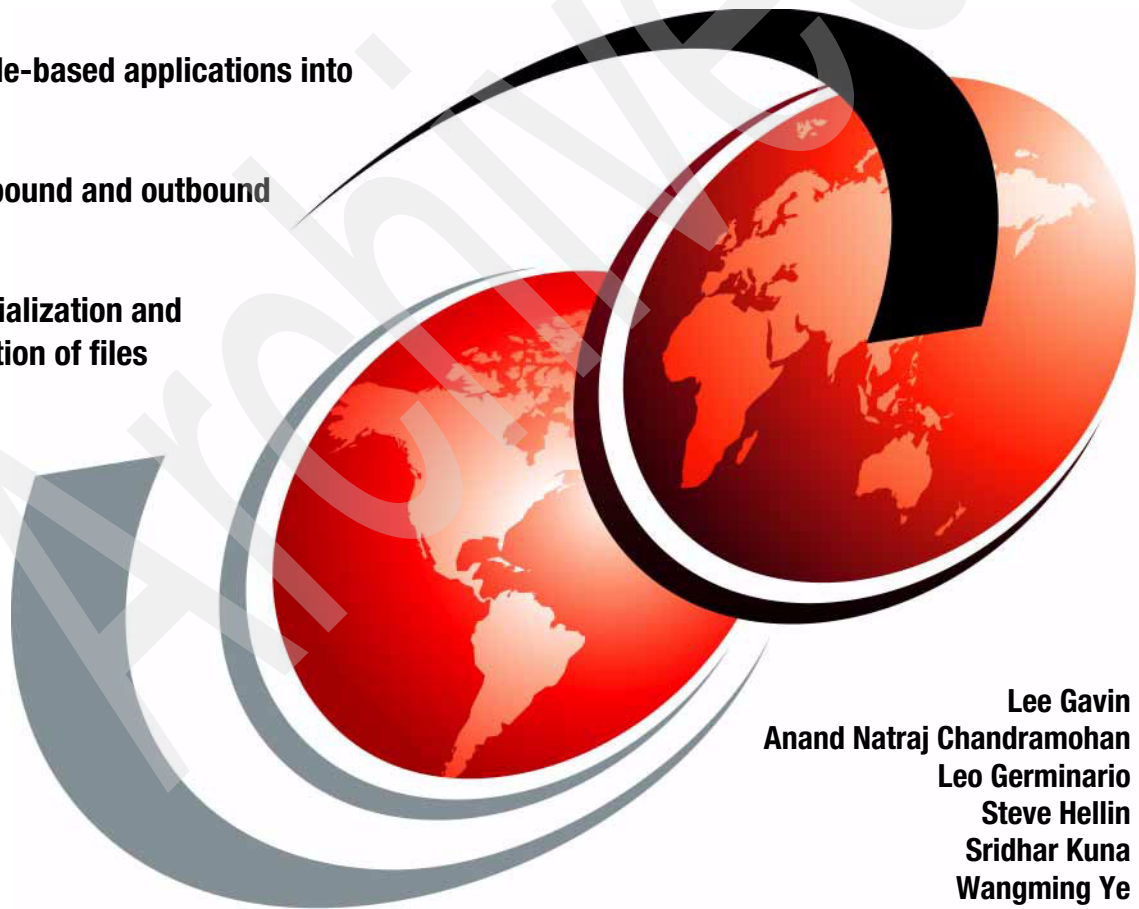


# A Simple Example: Using the WebSphere Adapter for Flat Files

Integrate file-based applications into  
your SOA

Process inbound and outbound  
files

Explore serialization and  
deserialization of files



Lee Gavin  
Anand Natraj Chandramohan  
Leo Germinario  
Steve Hellin  
Sridhar Kuna  
Wangming Ye





International Technical Support Organization

## **A Simple Example: Using the WebSphere Adapter for Flat Files**

December 2006

Archived

**Note:** Before using this information and the product it supports, read the information in “Notices” on page v.

## **First Edition (December 2006)**

This edition applies to Version 6 of IBM WebSphere Adapter for Flat Files.

© Copyright International Business Machines Corporation 2006. All rights reserved.

Note to U.S. Government Users Restricted Rights -- Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

# Contents

<b>Notices</b> .....	v
Trademarks .....	vi
<b>Preface</b> .....	vii
The team that wrote this Redpaper .....	viii
Become a published author .....	ix
Comments welcome .....	ix
<b>Chapter 1. WebSphere Adapter for Flat Files</b> .....	1
1.1 Differences between adapters .....	2
1.1.1 Architectural differences .....	2
1.1.2 Functional differences .....	2
<b>Chapter 2. Process outbound files</b> .....	15
2.1 Setup and initial configuration .....	16
2.2 Create External_Maintenance module .....	16
2.3 Enterprise service discovery for outbound artifacts .....	19
2.4 Create application-specific business objects .....	27
2.5 Create interface map .....	31
2.6 Build the outbound component of External_Maintenance .....	46
2.7 Test the outbound file creation .....	50
<b>Chapter 3. Process inbound files</b> .....	57
3.1 Enterprise service discovery for inbound artifacts .....	58
3.2 Create the interface and interface map .....	64
3.3 Build the inbound processing of External_Maintenance .....	77
3.4 Test the inbound file processing .....	78
3.4.1 Testing using a Java component .....	78
3.4.2 Test by attaching to the module .....	84
<b>Appendix A. Additional material</b> .....	87
Locating the Web material .....	87
Using the Web material .....	87
System requirements for downloading the Web material .....	88
How to use the Web material .....	88



# Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:  
*IBM Director of Licensing, IBM Corporation, North Castle Drive, Armonk, NY 10504-1785 U.S.A.*

**The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:** INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

## COPYRIGHT LICENSE:


This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

# Trademarks

The following terms are trademarks of the International Business Machines Corporation in the United States, other countries, or both:

@server®

@server®

Redbooks (logo) ™

ibm.com®

IBM®

Redbooks™

WebSphere®

The following terms are trademarks of other companies:

EJB, Java, J2EE, and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Expression, Windows, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.



# Preface

The WebSphere® Adapter for Flat Files enables integration with applications that provide a basic flat file interface. This adapter can be used with applications that do not provide programmable or service interfaces, and for which integration at the data tier is impractical. This JCA resource adapter facilitates the bidirectional exchange of any type of text or binary file, and provides the ability to:

- ▶ Parse and serialize native data formats.
- ▶ Write to new files.
- ▶ Append to or overwrite existing files.
- ▶ Poll a directory for new files to deliver to the server.
- ▶ List, retrieve, and delete target files in a directory.

The WebSphere Adapters are configured within WebSphere Integration Developer for deployment with WebSphere Process Server and WebSphere ESB. The adapter implements EMD to generate service descriptions and configuration artifacts.

WebSphere Business Integration Adapter for Flat Files is available for use with WebSphere Message Broker and other integration servers as well as WebSphere Process Server and WebSphere Enterprise Service Bus.

In this IBM Redpaper we show a simple example of how to configure the WebSphere Adapter for Flat Files to enable the reading and processing of incoming files and the creation of files for outgoing processing.

The examples show the steps involved in mapping data objects to files and vice versa using the mapping tools within WebSphere Integration Developer and also the APIs available for serialization and deserialization of data. We illustrate the use of mapping for testing and comparing values contained in a data object or file and the translation of different data types using the mapping tools.

We do not illustrate the deployment of the adapter into WebSphere Process Server, as this is outlined in great detail in the product documentation and WebSphere Infocenter, nor do we discuss runtime configuration options.

What we show in this Redpaper is a simple example of how you might like to use this adapter.

## The team that wrote this Redpaper

This Redpaper was produced by a team of specialists from around the world working at the International Technical Support Organization, Raleigh Center.

**Lee Gavin** is a Consulting IT Specialist in IBM Software Group. She works in WebSphere Process Integration Technical Sales (TechWorks for South West EMEA). Prior to joining TechWorks she worked at the International Technical Support Organization, Raleigh Center, writing extensively and teaching IBM classes worldwide on all areas of WebSphere Process Integration.

**Anand Natraj Chandramohan** is a Software Engineer in the IBM Software Labs, India, specializing in Information Management.

**Leo Germinario** is an Advisory IT Specialist in the Java™ Technology Center, IBM Italy in Bari, which is organized under the IBM Business Consulting Services. He has five years of experience in J2EE™ Application Integration in several sectors and application areas. He holds a Computer Science degree from the University of Bari, in Italy. His areas of expertise include Data Modelling, Integrated Application Architecture, and J2EE Application Integration Development.

**Steve Hellin** is an IBM Certified IT Architect in WebSphere Technical Sales for the Americas. He has over nine years of systems integration experience, and has been working with WebSphere Business Integration solutions involving WebSphere MQ, WebSphere InterChange Server, and WebSphere Business Integration Adapter for over six years. He is an IBM Certified Deployment Professional for WebSphere InterChange Server. He holds a Bachelor of Science degree in Computer Science from the University of Illinois at Urbana-Champaign.

**Sridhar Kuna** is a Software Engineer at the IBM India Software Labs. He is in WebSphere Business Integration Adapter development and works with partner lab IBM Crossworlds, Burlingame. He began his career with IBM in 2003 and he has almost three years of experience in the EAI field. He holds a Master of Engineering degree in Software Engineering from Birla Institute of Technology, India, and a Bachelor of Technology degree in Electrical and Electronics Engineering from Andhra University, Andhra Pradesh, India. His areas of expertise include WBI and WebSphere Adapters in Diamond environment, SOA Tools, ICS, WebSphere MQ, and J2EE technologies.

