How-To Guide

Document Version: 1.3 – 2020-12-10

How To... Create RESTful APIs And Consume Them in ABAP Development Tools

Software Development Kit for ABAP Development Tools

Document History

Document Version	Description
1.3	Reworking section “1.4 Prerequisites”
1.2	Adding section “2.4.3.1 Prerequisites”
1.1	Updating section “3.4.6.1 Implementing the Serialize and Deserialize Method” and minor issues
1.0	First official release of this guide

Table of Contents

1 Introduction 4
1.1 Business Scenario 4
1.2 Target Group 4
1.3 Background Information 4
1.4 Prerequisites 5
1.4.1 SAP 5
1.4.2 Eclipse 5
1.5 Implementation 5
1.6 Glossary 5
1.7 Example 6

2 Tutorial for Creating Resources on the AS ABAP 6
2.1 Prerequisites 6
2.2 Creation of the Resource Controller 6
2.2.1 Creating a Resource Controller Class 6
2.2.2 Creating a Simple Transformation 8
2.2.3 Implementing the Controller Method GET 8
2.3 Registration of the Resource Controller 10
2.3.1 Creating a Resource Application Class 10
2.3.2 Implementing a Resource Application Class 10
2.3.3 Creating an Enhancement Implementation 13
2.3.4 Assigning Filter Values to the Enhancement Implementation 14
2.4 Making the Resource Discoverable 14
2.4.1 Registering the Resource Application for the Discovery 14
2.4.2 Assigning Filter Values to the Enhancement Implementation 15
2.4.3 Providing an Own Discovery Resource Application 16

3 Tutorial for Consuming REST Resources in the Eclipse Client 18
3.1 Prerequisites 19
3.2 Creation of an Eclipse Plug-in 19
3.2.1 Creating the Flight Plug-in 19
3.2.2 Declaring the Dependencies 19
3.3 Implementation of a Simple Back End Call 19
3.3.1 Implementing the Handler Method 21
3.4 Deserialization of the REST Response 23
3.4.1 Creating a Class for FlightData 23
3.4.2 Creating a Content Handler 24
3.4.3 Creating a New Plug-In Fragment for Unit Tests 25
3.4.4 Declaring Dependencies 25
3.4.5 Creating a Unit Test for the Content Handler 25
3.4.6 Getting Your Test Running 28
3.4.7 Running the Unit Test 35
3.5 Usage of the Discovery to Get the Flight URI 37

1 Introduction

1.1 Business Scenario

This tutorial shows you how to implement a simple RESTful API in the ABAP back end and how to consume it from the Java client.

It follows the implementation of a resource controller for getting data from a flight resource. This RESTful API allows an Eclipse client to read and display existing flight data from the Application Server ABAP (AS ABAP).

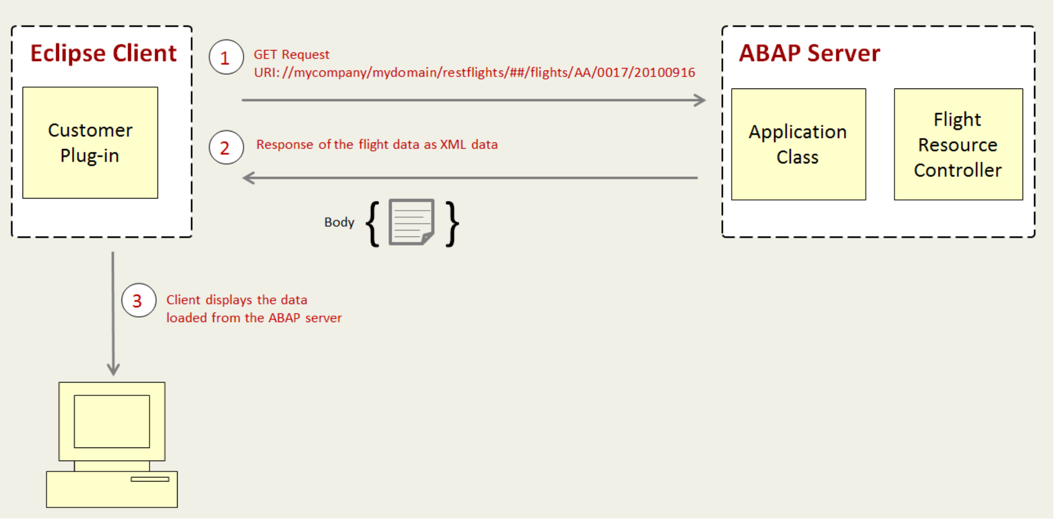

Eclipse Client
Customer Plug-in
1
GET Request
URI: //mycompany/mydomain/restflights/##/flights/AA/0017/20100916
2
Response of the flight data as XML data
Body
ABAP Server
Application Class
Flight Resource Controller
3
Client displays the data loaded from the ABAP server

Figure 1: RESTful API that reads existing flight data from the ABAP server and consumes it on the Eclipse client

1.2 Target Group

Developers who create new Eclipse-based tools.

1.3 Background Information

This tutorial provides guidance on the following topics:

General:

- Communication between the client and the ABAP server

ABAP Server:

- Creating server-side resource controllers that are responsible for handling the API calls
- Serializing and deserializing XML data through simple transformations
- Creating application classes. These make it possible to call resource controllers from the client
- Adding the new API to the discovery API document

Eclipse Client:

- Creating a new Eclipse plug-in
- Calling the API from the client
- Deserializing the response data from the server using content handlers
- Writing integration tests for the RESTful API

1.4 Prerequisites

1.4.1 SAP

This documentation belongs to the latest version of ABAP Development Tools (ADT) and refers to the range of functions that have been shipped as part of the standard delivery for:

- SAP NetWeaver 7.40 SP 2 and higher
- Application Server ABAP 7.53 and higher

Note that SAP Cloud Platform ABAP Environment is currently not supported.

1.4.2 Eclipse

This documentation belongs to the latest version of the supported Eclipse development platform and target platform, for example, Eclipse IDE for Java developers.

1.5 Implementation

ABAP Development Tools are built around the principle of REST (REpresentational State Transfer; see [External link:] REST at Wikipedia). In the first instance, this means you need to consider the resources (that are, entities) that are relevant for your application. These resources will be accessible using standard HTTP methods (GET, PUT, POST, DELETE). In order to be able to access these resources from the client side, you need unique resource identifiers. The ABAP server will serve these resources at their corresponding URIs and the Eclipse client will consume them like a Web browser.

Consequently, we will start on the server side to create a resource controller, implement the GET method, and register it so that client requests can be dispatched. Afterwards, we will implement the client side.

1.6 Glossary

In this document, the following terms are used:

Term	Definition
Hypermedia as the Engine of Application State (HATEOAS)	Constraint of the REST application architecture
Representational State Transfer (REST)	Architectural style for distributed systems such as the World Wide Web
Resource application	Domain-specific router class that is identified using a URI - for example, a resource path. Each resource application is implemented as a BADI implementation. The relevant router class for a given URI is identified through filter values.
Resource controller	Instance that is responsible for creating, reading, updating, and deleting REST resources
RESTful API	API that allows access by a representational state transfer
REST resource	Resources that are identified in requests - for example, using URIs in Web-based REST systems.

1.7 Example

For the data transfer between the client, ABAP server, and database, a custom XML format is used that is produced through a simple transformation.

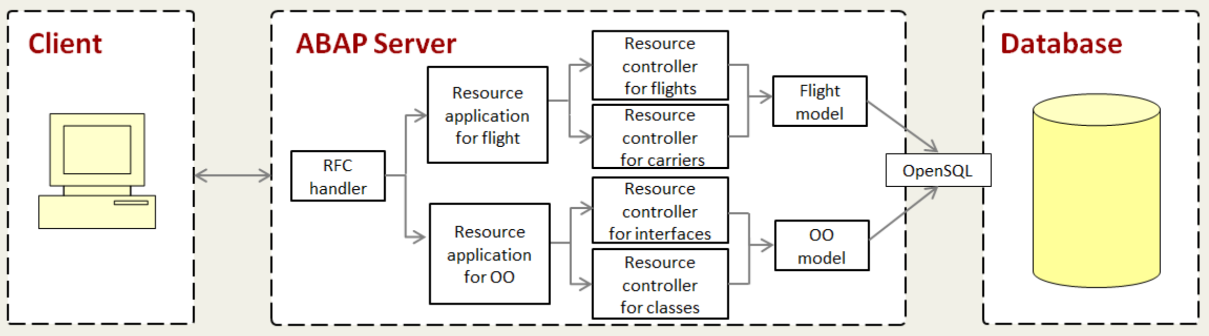

Client
ABAP Server
RFC handler
Resource application for flight
Resource application for OO
Resource controller for flights
Resource controller for carriers
Resource controller for interfaces
Resource controller for classes
Flight model
OO model
OpenSQL
Database

Figure 2: Interaction between the client, ABAP server, and database to display flight information on the Eclipse client

The RFC handler coordinates the data transfer between the client, ABAP server, and the database. It dispatches the client requests to a domain-specific router called resource application. Within the resource application, requests are delegated to single resource controllers for each single resource.

