by using the "Open ABAP Development Object" functionality of ABAP-in-Eclipse (shortcut: CTRL-SHIFT-A). It might be useful to enlarge the embedded SAP GUI by double-clicking on the Editor Tab. Expand the tree in the left panel, select node *Implementations*, right-click and then click 'Create Badi Implementation'.


ABAP - BAP_100_huester_en [BAP, 100, HUESTER, EN]/SADT_REST_RFC_APPLICATION - Eclipse SDK - C:\work\ide\ws\aie_dev
File Edit Navigate Search Project Run Window Help
ZSH_CL_SFLIGHT_ADT_RES_FLIGHTS
ZSH_CL_SFLIGHT_ADT_RES_APP
SADT_REST_RFC_APPLICATION
Enhancement Spot Edit Goto Utilities Environment System Help
Attributes
Enhancem. Implementations
Technical Details
Enh. Spot Element Definitions
Interface
Filter
Implementations
Create BAdI Implementation Alt+C
Implementations in Other Systems
HUESTER
ldcibap
INS
158M of 255M

Enhancement Spot SADT_REST_RFC_APPLICATION Display

Enhancement Spot SADT_REST_RFC_APPLICATION Active

BAdI Definitions	Description
BADI_ADT_REST_R	REST application class registration

Implementations

BAdI Definition BADI_ADT_REST_RFC_APPLICATION

Enhancement Implementation	BAdI Implementation
SEDI_ADT_RES_APP	ABAP_SOURCE
SAUNIT_ADT_RES_APP	ABAP_UNIT
SEDI_ADT_RES_APP	ACI
SEU_ADT_RES_APP	ACTIVATION
SADT_COMPATIBILITY_RES_APP	ADT_COMPATIBILITY
SADT_TEST_RES_APP	ADT_TEST

- On the popup screen, push the *create* button to create the enhancement implementation. Enter name and description:

Field	**Value**
Enhancement Implementation	ZXX_EXAMPLES_SFLIGHT
Description	SFLIGHT Example - <your name>

- On the modal dialog screen for the package name, enter $tmp as a package name and confirm.
  You have successfully created your own enhancement implementation.
- Do not close the dialog screen. Select the newly created enhancement implementation and push the button *Select Specified Enhancement Implementation*. Enter your data like this:

Field	**Value**
BAdI Implementation	XX_SFLIGHT
Description	SFLIGHT Example - <your name>
Implementing Class	ZXX_CL_SFLIGHT_ADT_RES_APP

(In each case, replace XX by your credentials.)

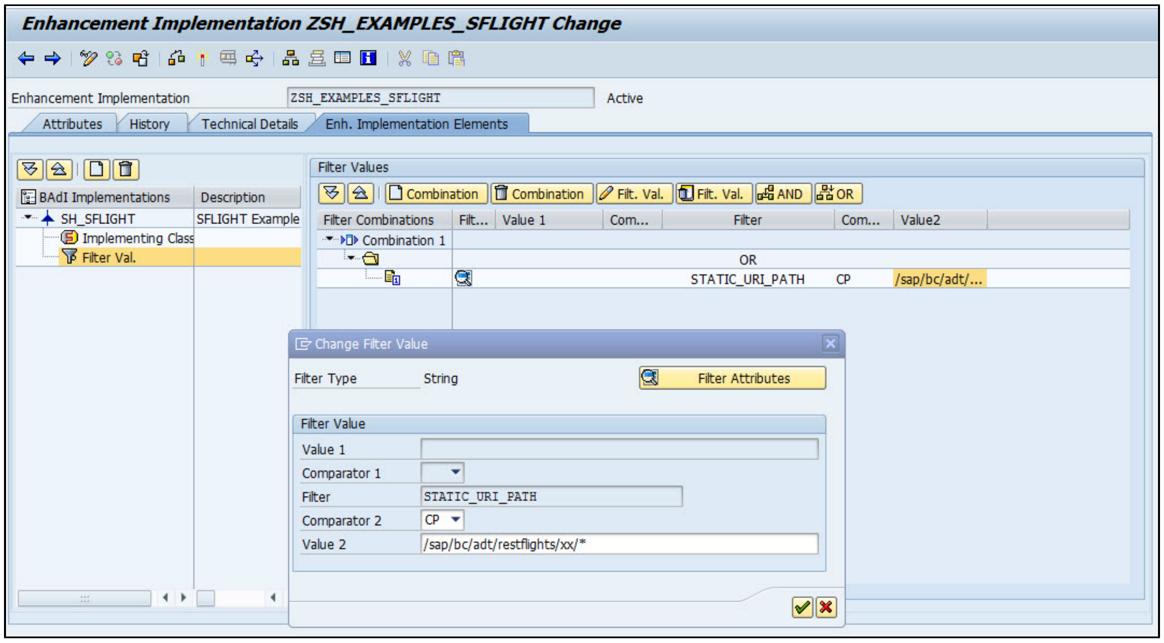
2.4 Assign Filter Values to the Enhancement Implementation