**Wangming Ye** is a Consulting IT Specialist in IBM Business Partner Technical Enablement (BPTE) organization, providing developer-to-developer technical support and services for ISVs to integrate solutions on WebSphere family products. Prior to joining BPTE, he was one of the main developers of the WebSphere Content Distribution Framework and he began his IBM career as a

DCE/DFS developer in 1999 following the IBM acquisition of the Transarc Corporation in Pittsburgh. He has authored several papers on the Web services interoperability between J2EE and.NET.

## Become a published author

Join us for a two- to six-week residency program! Help write an IBM Redbook dealing with specific products or solutions, while getting hands-on experience with leading-edge technologies. You'll have the opportunity to team with IBM technical professionals, Business Partners, and Clients.

Your efforts will help increase product acceptance and customer satisfaction. As a bonus, you'll develop a network of contacts in IBM development labs, and increase your productivity and marketability.

Find out more about the residency program, browse the residency index, and apply online at:

[ibm.com/redbooks/residencies.html](http://ibm.com/redbooks/residencies.html)

## Comments welcome

Your comments are important to us!

We want our papers to be as helpful as possible. Send us your comments about this Redpaper or other Redbooks™ in one of the following ways:

- ▶ Use the online **Contact us** review redbook form found at:

[ibm.com/redbooks](http://ibm.com/redbooks)

- ▶ Send your comments in an e-mail to:

[redbooks@us.ibm.com](mailto:redbooks@us.ibm.com)

- ▶ Mail your comments to:

IBM® Corporation, International Technical Support Organization  
Dept. HYTD Mail Station P099  
2455 South Road  
Poughkeepsie, NY 12601-5400

Archived

# WebSphere Adapter for Flat Files

The IBM WebSphere Adapter for Flat Files enables integration with applications that provide a basic flat file interface. This adapter can be used with applications that do not provide programmable or service interfaces, and for which integration at the data tier is impractical.

This JCA adapter includes the following features:

- ▶ Facilitates the bidirectional exchange of any type of text or binary file
- ▶ Provides the ability to write to new files and append to or overwrite existing files
- ▶ Polls a directory for new files to deliver to the configured endpoint
- ▶ Retrieves the contents of target files, checks for the existence of files, lists the files in a directory, and deletes target files

The adapter implements enterprise service discovery to facilitate the generation of service descriptions and configuration artifacts for the adapter.

## 1.1 Differences between adapters

There are two main differences between the previous version of the WebSphere Business Integration Adapter for JText and the new WebSphere Adapter for Flat Files. The architectural and functional differences are listed below.

### 1.1.1 Architectural differences

The architectural differences are:

- ▶ Protocol split - With the WebSphere Business Integration Adapter for JText, the adapter combined the functionalities of both the local file operations and FTP file operations. With the WebSphere Adapter for Flat Files, the adapter only deals with local file operations. This functional segregation makes the adapter more maintainable and more specialized for protocol-specific operations.
- ▶ Perception of an inbound event - With the WebSphere Business Integration Adapter for JText, individual data records in the event file were considered to be events and the adapter parsed the event files to extract the record content. With the WebSphere Adapter for Flat Files, the entire file is considered an event and the adapter does not parse the event file. This architecture was chosen in order to separate the different tasks involved with both file handling and data transformation. File handling deals with the entire file, which involves detecting the arrival of event files for inbound processing, reading the entire content of the file, and writing the file content for outbound processing, while data transformation involves file parsing and extracting data records out of the file. This split between protocol handling and data transformation makes the individual components more reusable and maintainable.

### 1.1.2 Functional differences

The functional differences are:

- ▶ More outbound processing functionalities - The WebSphere Adapter for Flat Files supports more operations for outbound processing compared to the WebSphere Business Integration Adapter for JText. The WebSphere Business Integration Adapter for JText request operations supported only create, append, and overwrite through the appropriate meta-object configuration. While the WebSphere Adapter for Flat Files supports create, append, overwrite, retrieve, delete, exists, and list operations.
- ▶ File splitting feature available for inbound processing - With the WebSphere Adapter for Flat Files, an event file can be delivered to an endpoint in specified chunk sizes.

## Process outbound files

In this chapter we build the outbound file processing example. In our scenario we have external applications that have an XML file-based requirement. Our company has a service-oriented approach to integration, and we would like to be able to integrate the application of our external partners. To this end we use the WebSphere Adapter for Flat Files (also known as the Flat File Adapter).

As no `DataHandler` capabilities (as are used in the WebSphere Business Integration Adapter for JText, for example) are supplied for this JCA adapter, we use a provided programming interface to perform a similar function.

We use the *BOXMLSerializer* API for conversion of business object to XML and XML to business object.

## 2.1 Setup and initial configuration

We have two scenarios where we use the Flat File Adapter:

- ▶ FlatFile Outbound

The adapter receives a BO from the WebSphere Process Server and serializes it into a request XML message (using the BOXMLSerializer API). This request XML message can then be forwarded on to our external application in the correct format.

- ▶ FlatFile Inbound

The adapter receives an XML message from our external application and deserializes it into a BO (using the BOXMLSerializer API). It is then forwarded to WebSphere Process Server.

As part of our company's ongoing effort to implement a service-oriented architecture, we have some core components that have been built and designed. These are listed in Appendix A, "Additional material" on page 87.

It is assumed that you have imported the common components and library for our solution, which are contained in Appendix A, "Additional material" on page 87, and that the Flat File Connector is ready for configuration. Ensure that the common maintenance module is deployed to the server prior to testing.

## 2.2 Create External\_Maintenance module

The installer for the adapter does not deploy or configure the adapter. We must now use the WebSphere Integration Developer to import the RAR file and run the enterprise service discovery wizard to discover the generated artifacts. We start from here. From the Business Integration perspective:

1. Create a new module.
2. Enter External\_Maintenance for the module name.

Click **Finish**. We now must import the Adapter RAR file from the Flat File Adapter installation directory:

1. Select **File** → **Import**.



2. Select **RAR file** and click **Next** (Figure 2-1).

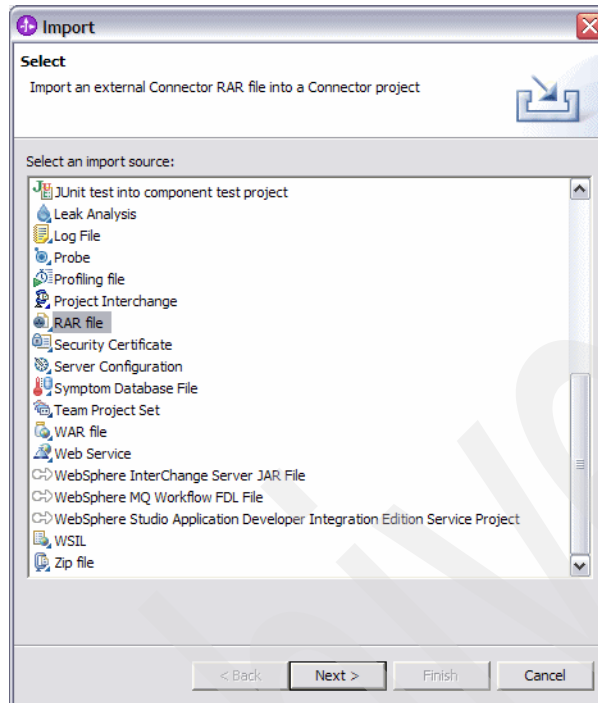


Figure 2-1 Select import type

3. Browse for where you have stored the Adapter RAR file (Figure 2-2). (The connector project name will be inserted automatically once the RAR file is selected.)

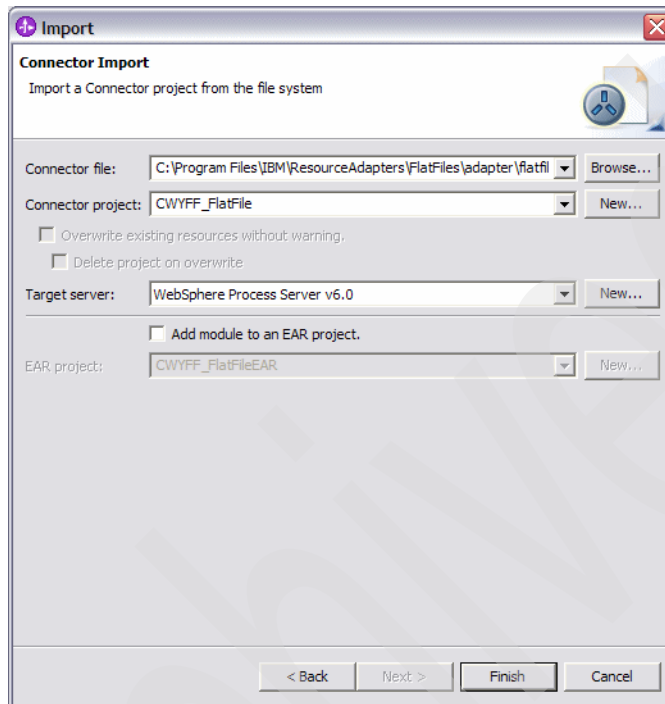


Figure 2-2 Browse for RAR file

4. Click **Finish**.
5. In the Confirm Enablement window (Figure 2-3) click **OK**.

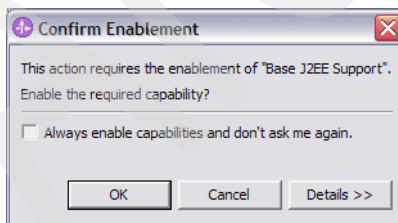


Figure 2-3 Confirm enablement

## 2.3 Enterprise service discovery for outbound artifacts

We now must run the enterprise discovery wizard to discover and import the adapter artifacts for outbound processing:

1. Right-click the **External\_Maintenance** module and select **New** → **Enterprise Service Discovery** (Figure 2-4).

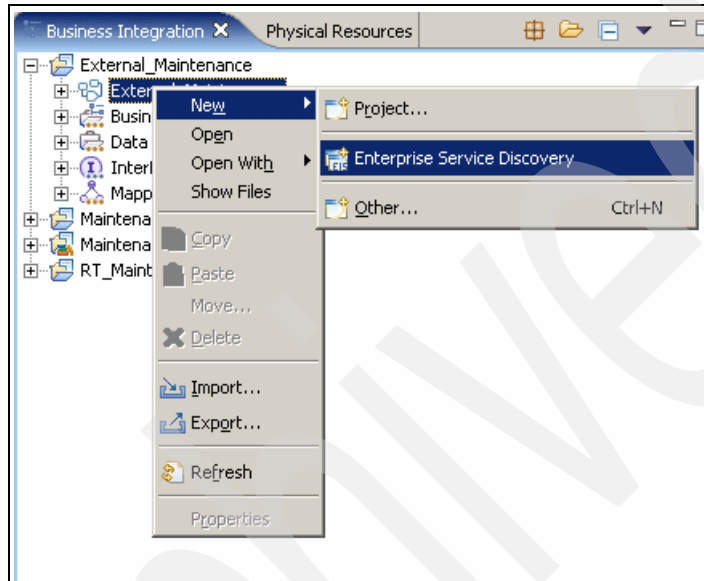


Figure 2-4 Enterprise service discovery

2. Select the Flat File Connector project (**CWYFF\_FlatFile**) and click **Next** (Figure 2-5).

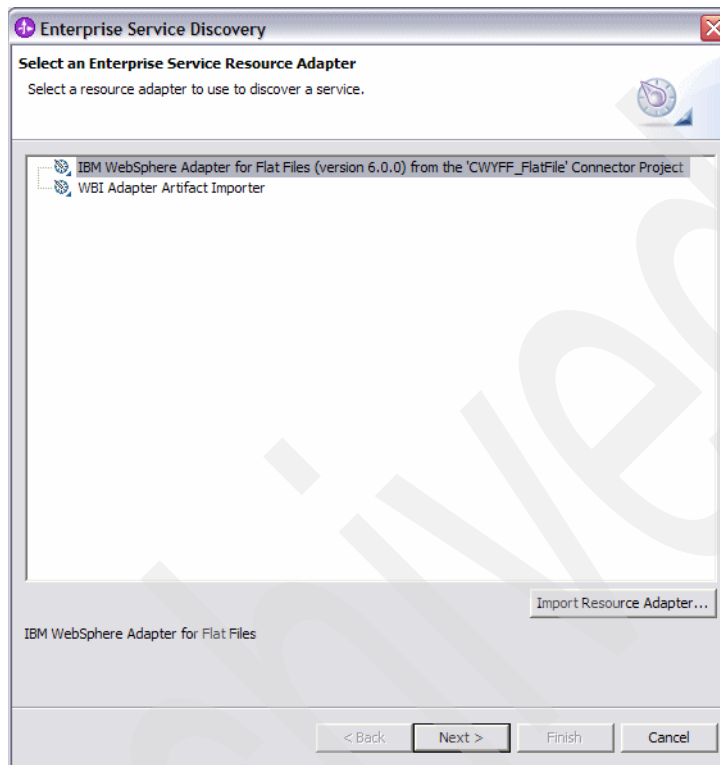


Figure 2-5 Select project

**Note:** If you do not see the flat file in the list, try running a clean build of your projects.

3. From the drop-down list, select **Service Type** → **Outbound**. Click **Next** (Figure 2-6).

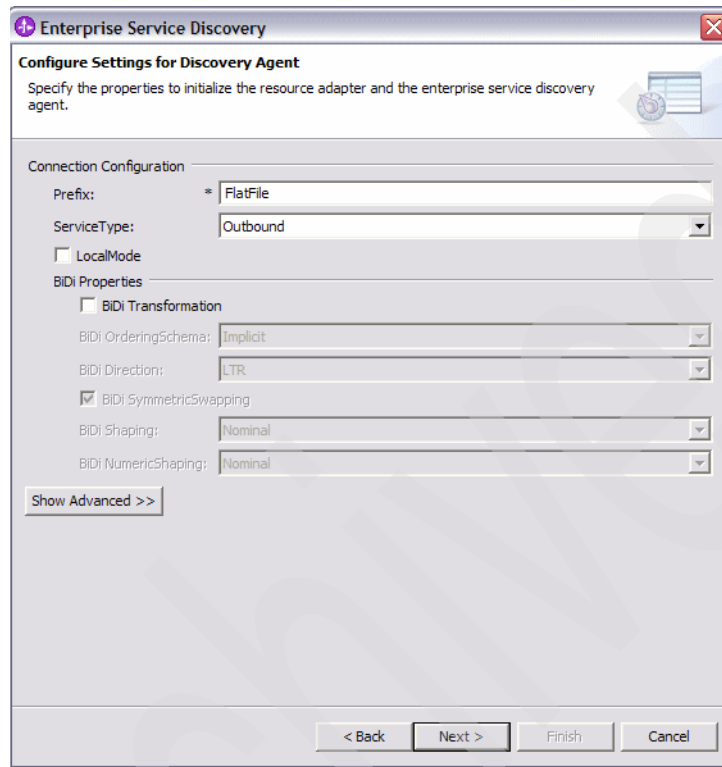


Figure 2-6 Select outbound service type

4. Click **Run Query** (Figure 2-7).

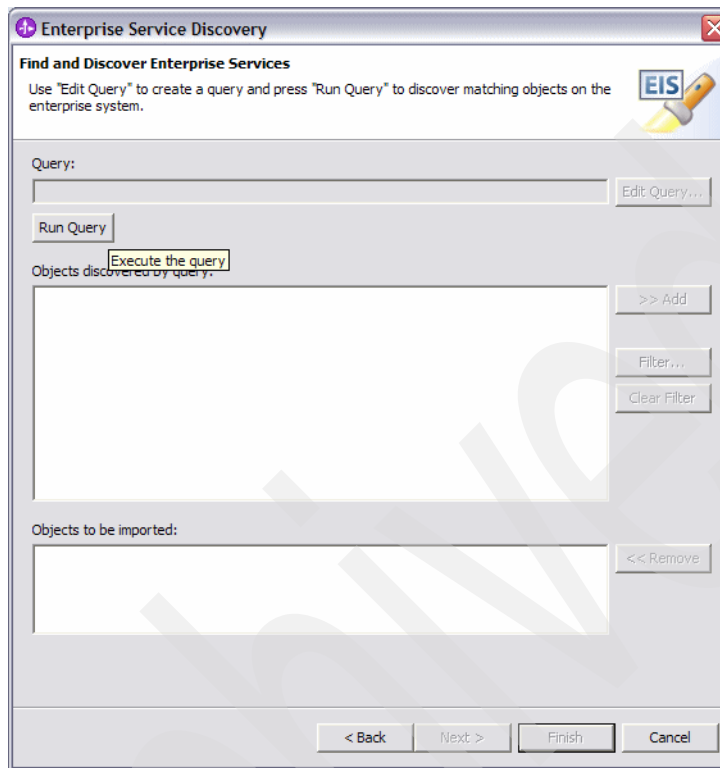


Figure 2-7 Run query

5. Select the **Outbound** folder in the discovered objects.

6. Click > **Add** (Figure 2-8).

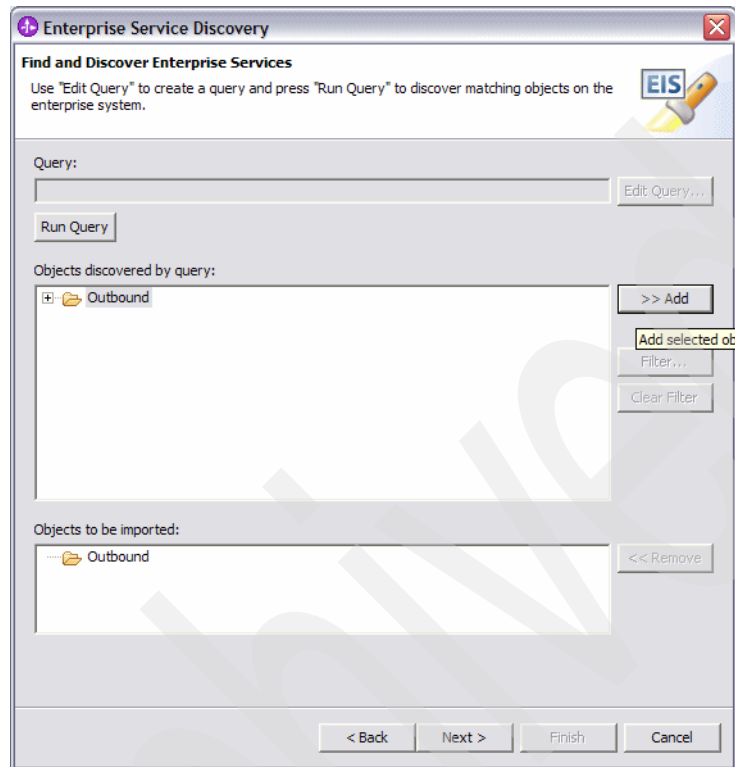


Figure 2-8 Select discovered objects

7. Enter a BO location (we chose wbi/asbo). Click **Next** (Figure 2-9).

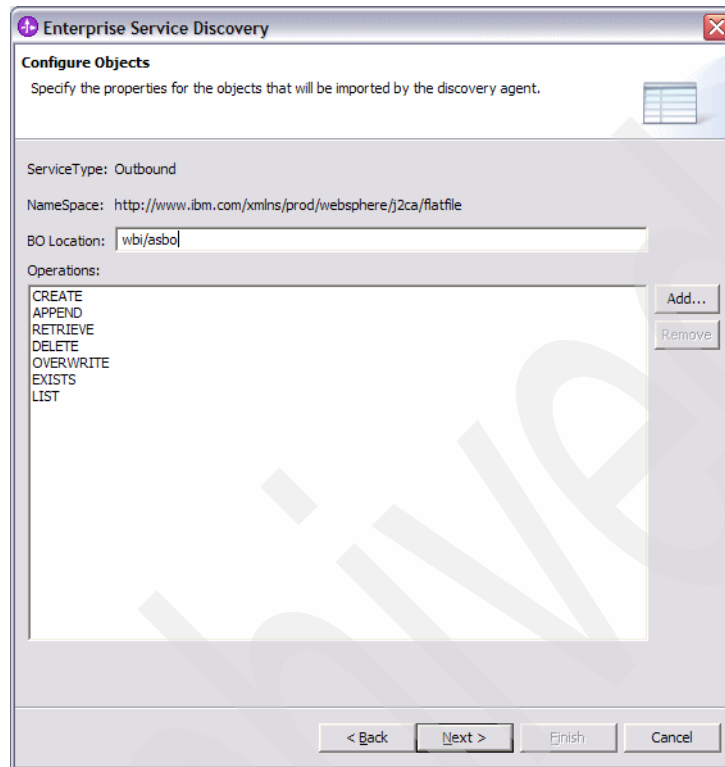


Figure 2-9 Enter BO location

8. Select **Use discovered connection properties**.
9. Click **Finish**.



10. In the External\_Maintenance module, we now see generated:

- Data types (FlatFile and FlatFileBG)
- Outbound interface (FlatFileOutboundInterface)

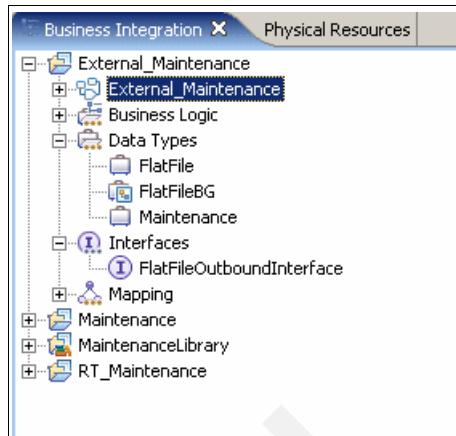


Figure 2-10 Generated artifacts

11. Open the assembly diagram. We see the FlatFileOutboundInterface, which is an import component with an EIS binding.

**Tip:** With an import component, a call is initiated by the service created in WebSphere Integration Developer. In other words, the module is invoking a program on an EIS system. In this case, we invoke the adapter.

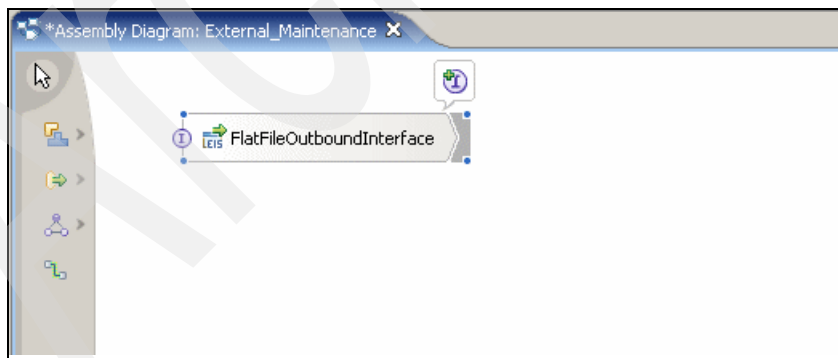


Figure 2-11 Assembly diagram

We now add the MaintenanceLibrary, as we will be using in the GBO data types in module.

12. Right-click **External\_Maintenance** and open the Dependency Editor.
13. Click **Add Libraries** and select **MaintenanceLibrary** (Figure 2-12).

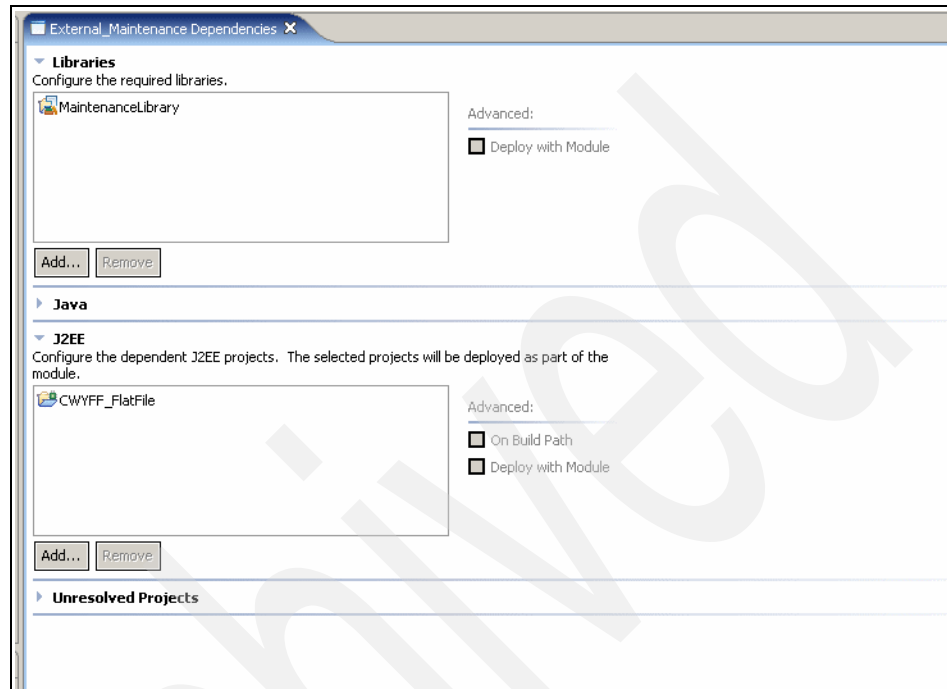


Figure 2-12 Dependencies

14. Save the changes.

15. We also see here in the J2EE pane (see Figure 2-12 on page 26) that the project for the Flat File connector itself will be deployed with our application. This is known as an imbedded deployment.

**Important:** WebSphere Adapter deployment considerations.

The deployment of WebSphere Adapters requires specific options regarding scope. You can deploy a WebSphere Adapter in two ways, using the administrative console:

- ▶ Standalone - The adapter is installed at the node level and is not associated with a specific application.  
  
Note that the deployment of standalone WebSphere Adapters is not supported in WebSphere Process Server v6.0.
- ▶ Embedded - The adapter is part of an application. Deploying the application also deploys the adapter.

For embedded WebSphere Adapters:

- ▶ The RAR file can be *application-scoped within an SCA module* (with EIS imports or exports).