NOTE

In this tutorial, flights in the table *SFLIGHT* are exposed as a resource at the absolute path: `/mycompany/mydomain/restflights/##/flights/{carrier_id}/{connection_id}/{flight_date}`

In all examples, when you create your individual resources or ABAP development objects, replace `##` in the names with an abbreviation. Note that the name is not longer than 7 characters.

2 Tutorial for Creating Resources on the AS ABAP

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

In the following tutorial, you will learn how to:

- Create the resource controller to get access to the information on the back end.
- Register the resource controller in order to delegate the requests from the client.
- Make the resource discoverable in order to avoid hard-coded URIs at the client.

2.1 Prerequisites

If you work with SAP NetWeaver 7.40 SP2, you need to implement SAP Note 1774777 in advance.

To access your implemented URI-resources, you need to register these URIs in the authorization object `S_ADT_RES`. Otherwise, you will get responses with the HTTP-Code 403 Forbidden in 2.4.4.1.

If you need further assistance for access permission, contact your SAP system administrator.

2.2 Creation of the Resource Controller

The resource controller class implements the access to the resource information in the back end. For our purposes, an ABAP Development Tools-specific abstract basis class `CL_ADT_REST_RESOURCE` exists, which will be implemented by our resource controller class.

2.2.1 Creating a Resource Controller Class

Create the resource controller class for a single flight:

1. In ABAP Development Tools, create the ABAP class *`Z_CL_##_SFLIGHT_RES_FLIGHT`*.
2. Using the creation wizard, set *`CL_ADT_REST_RESOURCE`* as the superclass.
3. Confirm the creation with *OK*.

4. Add the definition part as shown in 2.2.1.1 Example.
5. Add the unimplemented methods. Note that the GET method is implemented in 2.2.3 Implementing the Controller Method GET

NOTE

The base class `CL_ADT_REST_RESOURCE` provides default implementations for the methods `GET, POST, PUT,` and `DELETE,` that is, a subset of the methods defined in HTTP. For implementing read access of a resource, you have to override the `GET` method.

2.2.1.1 Example

Definition part of the ABAP class:

```
class Z_CL_##_SFLIGHT_RES_FLIGHT definition
    public
  inheriting from CL_ADT_REST_RESOURCE
  final
  create public .

public section.

  constants CO_CLASS_NAME type SEOCLSNAME value 'Z_CL_##_SFLIGHT_RES_FLIGHT'. "#EC
NOTEXT
  constants CO_RESOURCE_TYPE type STRING value 'SFLIGHT'. "#EC NOTEXT
  constants CO_ST_NAME type STRING value 'Z_##_SFLIGHT_ADT_ST_FLIGHT'. "#EC NOTEXT
  constants CO_ROOT_NAME type STRING value 'FLIGHT_DATA'. "#EC NOTEXT

      " Get content handler for flight data
  class-methods GET_CONTENT_HANDLER
    returning
      value(RESULT) type ref to IF_ADT_REST_CONTENT_HANDLER .
    methods GET redefinition.
      " Get URI for single flight

protected section.

  private section.
    " read flight key from URI
    methods GET_FLIGHT_KEY_FROM_URI
      importing
        REQUEST       type ref to IF_ADT_REST_REQUEST
      exporting
        CARRIER_ID    type S_CARR_ID
        CONNECTION_ID type S_CONN_ID
        FLIGHT_DATE   type S_DATE
      raising
        CX_ADT_REST.

endclass.
```

2.2.2 Creating a Simple Transformation

In order to serialize and deserialize content data, create a simple transformation.

1. Create the repository object with shortcut `Ctrl Shift N`.
2. In the creation dialog, enter *Transformation*.
3. Select the repository object *Transformation* and confirm.
4. Enter the *Object Name* `Z_##_SFLIGHT_ADT_ST_FLIGHT`.
5. Select the *Finish* button.
6. In the SAP GUI creation wizard, enter the *Short Description* “Single Flight”.
7. Select the *Transformation Type “*Simple Transformation”.
8. Confirm with the *OK* button.
9. In the modal dialog screen for the package name, select the *Local Object* button*.*
10. In the tab *SourceCde,* enter the following source code and activate:

```
<?sap.transform simple?>
<tt:transform xmlns:tt="http://www.sap.com/transformation-templates"
xmlns:ddic="http://www.sap.com/abapxml/types/dictionary"
xmlns:flights="http://www.sap.com/adt/examples/flights">


  <tt:root name="flight_data" type="ddic:sflight"/>


  <tt:template>
    <flights:flight tt:extensible="deep" tt:ref="flight_data">
    <tt:attribute name="carrierid" value-ref="carrid"/>
    <tt:attribute name="connectionid" value-ref="connid"/>
    <tt:attribute name="date" value-ref="fldate"/>
    <tt:attribute name="price" value-ref="price"/>
    <tt:attribute name="currency" value-ref="currency"/>
    <tt:attribute name="planetype" value-ref="planetype"/>
    <tt:attribute name="seatsmax" value-ref="seatsmax"/>
    <tt:attribute name="seatsoccupied" value-ref="seatsocc"/>
    </flights:flight>
  </tt:template>
</tt:transform>
```

2.2.3 Implementing the Controller Method GET

The method `GET` reads flight information from the database table `SFLIGHT`, serializes the data, and sends it back to the client. The method receives three parameters: `Carrier ID, flight ID`, and `flight date`. These build the key for reading a single entry from the table `SFLIGHT` and must be extracted from the URI.

If the selection was successful, the return value is set by invoking the method *`SET_BODY_DATA`* on the *`RESPONSE`* object. Otherwise, an exception is raised.

NOTE

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.

2.2.3.1 Example

In the code snippet below, you can see the `GET` method that accesses the URI segment parameters. 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, you can call the method using the parameter `MANDATORY = ABAP_TRUE`.

```
CLASS Z_CL_##_SFLIGHT_RES_FLIGHT IMPLEMENTATION.
```

```
   method GET.

    data: CARRIER_ID               type S_CARR_ID,
          CONNECTION_ID            type S_CONN_ID,
          FLIGHT_DATE              type S_DATE,
          WA_FLIGHT                type SFLIGHT.

    GET_FLIGHT_KEY_FROM_URI(
          exporting
            REQUEST = REQUEST
          importing
            CARRIER_ID    = CARRIER_ID
            CONNECTION_ID = CONNECTION_ID
            FLIGHT_DATE   = FLIGHT_DATE ).

    " Read the resource from the database
    " (simplified example! Do not implement access to the database in the resource
controller!)
    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
      " Set response content
      RESPONSE->SET_BODY_DATA( CONTENT_HANDLER = GET_CONTENT_HANDLER( )
                               DATA            = WA_FLIGHT ).
    endif.

  endmethod.


  method GET_CONTENT_HANDLER.
    RESULT = CL_ADT_REST_CNT_HDL_FACTORY=>GET_INSTANCE( )-
>GET_HANDLER_FOR_XML_USING_ST(
      exporting
        ST_NAME      = CO_ST_NAME
        ROOT_NAME    = CO_ROOT_NAME ).

  endmethod.



  method GET_FLIGHT_KEY_FROM_URI.
    " 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 ).

  endmethod.

ENDCLASS.
```

NOTE

In the example, a generic content handler class is used to call a simple transformation as a content handler (see method `GET_CONTENT_HANDLER`).

2.3 Registration of the Resource Controller

Requests from the client have to be delegated to the appropriate resource controller class. Registration is performed using a BAdI with a filter value for the static URI path, which is the prefix of all resource identifiers (URIs). The implementations of this BAdI are called *resource applications*. A resource application is a domain-specific router for requests.

NOTE

In the tutorial, a separate name space is given for the controller. Thus, you have to create your own resource application.

2.3.1 Creating a Resource Application Class

In ABAP Development Tools, create a new class *`Z_CL_##_SFLIGHT_RES_APP`*. Set class *`CL_ADT_DISC_RES_APP_BASE`* as superclass because this class supports the implementation of the discovery service for discoverable resources.

2.3.2 Implementing a Resource Application Class

1. Redefine and implement the methods `REGISTER_RESOURCES` and `GET_APPLICATION_TITLE`, as shown below.
   NOTE
   Here, the first call of `REGISTER_DISCOVERABLE_RESOURCE` is shown as an example for a discoverable resource. You can uncomment the code at first since there is no description in the tutorial of how to implement the collection resource `Z_CL_##_SFLIGHT_RES_COL`.
2. If you need to add your resource to the discovery service, implement the following class:

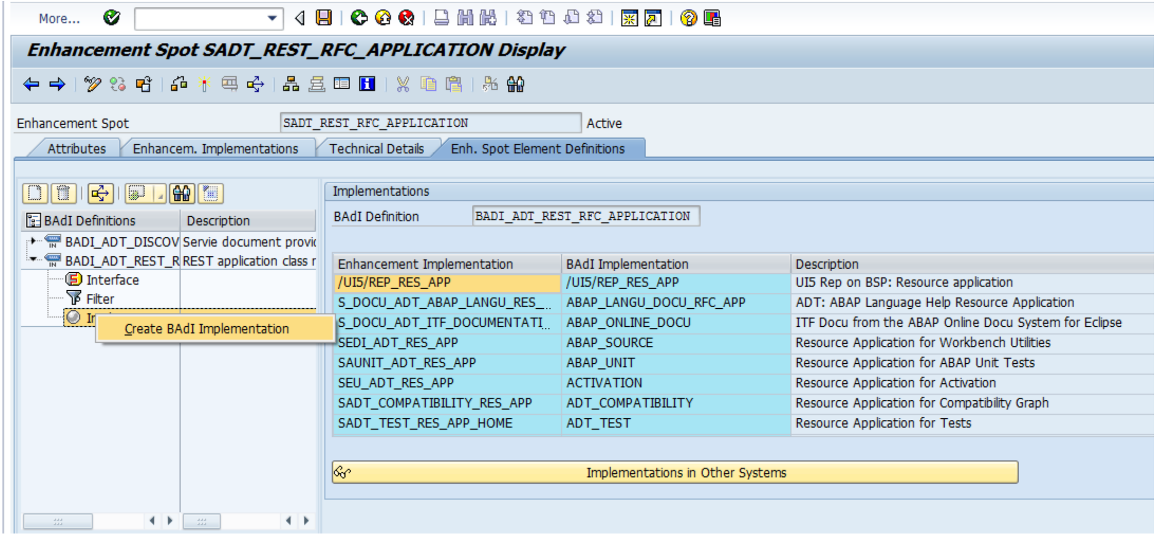
```
class Z_CL_##_SFLIGHT_RES_APP definition
  public
  inheriting from CL_ADT_DISC_RES_APP_BASE
  final
  create public .

public section.
    methods:
      IF_ADT_REST_RFC_APPLICATION~GET_STATIC_URI_PATH redefinition.

  protected section.
    methods:
      GET_APPLICATION_TITLE redefinition ,
      REGISTER_RESOURCES    redefinition.
  private section.
endclass.


class Z_CL_##_SFLIGHT_RES_APP implementation.


method GET_APPLICATION_TITLE.
    RESULT = 'Flights'.
  endmethod.


  method REGISTER_RESOURCES.

    data COLLECTION type ref to IF_ADT_DISCOVERY_COLLECTION.
    REGISTRY->REGISTER_DISCOVERABLE_RESOURCE(
      URL             = '/restflights/##/carriers'
      HANDLER_CLASS   = 'CL_SFLIGHT_ADT_RES_CARRIERS'
      DESCRIPTION     = 'Carriers'
      CATEGORY_SCHEME = 'http://www.mycompany.com/adt/categories/examples/##/flights'
      CATEGORY_TERM   = 'carriers' ).

    COLLECTION = REGISTRY->REGISTER_DISCOVERABLE_RESOURCE(
      exporting
        URL             = '/restflights/##/flights'
        HANDLER_CLASS   = 'Z_CL_##_SFLIGHT_RES_FLIGHT_COL' "Handler class is just used
for example
        DESCRIPTION     = 'Flights'
        CATEGORY_SCHEME =
'http://www.mycompany.com/adt/categories/examples/##/flights'
        CATEGORY_TERM   = 'flights' ).

    COLLECTION->REGISTER_DISC_RES_W_TEMPLATE(
      exporting
```

```
        RELATION      =
'http://www.mycompany.com/adt/relations/examples/##/flights/singleaccess'
        TEMPLATE      =
'/restflights/##/flights/{carrier_id}/{connection_id}/{flight_date}'
        DESCRIPTION   = 'Flight Example'
        TYPE          = 'application/xml'
        HANDLER_CLASS = Z_CL_##_SFLIGHT_RES_FLIGHT=>CO_CLASS_NAME
    ).
   endmethod.

  method IF_ADT_REST_RFC_APPLICATION~GET_STATIC_URI_PATH.
    result = '/mycompany/mydomain'.
  endmethod.

endclass.
```

2.3.3 Creating an Enhancement Implementation

> Caution
> The following steps may have an impact on existing registrations that are used productively!
> Do not activate the implementation until it is explicitly mentioned in the tutorial.

1. In ABAP Development Tools in Eclipse, run the *Open ABAP Development Object* functionality by shortcut `CTRL SHIFT A`.
2. Open the enhancement spot `SADT_REST_RFC_APPLICATION`.
3. In the left panel, expand the node `BADI_ADT_REST_RFC_APPLICATION`.
4. Select the node *Implementations.*
5. In the context menu, choose *Create BAdI Implementation*.



Enhancement Spot SADT_REST_RFC_APPLICATION Display

Enhancement Spot: SADT_REST_RFC_APPLICATION Active

More...
Attributes
Enhancem. Implementations
Technical Details
Enh. Spot Element Definitions
BAdI Definitions
Description
BADI_ADT_DISCOV Servie document provic
BADI_ADT_REST_R REST application class r
Interface
Filter
Create BAdI Implementation
Implementations in Other Systems

Implementations

BAdI Definition: BADI_ADT_REST_RFC_APPLICATION

Enhancement Implementation	BAdI Implementation	Description
/UI5/REP_RES_APP	/UI5/REP_RES_APP	UI5 Rep on BSP: Resource application
S_DOCU_ADT_ABAP_LANGU_RES_..	ABAP_LANGU_DOCU_RFC_APP	ADT: ABAP Language Help Resource Application
S_DOCU_ADT_ITF_DOCUMENTATI..	ABAP_ONLINE_DOCU	ITF Docu from the ABAP Online Docu System for Eclipse
SEDI_ADT_RES_APP	ABAP_SOURCE	Resource Application for Workbench Utilities
SAUNIT_ADT_RES_APP	ABAP_UNIT	Resource Application for ABAP Unit Tests
SEU_ADT_RES_APP	ACTIVATION	Resource Application for Activation
SADT_COMPATIBILITY_RES_APP	ADT_COMPATIBILITY	Resource Application for Compatibility Graph
SADT_TEST_RES_APP_HOME	ADT_TEST	Resource Application for Tests

Figure 3: Integrated SAP GUI for the administration of enhancement spots

6. In the popup window, choose the *Create* button to define the enhancement implementation.
7. Enter the *Name* and *Description and trigger the creation*:

Field	Value
Enhancement Implementation	Z##_SFLIGHT_TUTORIAL
Description	SFLIGHT Tutorial Creating RESTful API's for ABAP in Eclipse
Composite Enhancement Impl.	-

8. In the modal dialog screen for the package name, select the button *Local Object.*
9. Select the created enhancement implementation and double-click.
10. In the creation dialog *BAdI Implementation*, enter the following data:

Field	Value
BAdI Implementation	SFLIGHT_RESOURCE_APPLICATION
Description	Registration of SFLIGHT Resources
Implementing Class	Z_CL_##_SFLIGHT_RES_APP

11. Confirm with the OK button.

2.3.4 Assigning Filter Values to the Enhancement Implementation

1. In the enhancement implementation editor, expand the node `SFLIGHT_RESOURCE_APPLICATION` in the left panel.
2. Double-click the button *Filter Val.*
3. Toggle to the edit mode.
4. In the toolbar, press the *Create Filter Combination* button.
5. Select the line below *Combination 1*.
6. In the toolbar, choose the button *Filt. Val* to c*hange the filter values.*
7. In the following dialog, enter:

Field	Value
Filter	STATIC_URI_PATH
Comparator 2	CP
Value 2	`/mycompany/mydomain/restflights/##/*`

NOTE

The length of the filter value string must not exceed 40 characters! Otherwise, an error message is displayed.

9. Confirm with the OK button.
10. In ABAP Development Tools, refresh the class `Z_CL_##_SFLIGHT_RES_APP` and remove the created interface definition. Afterwards, activate the class.
11. Check if the BAdI *Implementation* is still consistent.
12. Activate the enhancement implementation.
13. Switch back to Display mode.

2.4 Making the Resource Discoverable

The guiding principle for REST services is that clients should make transitions only through actions that are dynamically identified within hypermedia by the server (HATEOAS). As a consequence, clients should not make any assumptions about URIs or use hard-coded URIs. In chapter 3.3 Implementation of a Simple Back End Call, the URI is defined as a constant in the client program, which violates this principle because the server is the only instance that can generate URIs.

Therefore, we will provide a new discovery service as an entry point for the client to explore the services in the back end. The discovery service returns a URI template that can be used by the client to instantiate URIs using standardized rules.

NOTE

You can skip this step and continue with the client implementation first. The discovery service is needed when you implement the client in chapter 3.5 Usage of the Discovery to Get the Flight URI.