- In the enhancement implementation editor, double-click on the icon for filter values on the left. Press the button *Create filter combination*. Select the line below *Combination 1* and press the button *Change filter value.* In the following dialog, enter

Field	**Value**
Filter	STATIC_URI_PATH
Comparator 2	CP
Value 2	/sap/bc/adt/restflights/xx/*

(Again, replace xx by your credentials.)

Attention: The length of the filter value string must not exceed 40 characters! Longer values will be rejected and you will be informed by an error message.



Enhancement Implementation ZSH_EXAMPLES_SFLIGHT Change

Enhancement Implementation: ZSH_EXAMPLES_SFLIGHT Active

Change Filter Value

Filter Type	String
Value 1	
Comparator 1	
Filter	STATIC_URI_PATH
Comparator 2	CP
Value 2	/sap/bc/adt/restflights/xx/*

- Activate the enhancement implementation.
- Finished! This is all you have to do in the ABAP backend. Next step is to test if the new flight resource can be accessed from the client.

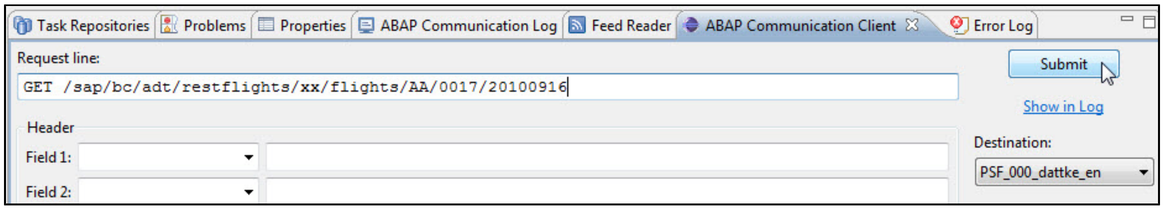
Do not forget to follow the additional instructions at the end of this page to make your resource controller productive

3. Make the Resource Discoverable

This step is necessary if you have decided to make the resource discoverable which is recommended in order to avoid hard coded URI's at the client. In the tutorial, it is not yet explained how to create a collection resource controller but the class CL_SFLIGHT_ADT_RES_FLIGHT_COLL can be used as an example.
In the step before you have implemented the BAdI BADI_ADT_REST_RFC_APPLICATION. Because the implementing class is derived from class CL_ADT_DISC_RES_APP_BASE which implements the interface IF_ADT_DISCOVERY_PROVIDER, you can simply implement the BAdI BADI_ADT_DISCOVERY_PROVIDER by using the same implementing class. Add a new BAdI implementation to the enhancement implementation which may be named XX_SFLIGHT_DISCOVERY with the implementing class ZXX_CL_SFLIGHT_ADT_RES_APP. Save and activate - that's it!
In the next step, it is described how to test the discovery service.