- ▶ The RAR file can be *application-scoped within a non-SCA module*. The application itself, containing the EIS imports and exports, is a separate SCA module.

You should not install standalone WebSphere Adapters. The administrative console does not preclude the installation of standalone WebSphere Adapters, but this should not be done. WebSphere Adapters should be embedded in applications.

Only embedded WebSphere Adapters are appropriate for deployment in WebSphere Process Server. Furthermore, deployment of an embedded WebSphere Adapter is only supported for RAR files that are application-scoped within an SCA module. Deployment in a non-SCA module is not supported.

## 2.4 Create application-specific business objects

In this part of our scenario, the application-specific data refers to the data that will be sent and received by an external application (external contractors). The documents that we are exchanging with our partner contractors are XML. We now must create the data types that represent these XML file formats (the application-specific data objects).

Appendix A, “Additional material” on page 87, contains the XML schemas that are used. Example 2-1 is the ContractorRequest schema that we use for our business object.

*Example 2-1 ContractorRequest.xsd*

---

```
<?xml version="1.0"?>
<RC:schema targetNamespace="http://external.request.redmaint.com"
  xmlns:ContractorRequest="http://external.request.redmaint.com"
  xmlns:RC="http://www.w3.org/2001/XMLSchema">
  <RC:element name="From" type="RC:string"/>
  <RC:element name="To" type="RC:string"/>
  <RC:element name="ExpectedCompletion" type="RC:string"/>
  <RC:element name="ActualCompletion" type="RC:string"/>
  <RC:element name="StatusDescription" type="RC:string"/>
  <RC:element name="Tenant" type="RC:int"/>
  <RC:element name="Apartment" type="RC:int"/>
  <RC:element name="MaintenanceId" type="RC:int"/>
  <RC:element name="Description" type="RC:string"/>
  <RC:element name="CreateDate" type="RC:string"/>
  <RC:complexType name="Maintenance">
    <RC:sequence>
      <RC:element ref="ContractorRequest:From"/>
      <RC:element ref="ContractorRequest:To"/>
      <RC:element ref="ContractorRequest:ExpectedCompletion"/>
      <RC:element ref="ContractorRequest:ActualCompletion"/>
      <RC:element ref="ContractorRequest:StatusDescription"/>
      <RC:element ref="ContractorRequest:Tenant"/>
      <RC:element ref="ContractorRequest:Apartment"/>
      <RC:element ref="ContractorRequest:MaintenanceId"/>
      <RC:element ref="ContractorRequest:CreateDate"/>
      <RC:element ref="ContractorRequest:Description"/>
    </RC:sequence>
  </RC:complexType>
  <RC:element name="MaintenanceRecord" type="ContractorRequest:Maintenance"/>
</RC:schema>
```

---

We import this schema to create our data type:

1. Select **Data Types** → **Import**.
2. In the Import window, select **File System** → **Next**.
3. Select the **ContractorRequest** schema definition.
4. Click **Finish**.