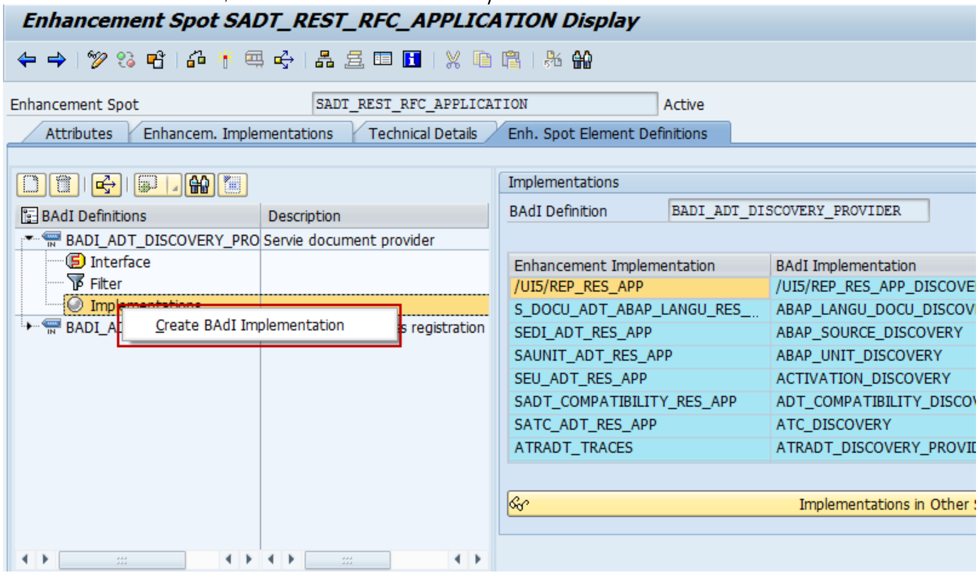
2.4.1Registering the Resource Application for the Discovery

So that the resources are made available for the discovery, the resource application class has to be registered in the discovery BAdI, which is called by a generic resource controller for the discovery document. As a result of the registration of the resource application for a given discovery resource URI, all discoverable resources defined in our resource application will be added to the corresponding discovery document.

After the BAdI `BADI_ADT_REST_RFC_APPLICATION` has been implemented (see 2.3 Registration of the Resource Controller), you can implement the BAdI `BADI_ADT_DISCOVERY_PROVIDER` using the same implementing class.

2.4.1.1 Procedure

1. In ABAP Development Tools in Eclipse, run the Open ABAP Development Object functionality using the shortcut `CTRL SHIFT A`.
2. Open the enhancement spot `SADT_REST_RFC_APPLICATION`.
3. In the left panel, expand the node `BADI_ADT_DISCOVERY_PROVIDER`.
4. Select node *Implementations.*
5. In the context menu, choose *Create BAdI Implementation*.



Enhancement Spot SADT_REST_RFC_APPLICATION Display

Enhancement Spot: SADT_REST_RFC_APPLICATION Active

Attributes
Enhancem. Implementations
Technical Details
Enh. Spot Element Definitions
Create BAdI Implementation
Implementations in Other

BAdI Definitions	Description
BADI_ADT_DISCOVERY_PRO	Servie document provider
Interface	
Filter	
Implementations	
BADI_A	s registration

Implementations

BAdI Definition: BADI_ADT_DISCOVERY_PROVIDER

Enhancement Implementation	BAdI Implementation
/UI5/REP_RES_APP	/UI5/REP_RES_APP_DISCOVE
S_DOCU_ADT_ABAP_LANGU_RES_...	ABAP_LANGU_DOCU_DISCOV
SEDI_ADT_RES_APP	ABAP_SOURCE_DISCOVERY
SAUNIT_ADT_RES_APP	ABAP_UNIT_DISCOVERY
SEU_ADT_RES_APP	ACTIVATION_DISCOVERY
SADT_COMPATIBILITY_RES_APP	ADT_COMPATIBILITY_DISCO
SATC_ADT_RES_APP	ATC_DISCOVERY
ATRADT_TRACES	ATRADT_DISCOVERY_PROVID

Figure 4: Integrated SAP GUI for the administration of enhancement spots

6. In the popup window, select the `Z##_SFLIGHT_TUTORIAL` enhancement implementation.
7. Confirm the selection.
8. Add a new BAdI implementation and enter the following data:

Field	Value
BAdI Implementation	DISCOVERY_PROVIDER
Description	Discovery Provider for Flight Resource Application
Implementing Class	Z_CL_##_SFLIGHT_RES_APP

9. Save the BAdI implementation.
10. Refresh the class `Z_CL_##_SFLIGHT_RES_APP`.
    NOTE
    If syntax errors are generated in the implementation, remove the declaration of the interface `IF_ADT_DISCOVERY_PROVIDER`. Afterwards, activate the class.
11. Activate the created BAdI.

2.4.2 Assigning Filter Values to the Enhancement Implementation

1. In the enhancement implementation editor, expand the node `DISCOVERY_PROVIDER` in the left panel.
2. Double-click the button *Filter Val.*
3. Toggle to the edit mode.
4. In the tool bar, press the *Create Filter Combination* button.
5. Select the line below *Combination 1*.
6. In the toolbar, choose the button *Filt. Val* to c*hange the filter values.*

7. In the following dialog, enter:

Field	Value
Filter	URI
Comparator 1	=
Value 1	`/mycompany/discovery`

NOTE

The length of the filter value string must not exceed 40 characters! Otherwise, an error message is displayed.

14. Confirm with the *OK* button.
15. Check for syntax errors.
16. Activate the enhancement implementation.
17. Switch back to Display mode.

2.4.3 Providing an Own Discovery Resource Application

So far we have just made the discovery information available for the generic resource controller. Now we will register the new discovery resource as a REST service.

MORE INFORMATION

In chapter 3.5 we will see how the discovery document can be consumed by the client.

2.4.3.1 Prerequisites

To implement your own URIs for your own discovery, you have to contact your SAP system administrator in order to get access permission.

If you want to access a resource without access permission, you can use the existing discovery `/sap/bc/adt` which is provided by SAP. In this case, you can continue the tutorial at chapter 3 Tutorial for Consuming REST Resources in the Eclipse Client.

2.4.3.2 Creating a Resource Application Class for the Discovery

The resource is needed when registering the resources for the *discovery service*.

1. In ABAP Development Tools, create the ABAP class *`Z_CL_##_DISCOVERY_RES_APP`*.
2. In the creation wizard, set *`CL_ADT_RES_APP_BASE`* as the super class.
3. Confirm the creation with *OK*.
4. Use code example below to get a valid implementation.
5. Check the created ABAP class for syntax errors.
6. Activate the created ABAP class.

Example

Code example for the created *`Z_CL_##_DISCOVERY_RES_APP`* class:

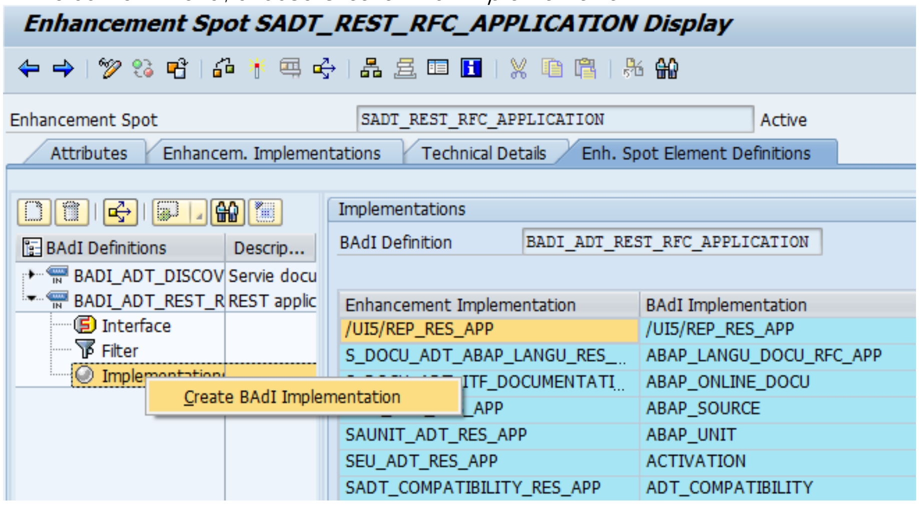
```
class Z_CL_##_DISCOVERY_RES_APP definition
  public
  inheriting from CL_ADT_RES_APP_BASE
  final
  create public .

public section.

  methods IF_ADT_REST_RFC_APPLICATION~GET_STATIC_URI_PATH
```

```
    redefinition .
  protected section.
    methods:
      FILL_ROUTER redefinition.

  private section.
endclass.



class Z_CL_##_DISCOVERY_RES_APP implementation.

  method FILL_ROUTER.
    ROUTER->ATTACH( IV_TEMPLATE = '/discovery' IV_HANDLER_CLASS =
CL_ADT_RES_DISCOVERY=>CO_CLASS_NAME ).
  endmethod.

  method IF_ADT_REST_RFC_APPLICATION~GET_STATIC_URI_PATH.
    RESULT = '/mycompany'.
  endmethod.
endclass.
```

2.4.3.3 Creating the BAdI Implementation for the Discovery Resource Application

> Caution
> The following steps may have an impact on existing registrations that are used productively!
> Do not activate the implementation until it is explicitly mentioned in the tutorial.

1. In ABAP Development Tools in Eclipse, run the Open ABAP Development Object functionality using the shortcut `CTRL SHIFT A`.
2. Open the enhancement spot `SADT_REST_RFC_APPLICATION`.
3. In the left panel of the tree, expand the tree for `BADI_ADT_REST_RFC_APPLICATION`.
4. Select the node *Implementations*.
5. In the context menu, choose *Create BAdI Implementation*.



