Tutorial for Creating Resources in Java - Client

Overview

- Overview
- 1. Preparation
- 2. Creation of Eclipse Plug-ins
    - 2.1 The flight plugin
    - 2.2 The plugin fragment for unit tests
- 3. Create an integration test for testing the backend resource
- 4. Deserializing the request response
    - 4.1 Creating a class for FlightData
    - 4.2 Create a content handler
    - 4.3 Creating a unit test for the content handler
    - 4.4 Add an XML File with Test Data
- 5. Add the Content Handler to the Integration Test
- 6. Final Words you Should not Miss
- 7. Further References

1. Preparation

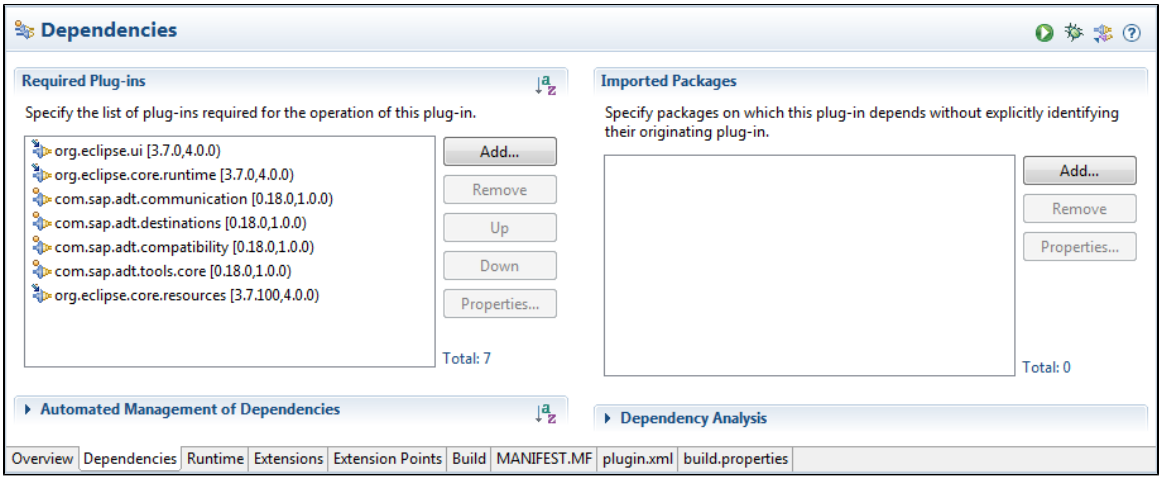
The complete code of the example can be downloaded at
Perforce. Please use the code for further reference. In the following, we will focus on a quite simplified end-to-end example.

2. Creation of Eclipse Plug-ins

At first, we will create Plug-ins for the sample code and a separate Plug-In fragment for tests.
It is recommended to switch to the *Plug-In Development* perspective.

2.1 The flight plugin

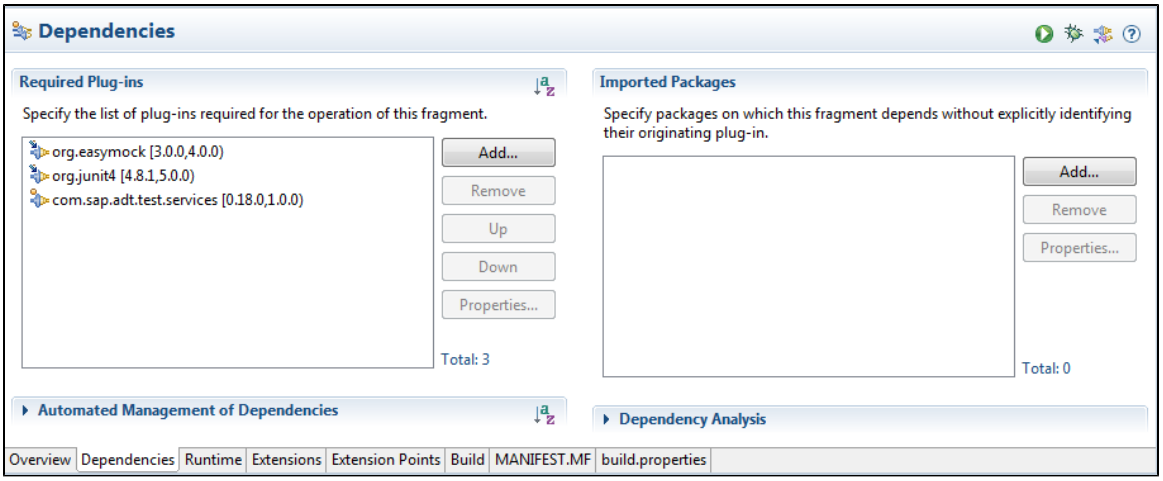
Create a new Eclipse plug-in *com.sap.adt.XX.examples.flight* (it's recommended to use your own namespace, so that you can also import the sample solution in parallel). Configure the plug-ins dependencies in the *MANIFEST.MF* file as follows:



Within the plugin, you get a package 'com.sap.adt.XX.examples.flight'.