4. Testing by use of the communication client

- Restore the workbench layout by double-click in the Editor Tab of the embedded GUI again.
- Open the communication client view in eclipse with CTRL-3, type "ABAP communication client".
    - Select the right project in the drop down menu.
    - Enter 'GET /sap/bc/adt/restflights/xx/flights/AA/0017/20100916' in the Request field (check the entries in database table SFLIGHT for valid entries before)
    - Press on the submit button

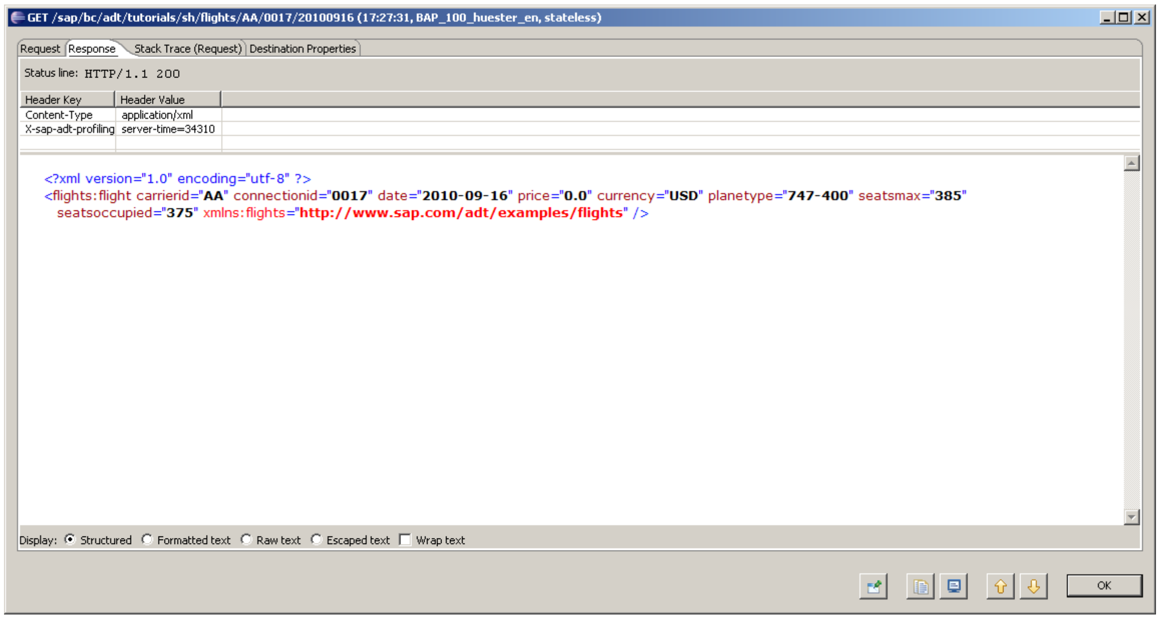


Request line:

```
GET /sap/bc/adt/restflights/xx/flights/AA/0017/20100916
```

Destination: PSF_000_dattke_en

- The ABAP communication log will be shown. The first line should be the request you have just sent to the backend.
    - Double-click on the request line shown in the ABAP communication log
    - To check the response, click on the tab response
- The response status should be OK (200) and the serialized response (XML data) shall be displayed


GET /sap/bc/adt/tutorials/sh/flights/AA/0017/20100916 (17:27:31, BAP_100_huester_en, stateless)
Request
Response
Stack Trace (Request)
Destination Properties
OK

Status line: HTTP/1.1 200

Header Key	Header Value
Content-Type	application/xml
X-sap-adt-profiling	server-time=34310

```
<?xml version="1.0" encoding="utf-8" ?>
<flights:flight carrierid="AA" connectionid="0017" date="2010-09-16" price="0.0" currency="USD" planetype="747-400" seatsmax="385"
  seatsoccupied="375" xmlns:flights="http://www.sap.com/adt/examples/flights" />
```

Display: [✓ Structured] Formatted text Raw text Escaped text ☐ Wrap text

- Proceed in the same way and type in an invalid request such as 'GET /sap/bc/adt/tutorials/xx/flights/XX/0000/18800101'
  - This should result in a response with a *404 Not Found* status

- If the resource is discoverable, enter GET /sap/bc/adt/discovery in the communication client request line. Press submit. You should get a response body including the information you have added in step 2.2!

5. Additional Steps for Resource Controllers to be Used Productively

So far you have successfully developed an own resource controller and tested it in the backend. Once you have decided to deliver the resource controller, please follow the steps below:

- Add a security test for your resource controller and maintain authorization proposals
  There is a detailed description at Security Checklist for Implementation of Resource Controllers.