5. This will create the *maintenance* data type in the data types category of the External\_Maintenance module. This data type is shown in Figure 2-13.

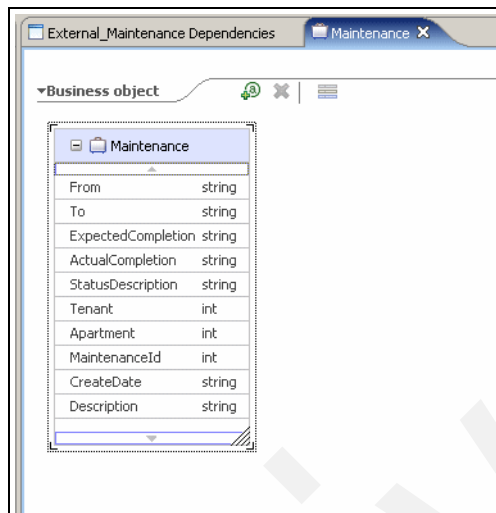


Figure 2-13 Maintenance data type

6. We check that the data type was correctly imported and that only the following elements are set as mandatory (Figure 2-14):

- To
- From
- Tenant
- Apartment
- MaintenanceId

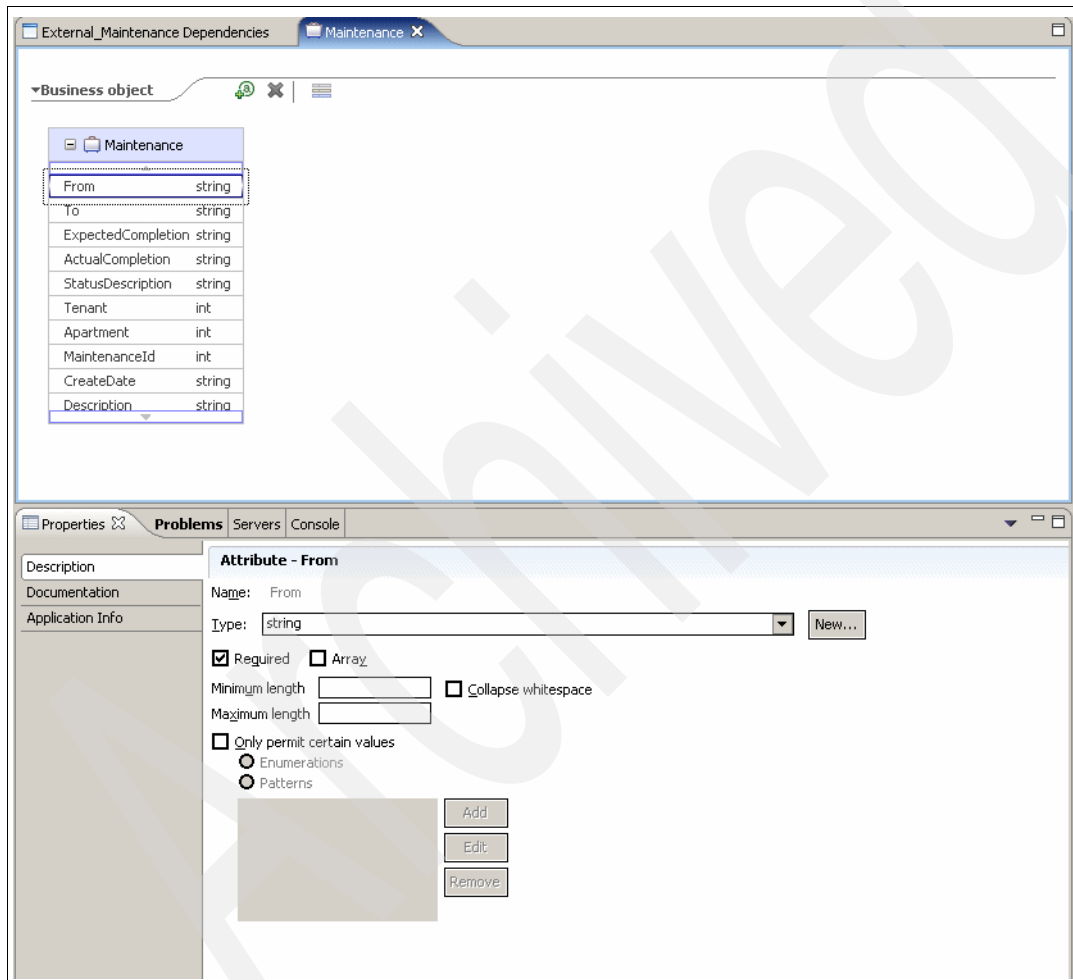


Figure 2-14 Attributes

7. Save any changes and close the maintenance data type.

## 2.5 Create interface map

An interface map resolves differences between the interfaces of interacting components.

Differences between interfaces in components that must interact with one another are common. These differences arise because in WebSphere Integration Developer we are often assembling components that were created for different applications. In our case, we have a GBO, which has resulted from data coming from an application (which is beyond the scope of this paper), and an XML file that must be written.

An interface map maps the operations and parameters of these interfaces so that the differences are resolved and the two components may interact. An interface map is like a bridge between the interfaces of two components, allowing them to be wired together despite differences.

We now need an interface map to handle the mediation:

1. Select **Interface Maps** → **New** → **Interface Map**.
2. Enter a name (we chose FlatFileOutbound) (Figure 2-15). Click **Next**.

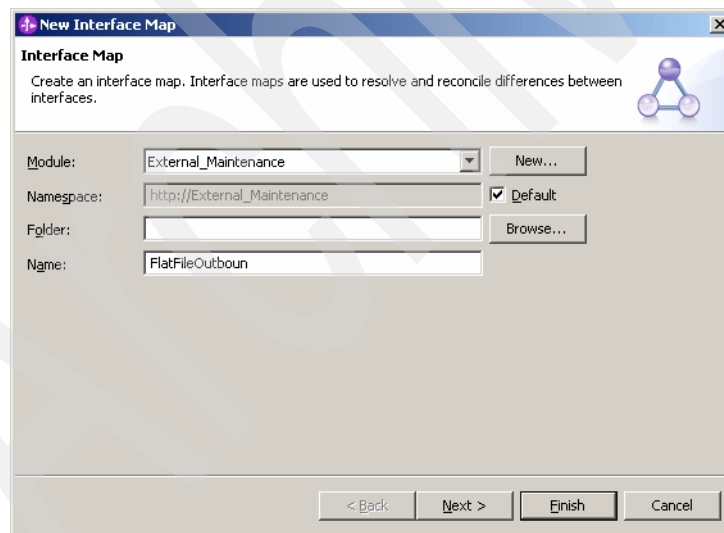


Figure 2-15 New interface map

3. Select a source interface (**ExternalContractor**) and target interface (**FlatFileOutboundInterface**) for mapping (Figure 2-16).

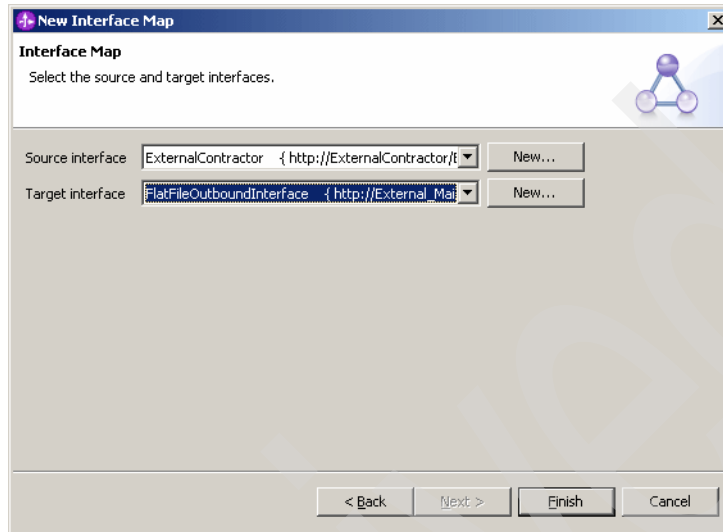


Figure 2-16 Select source and target interfaces

4. There may be an error on the Problems tab. This will be eliminated once the mapping is completed. Click **Finish**.

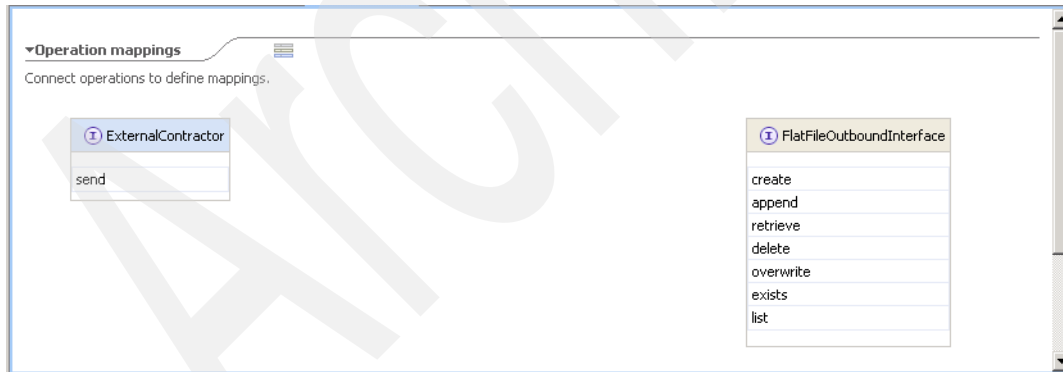


Figure 2-17 New interface map

5. Right-click the ExternalContractor **send** operation and select **Create Operation Mapping**.



- Click the **create** operation of the FlatFileOutboundInterface interface to complete the operation mapping (Figure 2-18).