2.2 The plugin fragment for unit tests

We will put all tests into a separate plugin fragment. **Create a new Eclipse plug-in fragment *com.sap.adt.XX.examples.flight.test*. Enter *com.sap.adt.XX.examples.flight* as host plugin name.** For the fragment which inherits the dependencies from the host plug-in, we need to define the following *additional* dependencies for the tests:



easymock is needed for UI testing which is not part of this example, so you do not have to add the dependency.
Within the plugin, you get a package 'com.sap.adt.XX.examples.flight.test'.

3. Create an integration test for testing the backend resource

Go to the test plugin and create a new Java class with the name *ResourceExampleTests* and add a new test method *invokeGetMethodOnRestResource()*. Now you should have the following coding:

```
public class ResourceExampleTests {
 @Test
 public void invokeGetMethodOnRestResource() {

 }
}
```

Next, we have to register a JCo destination provider with our test helper. It will later be used to connect to the backend with the default connection settings to unit tests. So we put this code in a class with the *@RunWithDestination* annotation. The code looks like this:

```
@RunWith(AdtIntegrationTest.class)
@RunWithDestination(DestinationTestUtil.BAP)
public class ResourceExampleTests {

 @Test
 public void invokeGetMethodOnRestResource() {

 }
}
```

Note: To organize your imports, press *Ctrl+Shift+O*.

Now, it is time to write the code for invoking the method on the backend resource controller. This coding consists of 4 steps:

1. Create a stateless connection for the backend.
2. Create the URI (id) for the resource which you want to manipulate
3. Create a rest resource for the system session and URI. This object offers HTTP methods to communicate with the backend.
4. Call the GET method on the resource.

In the end the necessary code looks like this:

```
// (imports skipped)
@SuppressWarnings("restriction")
@RunWith(AdtIntegrationTest.class)
@RunWithDestination(DestinationTestUtil.BAP)
public class ResourceExampleTests
{
  @Test
  public void invokeGetMethodOnRestResource() throws Exception
  {
    // create resource factory
IRestResourceFactory restResourceFactory = AdtCommunicationFrameworkPlugin
  .getDefault().getFactory().createRestResourceFactory();
    // get destination from test utility
String destination = DestinationTestUtil.getInstance()
    .getDestinationId();
    // set URI to a single flight resource
URI flightUri = new URI(
     "/sap/bc/adt/examples/restflights/flights/AA/0017/20100916");
    IRestResource flightResource = restResourceFactory
   .createResourceWithStatelessSession(flightUri, destination);
    // check if resource in the backend exists (HTTP response code OK = 200)
IRestResponse response = flightResource.get(null, IRestResponse.class);
    assertEquals(HttpURLConnection.HTTP_OK,response.getStatus());
  }
}
```

The code shall compile now. Please run the unit test as a JUnit-Plugin-Test. If it fails, please check the table entries of the database table sflight in the backend. Adjust the URI so that an existing table entry will be selected.
This code does not yet deserialize the response body which contains the flight XML data because we did not yet provide a content handler and a data object which represents a single flight.

4. Deserializing the request response

In the backend, we have implemented a simple transformation which is used to deserialize and serialize the content data of the request or response, respectively.
At the client side, we will use a specific content handler which is able to serialize and deserialize the xml flight data. First, we will define a simple FlightData class which represents a single flight instance.

4.1 Creating a class for FlightData

Go to the plugin *com.sap.adt.XX.examples.flight* and create a new package *com.sap.adt.XX.examples.flight.data*.
Create a new class FlightData and copy the code below.

```
// (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;
 }
}
```

You will get compiler warnings for the unused private attributes. Use the eclipse source code editor functionality (ALT-SHIFT-S) to add the following methods automatically:
Add setters and getters for all attributes
Add a toString() method
Add a hashCode() method
Add an equals() method

The warnings will disappear.

We will now start to create a content handler with a corresponding unit test.