Enhancement Spot Display

Enhancement Spot SADT_REST_RFC_APPLICATION Active

BAdI Definition BADI_ADT_REST_RFC_APPLICATION

BAdI Definitions	Descrip...
BADI_ADT_DISCOV	Servie docu
BADI_ADT_REST_R	REST applic
Interface	
Filter	
Implementation	

Enhancement Implementation	BAdI Implementation
/UI5/REP_RES_APP	/UI5/REP_RES_APP
S_DOCU_ADT_ABAP_LANGU_RES_...	ABAP_LANGU_DOCU_RFC_APP
S_DOCU_ADT_ITF_DOCUMENTATI...	ABAP_ONLINE_DOCU
_APP	ABAP_SOURCE
SAUNIT_ADT_RES_APP	ABAP_UNIT
SEU_ADT_RES_APP	ACTIVATION
SADT_COMPATIBILITY_RES_APP	ADT_COMPATIBILITY

Enhancement Spot SADT_REST_RFC_APPLICATION Display
Attributes
Enhancem. Implementations
Technical Details
Enh. Spot Element Definitions
Implementations
Create BAdI Implementation

Figure 5: Integrated SAP GUI for the administration of enhancement spots

4. In the popup window, select the `Z##_SFLIGHT_TUTORIAL` enhancement implementation.
5. Confirm the selection.
6. Add a new BAdI implementation.
   Enter the following data:

Field	Value
BAdI Implementation	DISCOVERY_RESOURCE_MYCOMPANY
Description	Registration of the Discovery Resource
Implementing Class	Z_CL_##_DISCOVERY_RES_APP

2.4.3.4 Assigning the Filter Values to the Enhancement Implementation

1. In the enhancement implementation editor, expand the node `DISCOVERY_RESOURCE_MYCOMPANY` in the left panel.
2. Double-click the button `Filter Val.`
3. Toggle to the edit mode.
4. In the toolbar, select the *Combination* button.
5. Select the line below *Combination 1.*
6. In the toolbar, choose the button *Filt. Val* to change the filter values.
7. In the following dialog, enter:

Field	Value
Value 1	`/mycompany/discovery`
Comparator 1	=
Filter	`STATIC_URI_PATH`

NOTE
The length of the filter value string must not exceed 40 characters! Otherwise, an error message is displayed.

8. Confirm with the OK button.
9. Refresh the class `Z_CL_##_DISCOVERY_RES_APP`.
   NOTE
   If syntax errors are generated in the implementation, remove the declaration of the interface IF_ADT_REST_RFC_APPLICATION. Afterwards, activate the class.
10. Activate the enhancement implementation.

3 Tutorial for Consuming REST Resources in the Eclipse Client

After providing the back-end resource for our flight example, we will continue implementing the example at the Eclipse client.

We will start with the existing example plug-in for the HelloWorld command handler and modify the command handler to call our code.

MORE INFORMATION

[External Link: http://www.eclipse.org/resources]

3.1 Prerequisites

- An Eclipse target platform is added in the Run Configuration
- In the target platform, at least one ABAP project is added.
- Table `Sflight` should be filled with the corresponding entries. This table can be filled with entries by running the report `SAPBC_DATA_GENERATOR`.

3.2 Creation of an Eclipse Plug-in

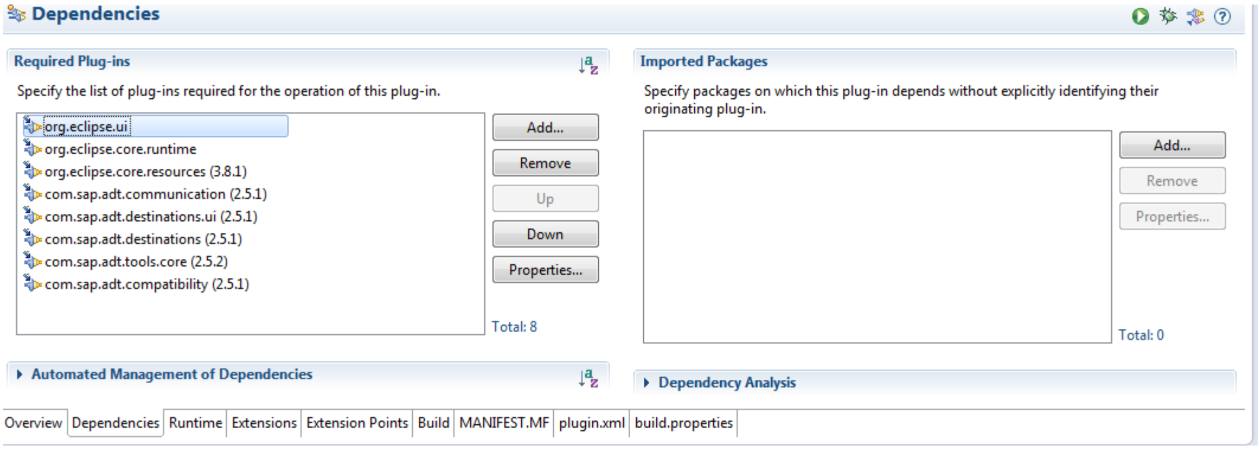
When you develop productive plug-ins, it is strongly recommended that you separate the test code and productive code by creating two plug-ins or a plug-in and a fragment. In order to keep our example simple, we will start with a single plug-in.

3.2.1 Creating the Flight Plug-in

Deploy the HelloWorld example in our workspace and verify that it works:

1. In the ABAP Development Tools, switch to the *Plug-In Development* perspective.
2. Create a new plug-in project
3. In the *New Plug-in Project* wizard page, enter `com.mycompany.adt.##.flight` as Name.
   NOTE
   Use your own namespace so that you can also import the sample solution in parallel.
4. Select *NEXT* to open the *Content* wizard page.
5. Accept the defaults and select *NEXT* to open the *Templates* wizard page*.*
6. Choose Hello, World Command as a template.
7. Select *Finish.*
8. Run the Eclipse target platform.
9. Choose the Sample Menu button to open a dialog window with the text Hello, Eclipse world.

3.2.2 Declaring the Dependencies



Dependencies

Required Plug-ins

Specify the list of plug-ins required for the operation of this plug-in.

- org.eclipse.ui
- org.eclipse.core.runtime
- org.eclipse.core.resources (3.8.1)
- com.sap.adt.communication (2.5.1)
- com.sap.adt.destinations.ui (2.5.1)
- com.sap.adt.destinations (2.5.1)
- com.sap.adt.tools.core (2.5.2)
- com.sap.adt.compatibility (2.5.1)

Total: 8

Imported Packages

Specify packages on which this plug-in depends without explicitly identifying their originating plug-in.

Total: 0

Automated Management of Dependencies

Dependency Analysis

Figure 6: Example of the required plug-in dependencies

In the tab *Dependencies*, add the following additional *Required Plug-ins*:

- `org.eclipse.core.resources`
- `com.sap.adt.communication`
- `com.sap.adt.compatibility`
- `com.sap.adt.destinations`
- `com.sap.adt.destinations.ui`
- `com.sap.adt.tools.core`

3.3 Implementation of a Simple Back End Call

In package `com.mycompany.adt.##.flight.handlers` you will find the sample handler class that contains an execute method invoked by a user action. Now you modify this code in order to call the `HTTP` method `GET` on the flight resource.