Figure 2-18 Operation mapping

- Create parameter mapping between the GBO\_Maintenance and FlatFileBG data types interfaces by dragging a connection between the two (Figure 2-19).

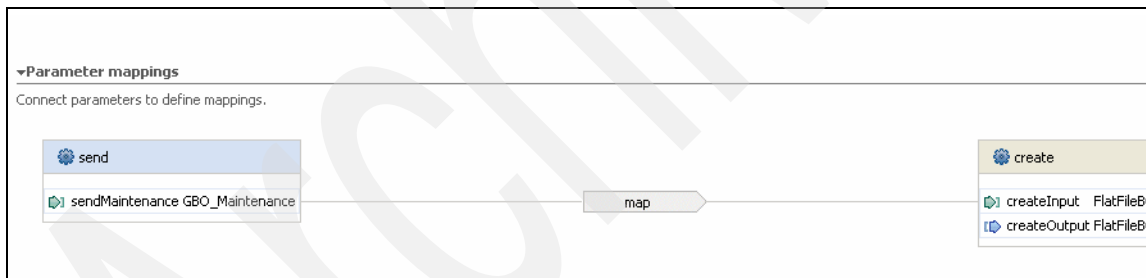


Figure 2-19 Create parameter mapping

- We now go to the Advanced view on the Properties tab for the creation of the required business object map.
- Click **New** at the business object Map.
- Enter a name (we chose GBO\_Maintenance\_to\_FlatFileBG) in the new window and click **Next**.
- Select input (**GBO\_Maintenance**) and output (**FlatFileBG**) data types for data mapping and click **Finish**.

## 12. Save the interface map (Figure 2-20).

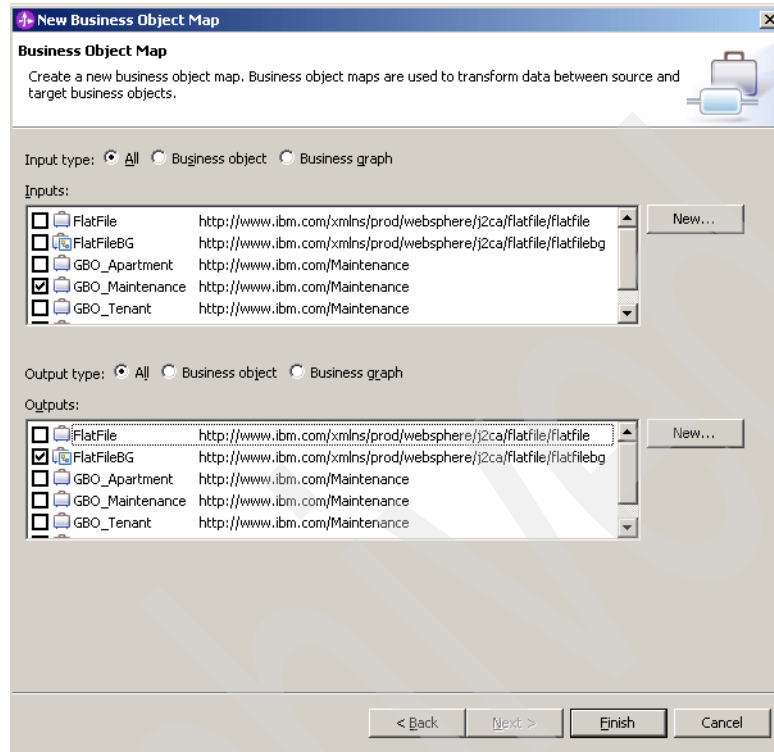


Figure 2-20 Select data types

**Tip:** If you receive a Schema validation of Map definition error on the Problem tab, you can ignore it. This will be eliminated once the data mapping is completed.

Now we map between GBO\_Maintenance data type and Maintenance data type.

*GBO\_Maintenance* is a generic BO that is used in the processing for our various internal applications. *Maintenance* is a application-specific BO that is compatible with the XML schema that our external applications require.

We use a variable to hold the transformed values, which has a BO type of maintenance.

After mapping between these two objects, we convert the application-specific data to the XML request message format required using the *BOXMLSerializer* API.

We move this transformed data into the *inputBytes* of the *FlatFileBG*.

1. Double-click the **GBO\_Maintenance\_to\_FlatFileBG** data map in the External\_Maintenance module.
2. Click **Add a Variable** (Figure 2-21).

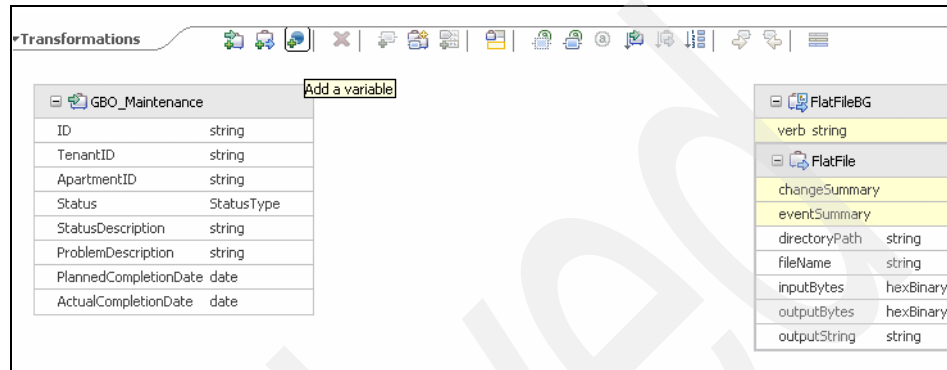


Figure 2-21 Add a variable

3. Click the new variable and give it a name (we chose `_Maintenance`, as it is essentially a holding area for our maintenance data) on the Properties tab and select **Business Object Type** of Maintenance (Figure 2-22).

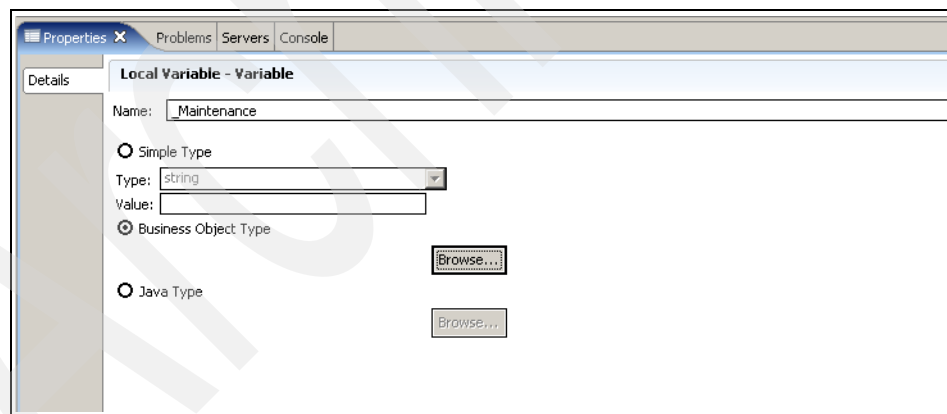


Figure 2-22 Rename variable

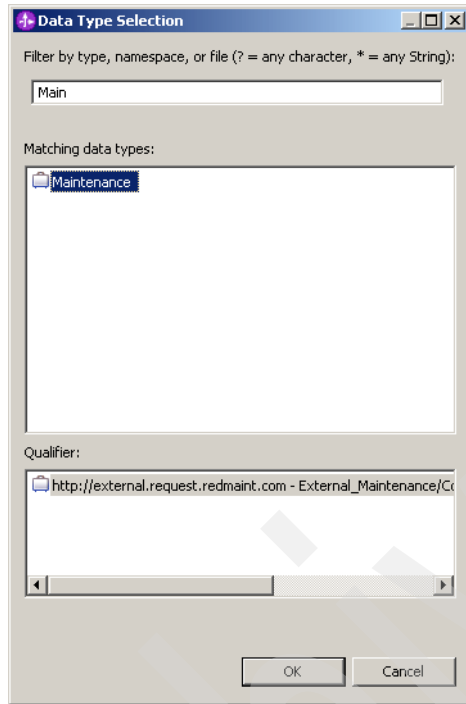


Figure 2-23 Select Maintenance data type

4. Click OK.

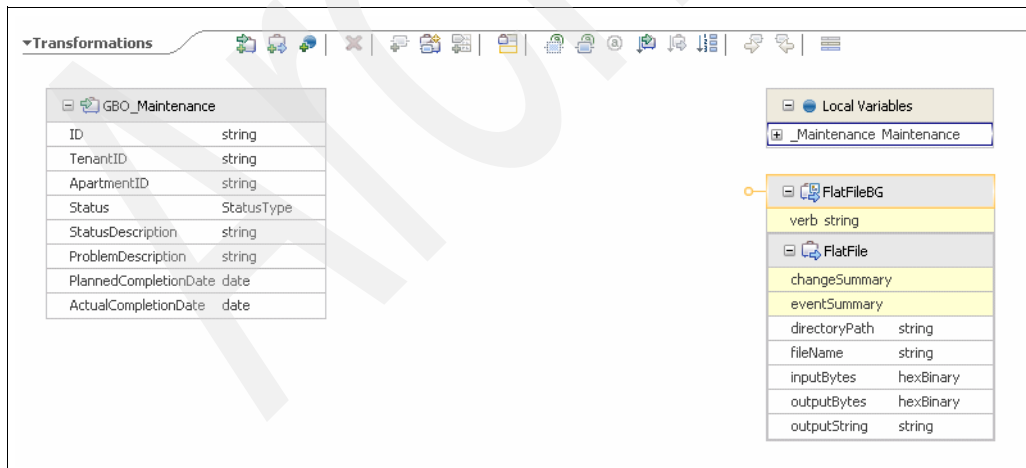


Figure 2-24 Variable in place

Now we have the all of objects ready for data mapping.