4.2 Create a content handler

Go to the plugin *com.sap.adt.XX.examples.flight* and create a new class FlightDataContentHandler in package com.sap.adt.XX.examples.flight.data which implements the interface IContentHandler<FlightData>. Create stubs for all interface methods. The methods getSupportedContentType and getSupportedDataType can be implemented easily (see below). For now, we leave the serialize and deserialize methods unimplemented.

```
// (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;
 }
}
```

4.3 Creating a unit test for the content handler

Go to the plugin fragment *com.sap.adt.XX.examples.flight.test* and create a new package com.sap.adt.XX.examples.flight.data. 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. **In the new package, create class TestsFlightDataContentHandler.**

```
@SuppressWarnings("restriction")
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());
 }

 @SuppressWarnings("restriction")
 @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 = FileUtils.toString(expectedBody.getContent()).replaceAll("[\n\r ]", "");

  // Call the serialization to get the actual string:
IMessageBody actBody = contentHandler.serialize(flightData, null);
  String actualXml = FileUtils.toString(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);
 }

}
```

In order to avoid some warnings due to discouraged access, we have added the annotation @SuppressWarnings("restriction") because this is not productive code.

The code will compile but does not yet work. We will need a sample XML file.

4.4 Add an XML File with Test Data

In the same package, create a new file "testflight.xml" with 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" />
```

Execute the unit tests as a JUnit Test ("Debug as JUnit Test"). Of course, both tests will fail. Feel free to implement the serialize and deserialize methods now. Here is the code:

```
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 carrierId = xsr.getAttributeValue(null, attributeName);
   if (carrierId == null) {
    throw new ContentHandlerException("Attribute" + attributeName + "not set");
   }
   return carrierId;
  }

  @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());
```

```
}
```

Hopefully, your test result is OK now!

5. Add the Content Handler to the Integration Test

Now, it is easy to modify the integration test we have created above and add the content handler. **Modify the last lines of the code in the test class ResourceExampleTests in the following way:**

```
...
FlightDataContentHandler flightHandler = new FlightDataContentHandler();
flightResource.addContentHandler(flightHandler);
// check if resource in the backend exists (HTTP response code OK = 200)
FlightData flightDataRead = flightResource.get(null, FlightData.class);
assertEquals("AA", flightDataRead.getCarrierId());
assertEquals("0017", flightDataRead.getConnectionId());
```

You can see that we have registered the content handler by use of the addContentHandler method of the resource. The get is now able to return an instance of FlightData instead of the IRestResponse.
The code should compile.
Start the test as a JUnit Plug-in test in the debug mode.
The test result should be green.

6. Final Words you Should not Miss

Congratulations, you successfully implemented your first resource controller and invoked it from Eclipse!

The example described here is just for training. When writing client code, you should follow some basic rules:

- **The client should never create any URI's or make any assumptions how they look like!** Unlike in this example, please use backend services to get the URI to access the resource. E.g., the discovery service shall be used to get the URI to the collection /sap/bc/adt/examples/restflights/flights as an entry URI to execute an initial flight query. Please take a look at the complete example class TestsIntegrationFlights.
- For productive code, of course, you need to **declare Plug-in dependencies correctly** rather than ignoring warnings
- Try to **avoid string literals and use constants** in your test classes, too.

We are constantly improving our documentation. Please let us know if you have any questions or find any errors!

7. Further References

As mentioned under *Preparation*, you can take a look at the example code in the adt tools plugin in packages com.sap.adt.examples.flight and com.sap.adt.examples.flight.test which can be downloaded from the Perforce server.

Example Class	Description
FlightManager	The class implements a simple example view which can be displayed by use of CTRL-3 "flights".
CarrierData	Data model for carriers (single carrier). Please note that the carrier data example is still based on asXML. For new resources, asXML must not be used any more
CarrierDataTable	Data model for carriers (collection). See above!
FlightData	Data model for flights. Corresponds with the example described here
FlightDataContentHandler	Content handler for flight data which is also part of the example above
FlightDataTable	Data model for a list of flights. It is used by the integration test for the GET method of the collection resource
FlightDataTableContentHandler	Content handler for a list of flights

TestsIntegrationCarriers	Integration test for carrier query using the CarrierData and CarrierDataTable
TestsIntegrationFlights	Integration test for flights. Example with create, read, update, delete (CRUD) and query on the collection resource
TestsUnitFlightDataContentHandler	Unit test for flight data content handler
TestsUnitFlightDataTableContentHandler	Unit test for flight data table content handler
testflight.xml	Sample XML for content handler unit test (single flight)
testflights.xml	Sample XML for content handler unit test (flight list)