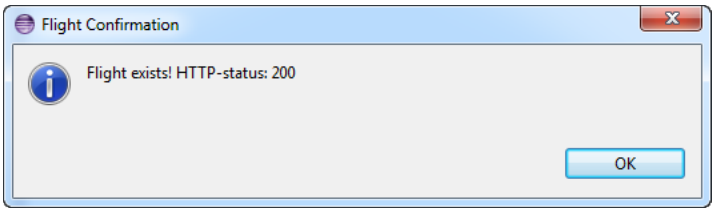
3.3.1 Implementing the Handler Method

1. Implement the `SampleHandler` class as follows:

```
package com.mycompany.adt.##.flight.handlers;

import java.net.URI;

import org.eclipse.core.commands.AbstractHandler;
import org.eclipse.core.commands.ExecutionEvent;
import org.eclipse.core.commands.ExecutionException;
import org.eclipse.core.resources.IProject;
import org.eclipse.jface.dialogs.IDialogConstants;
import org.eclipse.jface.dialogs.MessageDialog;
import org.eclipse.swt.widgets.Shell;
import org.eclipse.ui.PlatformUI;

import com.sap.adt.communication.message.IResponse;
import com.sap.adt.communication.resources.AdtRestResourceFactory;
import com.sap.adt.communication.resources.IRestResource;
import com.sap.adt.communication.resources.IRestResourceFactory;
import com.sap.adt.communication.resources.ResourceNotFoundException;
import com.sap.adt.destinations.ui.logon.AdtLogonServiceUIFactory;
import com.sap.adt.tools.core.project.AdtProjectServiceFactory;
import com.sap.adt.tools.core.project.IAbapProject;

/* Our sample handler extends AbstractHandler, an IHandler base class.
 *
 * @see org.eclipse.core.commands.IHandler
 * @see org.eclipse.core.commands.AbstractHandler
 */
public class SampleHandler extends AbstractHandler {
    // URI for access to our sample resource. The hard coded URI will be
    // replaced by a dynamic one using
    // URI templates in a second step.
    // AVOID USING HARD-CODED URI'S AT THE CLIENT! USE THE DISCOVERY SERVICE
    // INSTEAD!
    //Ensure that the flight exists in the database table before execution
    private static final String SAMPLE_FLIGHT_RESOURCE_URI =
"/mycompany/mydomain/restflights/##/flights/AA/0017/20120815";

    /**
     * The constructor.
     */
    public SampleHandler() {
    }

    /**
      * Handler method
    */
   public Object execute(ExecutionEvent event) throws ExecutionException {
      // Create resource factory
      IRestResourceFactory restResourceFactory = AdtRestResourceFactory
            .createRestResourceFactory();
      // Get available projects in the workspace
      IProject[] abapProjects = AdtProjectServiceFactory
            .createProjectService().getAvailableAbapProjects();
      // Use the first project in the workspace for the demo
      // to keep the example simple
      IAbapProject abapProject = (IAbapProject) abapProjects[0]
            .getAdapter(IAbapProject.class);
      // Trigger logon dialog if necessary
      AdtLogonServiceUIFactory.createLogonServiceUI().ensureLoggedOn(
            abapProject.getDestinationData(),
            PlatformUI.getWorkbench().getProgressService());
      // Create REST resource for given destination and URI
      String destination = abapProject.getDestinationId();
      URI flightUri = URI.create(SAMPLE_FLIGHT_RESOURCE_URI);
      IRestResource flightResource = restResourceFactory
            .createResourceWithStatelessSession(flightUri, destination);
      try {
         // Trigger GET request on resource data
         IResponse response = flightResource.get(null,
               IResponse.class);
         // confirm that flight exists
         openDialogWindow("Flight exists! HTTP-status:
"+String.valueOf(response.getStatus()), "Flight Confirmation");
      } catch (ResourceNotFoundException e) {
         displayError("No flight data found");
      } catch (RuntimeException e) {
         // Display any kind of other error
         displayError(e.getMessage());
      }
      return null;
   }

   /*
    * Display the exception text
    */
   private void displayError(String messageText) {
      String dialogTitle = "Flight Exception";
      openDialogWindow(messageText, dialogTitle);
   }

   /*
    * Display a simple dialog box with a text and an OK button to confirm
    */
    protected void openDialogWindow(String dialogText, String dialogTitle) {
        String[] DIALOG_BUTTON_LABELS = new String[] { IDialogConstants.OK_LABEL };
        MessageDialog dialog = new MessageDialog(getShell(), dialogTitle, null,
                dialogText, MessageDialog.INFORMATION, DIALOG_BUTTON_LABELS, 0);
        dialog.open();
    }

    protected Shell getShell() {
        Shell shell = PlatformUI.getWorkbench().getActiveWorkbenchWindow()
                .getShell();
        return shell;
    }

}
```

2. *Use shortcut `Shift Ctrl O` to organize the imports.*
3. *Check the `SFLIGHT` table in the backend for existing flights and modify the constant* `SAMPLE_FLIGHT_RESOURCE_URI` so that it represents an existing flight.
4. *In the toolbar, choose the* ▶ *Run As... button.*
5. *Start the Target IDE by using a specific run configuration.*
6. *In the menu bar of Target IDE, choose Sample Menu to run the command.*

If the connection works, the following message is displayed:



Flight exists! HTTP-status: 200

Figure 7: Message that the requested flight exists

Note

For this, authorizations are required. See more in SAP Note 1657744.

3.4 Deserialization of the REST Response

In the back end, a simple transformation is implemented that is used to deserialize and serialize the content data of the request or response, respectively.

On the client side, a specific content handler is used that is able to serialize and deserialize the XML flight data.

In this tutorial, we will create an empty content handler stub, provide a unit test, and implement the content handler using a test-driven approach.

3.4.1Creating a Class for FlightData

Define a simple `FlightData` class that represents a single flight instance.

1. In the src folder of the plugin, create the *package* `com.mycompany.adt.##.flight.data.`
2. In the package, create the class `FlightData` using the following code:

```
// (imports skipped)
public class FlightData {

    private String carrierId;
    private String connectionId;
    private String flightDate;
    private String planeType;
    private String price;
    private String currency;
    private String seatsMax;
    private String seatsOccupied;

    public FlightData() {}

    public FlightData(String carrierId,
                           String connectionId,
                           String flightDate,
                           String planeType,
                           String price, String currency,
                           String seatsMax,
                           String seatsOccupied) {
        super();
        this.carrierId = carrierId;
        this.connectionId = connectionId;
        this.flightDate = flightDate;
        this.planeType = planeType;
        this.price = price;
        this.currency = currency;
        this.seatsMax = seatsMax;
        this.seatsOccupied = seatsOccupied;
    }
}
```

NOTE

You will receive compiler warnings for the unused private attributes. In the source code editor, use the shortcut `ALT SHIFT S` to add the following methods automatically:

- Use Generate Getters and Setters...
- Use Override/Implement Methods to:
    - Add a toString() method
    - Add a hashCode() method
    - Add an equals() method

The warnings will disappear.

3.4.2 Creating a Content Handler

Now you create a content handler with a corresponding unit test.

1. In the package *`com.mycompany.adt.##.flight.data`*, create the class `FlightDataContentHandler` that implements the interface `IContentHandler`.
2. Create stubs for all interface methods.
3. Implement the following methods `getSupportedContentType` and `getSupportedDataType`:

```
// (imports skipped)
public class FlightDataContentHandler implements IContentHandler<FlightData> {


    @Override
    public FlightData deserialize(IMessageBody body,
            Class<? extends FlightData> dataType) {
        // TODO Auto-generated method stub
        return null;
    }

    @Override
    public IMessageBody serialize(FlightData dataObject, Charset charset) {
        // TODO Auto-generated method stub
        return null;
    }

    @Override
    public String getSupportedContentType() {
        return AdtMediaType.APPLICATION_XML;
    }

    @Override
    public Class<FlightData> getSupportedDataType() {
        return FlightData.class;
    }
}
```

3.4.3 Creating a New Plug-In Fragment for Unit Tests

1. Create a fragment project.
2. In the *New Fragment Project* wizard page, enter `com.mycompany.adt.##.flight.test` as project name
3. Accept the defaults on the first page and press next
4. Enter `com.mycompany.adt.##.flight` as *Host Plug-in*.
5. Select *Finish.*

3.4.4 Declaring Dependencies

In the dependencies plug-in, add the required plug-in `org.junit(4.11.0).`

3.4.5 Creating a Unit Test for the Content Handler

1. In the fragment plug-in, create the package *`com.mycompany.adt.##.flight.data.`*

NOTE

This package already exists in the "productive" Plug-in. Because we add tests for the *.data-Plug-in, the tests should be part of the same package.

2. Accept the defaults on the first page and press *Next*.
3. In the new package, create the class `TestsUnitFlightDataContentHandler` and add the following source code:

```
package com.mycompany.adt.##.flight.data;


import static org.junit.Assert.*;


import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.Reader;
import java.nio.charset.Charset;
import org.junit.Test;

import com.sap.adt.communication.content.AdtMediaType;
import com.sap.adt.communication.message.IMessageBody;
import com.sap.adt.communication.message.InputStreamMessageBody;

public class TestsUnitFlightDataContentHandler {

    private static final String TESTFILE_XML = "testflight.xml";
    private static final String PLANETYPE = "747-400";
    private static final String CURRENCY = "USD";
    private static final String CONNECTION_ID = "0017";
    private static final String CARRIER_ID = "AA";
    private static final String FLIGHT_DATE = "2010-09-16";
    private static final String PRICE = "500.00";
    private static final String SEATSMAX = "385";
    private static final String SEATS_OCCUPIED = "375";

    @Test
    public void deserialize() throws Exception {

        FlightDataContentHandler contentHandler = new FlightDataContentHandler();

        IMessageBody messageBody = createMessageBody(TESTFILE_XML);

        FlightData flightData = contentHandler.deserialize(messageBody,
FlightData.class);

        assertNotNull("Flight data is null", flightData);
        assertEquals(CARRIER_ID, flightData.getCarrierId());
        assertEquals(CONNECTION_ID, flightData.getConnectionId());
        assertEquals(FLIGHT_DATE, flightData.getFlightDate());
        assertEquals(PRICE, flightData.getPrice());
        assertEquals(CURRENCY, flightData.getCurrency());
        assertEquals(PLANETYPE, flightData.getPlaneType());
        assertEquals(SEATSMAX, flightData.getSeatsMax());
        assertEquals(SEATS_OCCUPIED, flightData.getSeatsOccupied());
    }

    @Test
    public void serialize() throws Exception {
        FlightDataContentHandler contentHandler = new FlightDataContentHandler();
        FlightData flightData = new FlightData(CARRIER_ID, CONNECTION_ID, FLIGHT_DATE,
PLANETYPE, PRICE, CURRENCY, SEATSMAX, SEATS_OCCUPIED);

        // Prepare expected string:
        IMessageBody expectedBody = createMessageBody(TESTFILE_XML);
        String expectedXML =
streamToString(expectedBody.getContent()).replaceAll("[\n\r ]", "");

        // Call the serialization to get the actual string:
        IMessageBody actBody = contentHandler.serialize(flightData, null);
        String actualXml = streamToString(actBody.getContent()).replaceAll("[\n\r ]",
"");
        //
        assertEquals(expectedXML, actualXml);
    }

    protected IMessageBody createMessageBody(String name) throws IOException {
        InputStream stream = getClass().getResourceAsStream(name);

        if (stream == null) {
            fail("No such file: '" + name + "' in " +

getClass().getProtectionDomain().getCodeSource().getLocation());
            return null; // never happens
        }

        return new InputStreamMessageBody(AdtMediaType.APPLICATION_XML, stream);
    }
    /**
     * Reads from the given input stream into a string; uses UTF-8 encoding
     *
     * @param inputStream
     *            the stream to read. Note that the <b>caller is responsible for
     *            its closing</b>.
     * @return the resulting string
     * @throws IOException
     *             in case of failures
     */
    public static String streamToString(InputStream inputStream) throws IOException {
        /**
         * The charset with name <code>UTF-8</code>
         */
        final Charset CHARSET_UTF8 = Charset.forName("UTF-8"); //$NON-NLS-1$
        resetStream(inputStream);

        StringBuilder fileData = new StringBuilder(10000);
        Reader reader = new BufferedReader(new InputStreamReader(inputStream,
CHARSET_UTF8));
        try {
            char[] buf = new char[1024];
            int numRead = 0;
            while ((numRead = reader.read(buf)) != -1) {
                fileData.append(buf, 0, numRead);
         }
      } finally {
         try {
            reader.close();
         } catch (IOException e) { //$JL-EXC$ //NOPMD
         }
      }
      return fileData.toString();
   }
   private static void resetStream(InputStream inputStream) {
      try {
         inputStream.reset();
      } catch (IOException e) { //$JL-EXC$ //NOPMD
         // reset() yields IOException in some InputStream implementations
         // cannot check this in advance
      }
   }

}
```