5. Expand the **\_Maintenance** local variable.
6. Select the **GBO\_Maintenance** ID and add a connection to the MaintenanceId of the **\_Maintenance** variable (Figure 2-25).

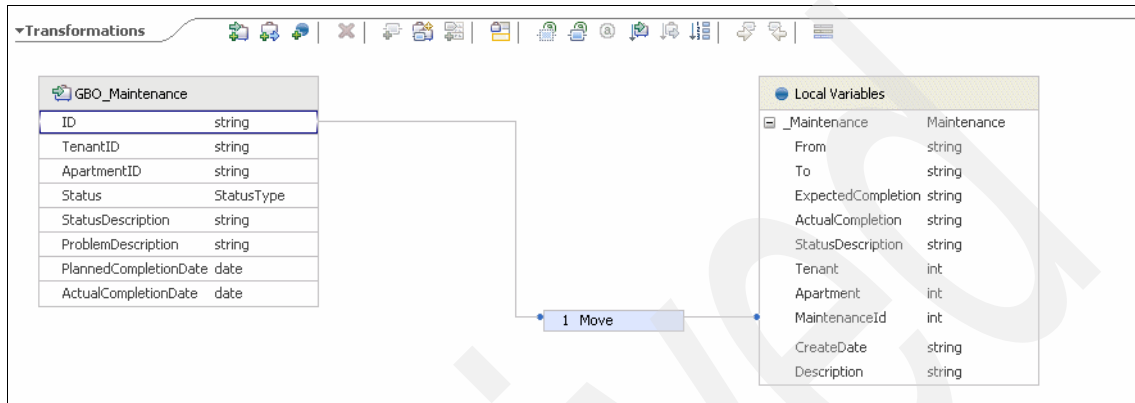


Figure 2-25 Add a move transformation

7. Complete the other move transformations, as shown in Figure 2-26.

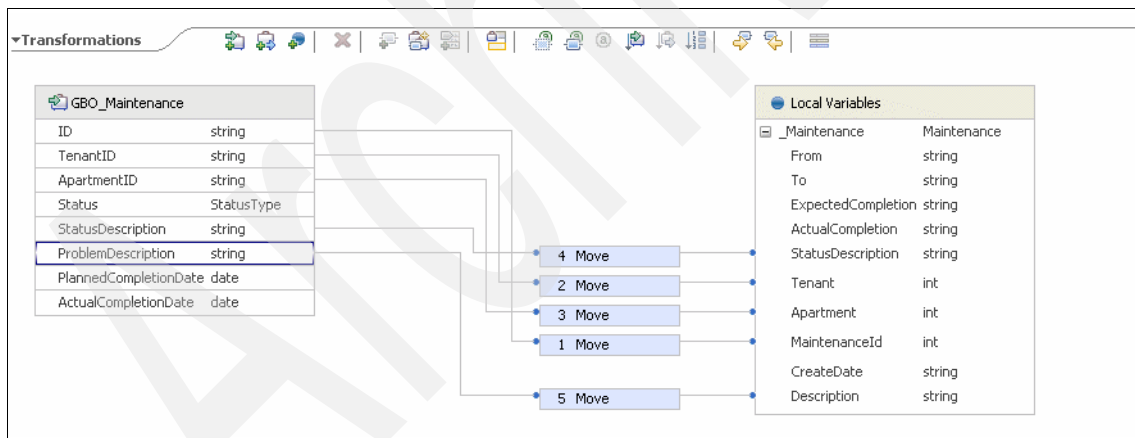


Figure 2-26 Move transformations

The source and target properties for the dates do not match:

- The PlannedCompletionDate is a date type.
- The ExpectedCompletion is a string type (the format of this string must be the date represented in milliseconds to meet the external application requirements).

We need to convert the date to a string by using *custom* mapping.

For our custom transformation, if the date exists (that is, not null) we will:

- Convert the date type to milliseconds (which will be a long).
- Convert the long type to a string.

8. Select **Transform type custom** on the Description tab in the Properties view (Figure 2-27).

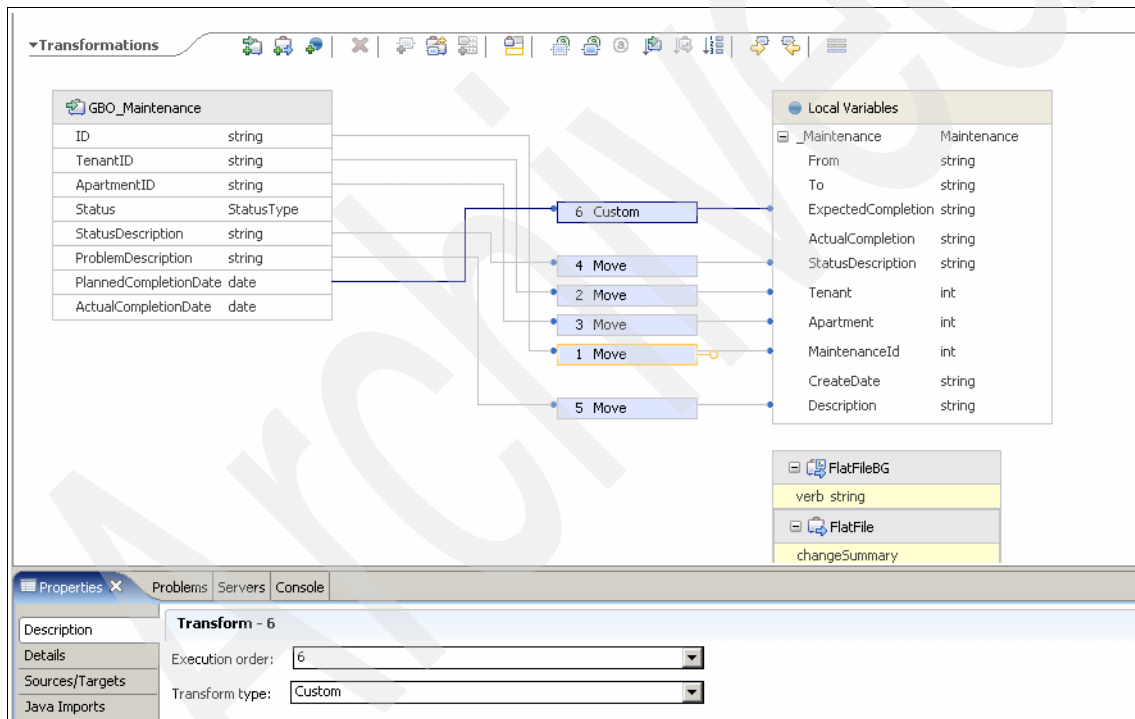


Figure 2-27 Custom transform

9. On the Details tab, click the **Visual** radio button to add the logic in the visual snippet editor.
10. Click **Expression®** in the left side tools and drop onto it the palette.

11. From the drop-down box select **GBO\_Maintenance\_PlannedCompletionDate**.
12. Add a check for not equal to null (`!= null`).
13. Click the **Choice** tool for the *if true - Otherwise* condition (Figure 2-27 on page 38).

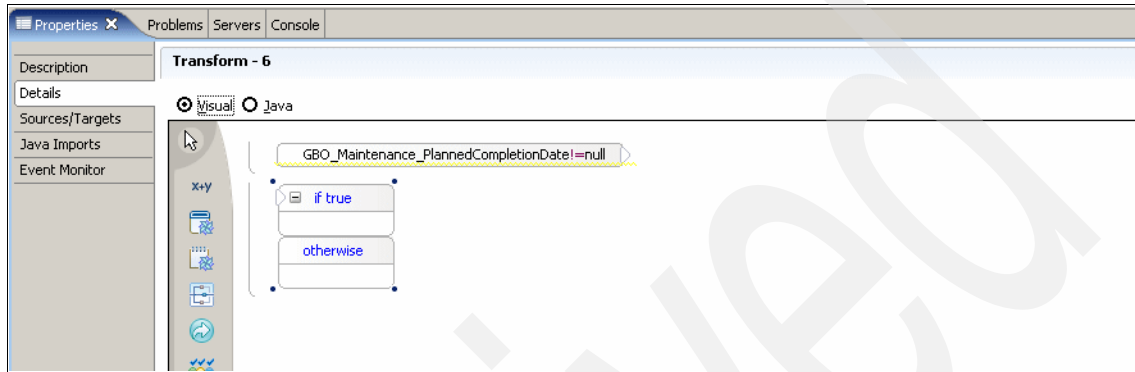


Figure 2-28 Add choice

14. Add a link between the true condition and the expression, as shown in Figure 2-29.



Figure 2-29 Add connection

15. Again, use the expression button to add the `GBO_Maintenance_PlannedCompletionDate`. Drop this into the true condition box. This becomes the input for our transformation.
16. Click the **Java** tool and specify type date with qualifier **java.util** and select the **getTime()** visual snippet and click **OK**.