NOTE

So that you can avoid warnings due to discouraged access, the annotation @SuppressWarnings("restriction") is added because this is not productive code.

3.4.5.1 Creating a Test File

Create the testflight.xml file as a new file in package `com.mycompany.adt.##.flight.data.` Use the following content:

```
<?xml version="1.0" encoding="UTF-8"?>
<flights:flight xmlns:flights="http://www.sap.com/adt/examples/flights" carrierid="AA"
connectionid="0017" date="2010-09-16" price="500.00" currency="USD" planetype="747-
400" seatsmax="385" seatsoccupied="375"></flights:flight>
```

Then you can run a unit test.

3.4.6 Getting Your Test Running

The code will compile, but will not work yet.

3.4.6.1 Implementing the Serialize and Deserialize Method

Here is the productive code. Use it to replace the corresponding methods in the existing implementation:

```
public class FlightDataContentHandler implements IContentHandler<FlightData> {


...
   private static final String FLIGHTS_ELEMENT = "flights";
   private static final String ATTR_SEATSOCCUPIED = "seatsoccupied";
   private static final String ATTR_SEATSMAX = "seatsmax";
   private static final String ATTR_PLANETYPE = "planetype";
   private static final String ATTR_CURRENCY = "currency";
   private static final String ATTR_PRICE = "price";
   private static final String ATTR_DATE = "date";
   private static final String ATTR_CONNECTIONID = "connectionid";
   private static final String ATTR_CARRIER_ID = "carrierid";
   private static final String FLIGHT_ELEMENT = "flight";
   private static final String HTTP_NAMESPACE =
"http://www.sap.com/adt/examples/flights";
   protected final AdtStaxContentHandlerUtility utility = new
AdtStaxContentHandlerUtility();

         @Override
   public FlightData deserialize(IMessageBody body, Class<? extends FlightData>
dataType) {
      XMLStreamReader xsr = null;
      try {
         xsr = this.utility.getXMLStreamReader(body);

         FlightData flightData = new FlightData();

         for (int event = xsr.next(); event != XMLStreamReader.END_DOCUMENT; event =
xsr.next()) {
            switch (event) {
            case XMLStreamReader.START_ELEMENT:
               if (HTTP_NAMESPACE.equals(xsr.getNamespaceURI()) &&
FLIGHT_ELEMENT.equals(xsr.getLocalName())) {
                  flightData.setCarrierId(getFlightAttribute(xsr, ATTR_CARRIER_ID));
                  flightData.setConnectionId(getFlightAttribute(xsr,
ATTR_CONNECTIONID));
                  flightData.setFlightDate(getFlightAttribute(xsr, ATTR_DATE));
                  flightData.setPrice(getFlightAttribute(xsr, ATTR_PRICE));
                  flightData.setCurrency(getFlightAttribute(xsr, ATTR_CURRENCY));
                  flightData.setPlaneType(getFlightAttribute(xsr, ATTR_PLANETYPE));
                  flightData.setSeatsMax(getFlightAttribute(xsr, ATTR_SEATSMAX));
                  flightData.setSeatsOccupied(getFlightAttribute(xsr,
ATTR_SEATSOCCUPIED));

               }
               break;
            }
         }

         return flightData;

      } catch (XMLStreamException e) {
         throw new ContentHandlerException(e.getMessage(), e);
      } catch (NumberFormatException e) {
         throw new ContentHandlerException(e.getMessage(), e);
      } finally {
         if (xsr != null) {
            this.utility.closeXMLStreamReader(xsr);
         }
      }
   }

   private String getFlightAttribute(XMLStreamReader xsr, String attributeName) {
      String attributeValue = xsr.getAttributeValue(null, attributeName);
      if (attributeValue == null) {
         throw new ContentHandlerException("Attribute " + attributeName + " not
set");
      }
      return attributeValue;
   }

   @Override
   public IMessageBody serialize(FlightData dataObject, Charset charset) {

      XMLStreamWriter xsw = null;

      try {
         xsw = this.utility.getXMLStreamWriterAndStartDocument(charset,
AdtStaxContentHandlerUtility.XML_VERSION_1_0);
         xsw.setPrefix(FLIGHTS_ELEMENT, HTTP_NAMESPACE);
         xsw.writeStartElement(HTTP_NAMESPACE, FLIGHT_ELEMENT);
         xsw.writeNamespace(FLIGHTS_ELEMENT, HTTP_NAMESPACE);
         xsw.writeAttribute(ATTR_CARRIER_ID, dataObject.getCarrierId());
         xsw.writeAttribute(ATTR_CONNECTIONID, dataObject.getConnectionId());
         xsw.writeAttribute(ATTR_DATE, dataObject.getFlightDate());
         xsw.writeAttribute(ATTR_PRICE, dataObject.getPrice());
         xsw.writeAttribute(ATTR_CURRENCY, dataObject.getCurrency());
         xsw.writeAttribute(ATTR_PLANETYPE, dataObject.getPlaneType());
         xsw.writeAttribute(ATTR_SEATSMAX, dataObject.getSeatsMax());
         xsw.writeAttribute(ATTR_SEATSOCCUPIED, dataObject.getSeatsOccupied());
         xsw.writeEndElement();
         xsw.writeEndDocument();
      } catch (XMLStreamException e) {
         throw new ContentHandlerException(null, e);
      } finally {
         this.utility.closeXMLStreamWriter(xsw);
      }
      return this.utility.createMessageBody(this.getSupportedContentType());

}
```

3.4.6.2 Creating a Utility Class

This example makes use of a parser utility that needs to be copied into the data package. Just copy the code below and put it below the data package.

NOTE

The example uses a utility class `AdtStaxContentHandlerUtility` that is not released. It is recommended that you copy the following code as a new class into package `com.mycompany.adt.##.flight.data`:

```
/**
```

```
 *
 */
import java.io.ByteArrayOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.io.StringReader;
import java.nio.charset.Charset;

import javax.xml.stream.XMLInputFactory;
import javax.xml.stream.XMLOutputFactory;
import javax.xml.stream.XMLStreamException;
import javax.xml.stream.XMLStreamReader;
import javax.xml.stream.XMLStreamWriter;

import com.sap.adt.communication.content.ContentHandlerException;
import com.sap.adt.communication.message.ByteArrayMessageBody;
import com.sap.adt.communication.message.IMessageBody;


public class AdtStaxContentHandlerUtility {
   /**
    * Common constant for XML version 1.0.
    */
   public static final String XML_VERSION_1_0 = "1.0"; //$NON-NLS-1$

   /**
    * Common constant for illegal (empty) argument exception messages.
    */
   public static final String NO_CONTENT = "no content"; //$NON-NLS-1$
   /**
    * Common constant for default character set: UTF-8, platform-neutral.
    */

   public static final Charset DEFAULT_CHARSET = Charset.forName("UTF-8"); //$NON-NLS-
1$

   private ByteArrayOutputStream stream = null;

   /**
    * Checks if <tt>dataObject</tt> is not <tt>null</tt>.
    *
    * @param dataObject
    *            object to be checked.
    * @throws IllegalArgumentException
    *             if object is <tt>null</tt>.
    */
   public void serializeCheck(Object dataObject) {
       if (dataObject == null) {
           throw new IllegalArgumentException();
       }
   }

   /**
    * Returns the {@link #DEFAULT_CHARSET} if <tt>charset</tt> is <tt>null</tt>
    * .
    *
    * @param charset
    *            character set to be checked.
    * @return {@link #DEFAULT_CHARSET} if <tt>charset</tt> is <tt>null</tt> or
    *         <tt>charset</tt> otherwise.
    */
   public Charset checkCharsetNotNull(Charset charset) {
       if (charset == null) {
           charset = DEFAULT_CHARSET;
       }
       return charset;
   }

   /**
    * Creates a new XMLStreamWriter and starts an empty XML document.
    *
    * @param charset
    *            character set to be used or {@link #DEFAULT_CHARSET} if
    *            <tt>charset</tt> is <tt>null</tt>.
    * @param version
    *            XML version to be used for the document. See
    *            {@link #XML_VERSION_1_0}.
    * @returna new XMLStreamWriter with started document.
    * @throws XMLStreamException
    *             internal StAX exception.
    * @see {@link #getXMLStreamWriter(OutputStream, Charset)}
    */
   public XMLStreamWriter getXMLStreamWriterAndStartDocument(Charset charset, String
version) throws XMLStreamException {
       this.stream = new ByteArrayOutputStream();

       charset = this.checkCharsetNotNull(charset);

       XMLStreamWriter xsw = this.getXMLStreamWriter(this.stream, charset);

       xsw.writeStartDocument(charset.name(), version);

       return xsw;
   }
   /**
    * Creates a new XMLStreamWriter for a given stream.
    *
    * @param stream
    *            output stream to be used.
    * @param charset
    *            character set to be used.
    * @return new empty XMLStreamWriter.
    * @throws XMLStreamException
    *             internal StAX exception.
    */
   public XMLStreamWriter getXMLStreamWriter(OutputStream stream, Charset charset)
throws XMLStreamException {
       final XMLOutputFactory xof = XMLOutputFactory.newInstance();

       XMLStreamWriter xsw = xof.createXMLStreamWriter(stream, charset.name());

       return xsw;
   }

   /**
    * Creates a XMLStreamReader for a given message body.
    *
    * @param body
    *            message body to be opened for reading.
    * @return new XMLStreamReader for <tt>body</tt>.
    * @throws XMLStreamException
    *             internal StAX exception.
    */

   public XMLStreamReader getXMLStreamReader(IMessageBody body) throws
XMLStreamException {
       if (body == null || body.getContentLength() == 0) {
           throw new IllegalArgumentException(NO_CONTENT);
       }

       try {
           final InputStream is = body.getContent();
           return this.getXMLStreamReader(is);
       } catch (IOException e) {
           throw new ContentHandlerException(e.getMessage(), e);
       }
   }

   /**
    * Creates a XMLStreamReader for a given input stream
    *
    * @param stream
    *             stream to be used for reading
    * @return new XMLStreamReader for <tt>stream</tt>.
    * @throws XMLStreamException
    *              internal StAX exception
    */
   public XMLStreamReader getXMLStreamReader(InputStream stream) throws
XMLStreamException {
       if (stream == null) {
           throw new IllegalArgumentException(NO_CONTENT);
       }
       final XMLInputFactory xif = XMLInputFactory.newInstance();

       return xif.createXMLStreamReader(stream);
   }



   /**
    * Creates a XMLStreamReader for a given message body.
    *
    * @param content
    *             the string to read
    * @return new XMLStreamReader for <tt>content</tt>.
    * @throws XMLStreamException
    *              internal StAX exception.
    */
   public XMLStreamReader getXMLStreamReader(String content) throws XMLStreamException
{
       if (content == null || content.length() == 0) {
           throw new IllegalArgumentException(NO_CONTENT);
       }

       final XMLInputFactory xif = XMLInputFactory.newInstance();

       return xif.createXMLStreamReader(new StringReader(content));
   }

   /**
    * Gracefully closes a given XMLStreamWriter.
    *
    * @param xsw
    *             a XMLStreamWriter.
    */
   public void closeXMLStreamWriter(XMLStreamWriter xsw) {
       if (xsw != null) {
           try {
               xsw.close();
           } catch (XMLStreamException e) { //$JL-EXC$ //NOPMD
           }
```

```
      }
      try {
         this.stream.close();
      } catch (IOException e) { //$JL-EXC$ //NOPMD
      }
   }

   /**
    * Gracefully closes a given XMLStreamReader.
    *
    * @param xsr
    *            a XMLStreamReader.
    */
   public void closeXMLStreamReader(XMLStreamReader xsr) {
      if (xsr != null) {
         try {
            xsr.close();
         } catch (XMLStreamException e) { //$JL-EXC$ //NOPMD
         }
      }
   }



   /**
    * Creates a message body from the current state of this
    * AdtStaxContentHandlerUtility instance.
    *
    * @param supportedContentType
    *            content MIME type of the message body.
    * @return newly created message body.
    */
   public IMessageBody createMessageBody(String supportedContentType) {
      byte[] data = this.stream.toByteArray();
      return new ByteArrayMessageBody(supportedContentType, data);
   }
}
```

3.4.7 Running the Unit Test

NOTE

At the end, the test should be green.

1. Add the `Content Handler` to the `Simple Backend Call`
2. Modify the junit test that is created and add the content handler.
3. In the class `SampleHandler`, replace the following source code in the method execute(ExecutionEvent event):

```
...
public class SampleHandler extends AbstractHandler {
...
public Object execute(ExecutionEvent event) throws ExecutionException {
        // Create resource factory
        IRestResourceFactory restResourceFactory = AdtRestResourceFactory
                .createRestResourceFactory();
        // Get available projects in the workspace
        IProject[] abapProjects = AdtProjectServiceFactory
                .createProjectService().getAvailableAbapProjects();
        // Use the first project in the workspace for the demo
        // to keep the example simple
        IAbapProject abapProject = (IAbapProject) abapProjects[0]
                .getAdapter(IAbapProject.class);
        // Trigger logon dialog if necessary
        AdtLogonServiceUIFactory.createLogonServiceUI().ensureLoggedOn(
                abapProject.getDestinationData(),
                PlatformUI.getWorkbench().getProgressService());
        // Create REST resource for given destination and URI
        String destination = abapProject.getDestinationId();    URI flightUri =
URI.create(SAMPLE_FLIGHT_RESOURCE_URI);
        IRestResource flightResource = restResourceFactory
                .createResourceWithStatelessSession(flightUri, destination);
        // Create and add the content handler:
        FlightDataContentHandler flightHandler = new FlightDataContentHandler();
        flightResource.addContentHandler(flightHandler);
        try {
            // Trigger GET request on resource data
            FlightData flightDataRead = flightResource.get(null,
                    FlightData.class);
            // Display some flight data in a popup window
            displayFlight(flightDataRead);
        } catch (ResourceNotFoundException e) {
            displayError("No flight data found");
        } catch (RuntimeException e) {
            // Display any kind of other error
            displayError(e.getMessage());
        }
        return null;
    }
    /*
     * Display the flight data which was read from the backend
     */
    private void displayFlight(FlightData flightDataRead) {
        String messageText = "Flight data found: "
                + flightDataRead.getCarrierId() + " "
                + flightDataRead.getConnectionId() + " Seats booked: "
                + flightDataRead.getSeatsOccupied() + "/"
                + flightDataRead.getSeatsMax();
        openDialogWindow(messageText, "Flight Lookup");
    }
```

```
}
```

NOTE

You can see that we have registered the content handler using the `addContentHandler` method of the resource. The GET request is now able to return an instance of `FlightData` instead of the `IRestResponse`. The code should compile.

3.5 Usage of the Discovery to Get the Flight URI

So far, we have defined the flight URI as a constant in the `SampleHandler` class, but not yet used the discovery service to get the URI from the back end using URI templates.

We will add some code now to call the discovery service in order to get the URI for a given flight key.

1. Add the following constants in the class definition:

```
   private static final String MYCOMPANY_DISCOVERY = "/mycompany/discovery";
   private static final String URI_PARAMETER_FLIGHT_DATE = "flight_date";
   private static final String URI_PARAMETER_CONNECTION_ID = "connection_id";
   private static final String URI_PARAMETER_CARRIER_ID = "carrier_id";
   private static final String FLIGHT_RELATION = 
"http://www.mycompany.com/adt/relations/examples/##/flights/singleaccess";
   private static final String SCHEME = 
"http://www.mycompany.com/adt/categories/examples/##/flights";
   private static final String TERM = "flights";
```

2. Add the method below:

```
   /*
    * Use Discovery Service to get the flight URI
    */
   private URI getFlightURI(String destination, String carrierId, String connectionId, 
String flightDate)
   {
      IAdtDiscovery discovery = AdtDiscoveryFactory.createDiscovery(destination, 
URI.create(MYCOMPANY_DISCOVERY));   // Get collection member by use of scheme and term
      IAdtDiscoveryCollectionMember collectionMember = 
discovery.getCollectionMember(SCHEME, TERM, new NullProgressMonitor());
      IAdtTemplateLink templateLink = 
collectionMember.getTemplateLink(FLIGHT_RELATION);
      IAdtUriTemplate uriTemplate = templateLink.getUriTemplate();
      String uri = uriTemplate.set(URI_PARAMETER_CARRIER_ID, 
carrierId).set(URI_PARAMETER_CONNECTION_ID, 
connectionId).set(URI_PARAMETER_FLIGHT_DATE, flightDate).expand();
      return URI.create(uri);
   }
```

3. Replace the existing assignment of the flight URI by a method call.
4. Run the `SampleHandler` on the target platform.

NOTE

The discovery service does not only provide a static collection URI for flights but also a template link that can be instantiated by the client using parameters. You can see that it is easy to get and instantiate the link simply by replacing the values of the URI parameters by the corresponding values.

www.sap.com/contactsap