17. Drop it in the *choice* space and it will create the `getTime` icon. This `getTime` will convert the date to milliseconds (Figure 2-30).

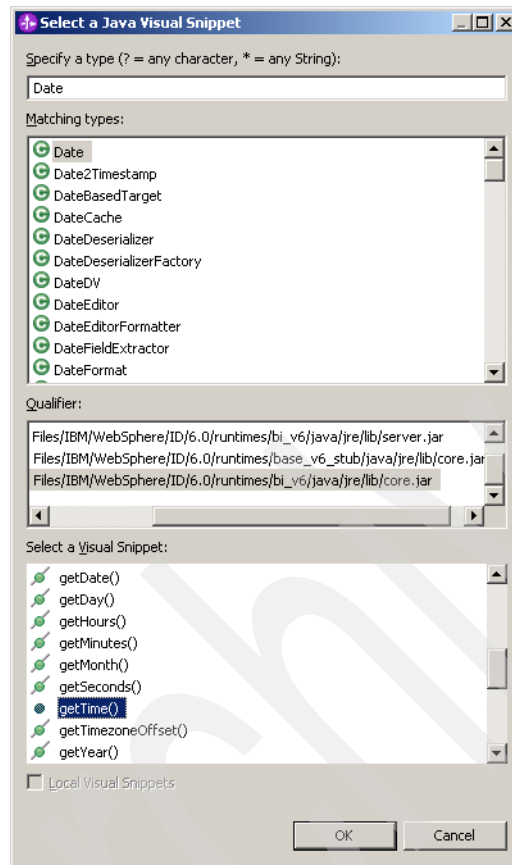


Figure 2-30 Select method

18. Click the **Java** tool and specify type *string* with qualifier `java.lang` and select the **valueOf(long)** visual snippet and click **OK**.

19. Drop it in the choice space and it will create the `valueOf` icon.



20. The `valueOf` will convert the milliseconds to a string (Figure 2-31).

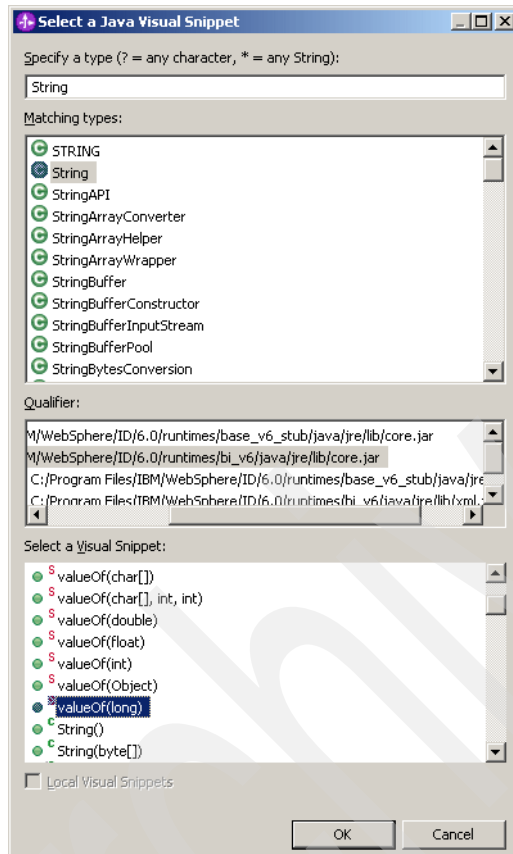


Figure 2-31 Select method

21. Use the expression button to drop the `_Maintenance_ExpectedCompletion` into the true condition box. (This is the output from the transformation.)

22.Link all of these objects as shown in Figure 2-32. The custom code is completed. It will now convert from date to milliseconds to string.

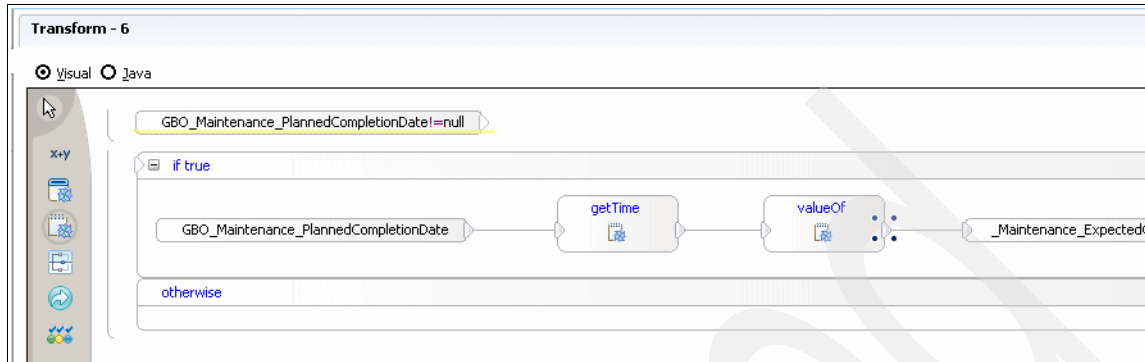


Figure 2-32 Completed custom transformation

23.We must set the constant values that are used as the trading partner identifiers. (The external application requires that we set AS2 identifiers required for document trading.)

24.Right-click the **From** variable in the \_Maintenance BO and select **CustomAssign** (Figure 2-33).

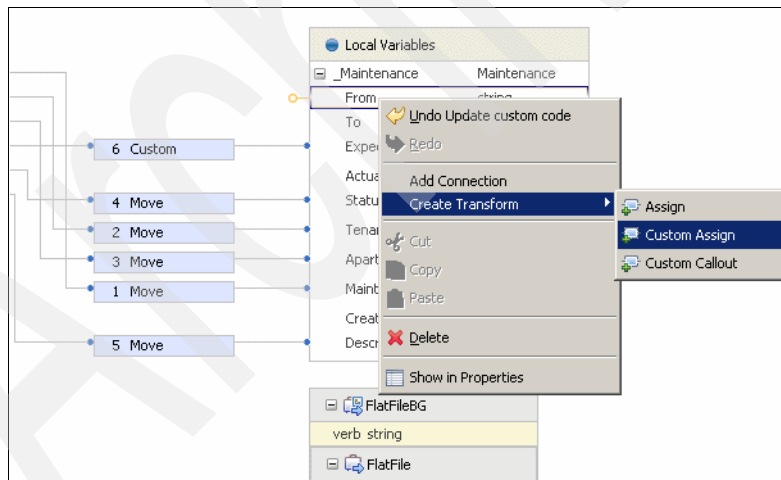


Figure 2-33 Select custom assign

25. In the visual view, assign redmaint to the From property of \_Maintenance, as shown Figure 2-34.

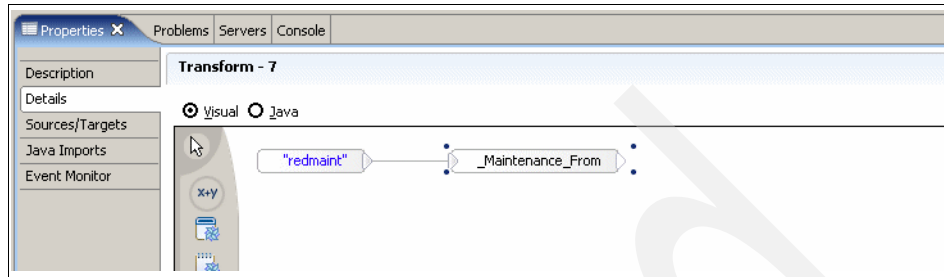


Figure 2-34 Custom assign

26. Add Custom Assign to the To variable of \_Maintenance BO, and in the Visual view, assign redcontractor to To of \_Maintenance, as shown in Figure 2-35.

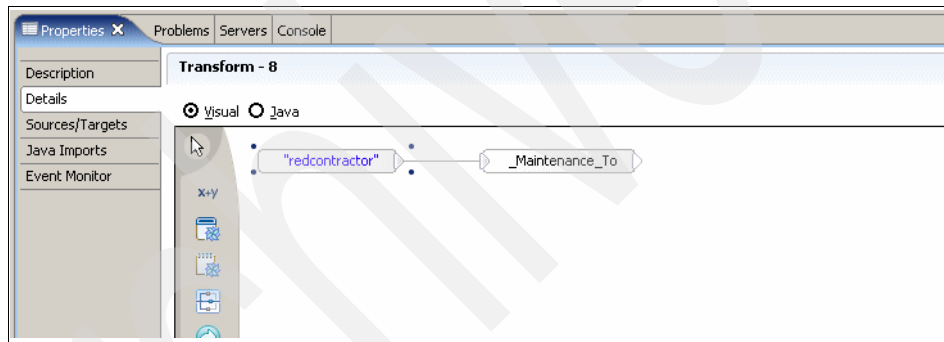


Figure 2-35 Custom assign

We now set the various file attributes:

1. We start with the directory path to the location where we will be putting the outbound file. This is the directory that the external application will be polling for new files to be processed/transmitted.

2. Create an *Assign* transformation for `directoryPath` of `FlatFileBG` and assign a user-defined value (we chose `C:\IBM\FlatFileOut` as our location) ():

`C:\IBM\FlatFileOut`

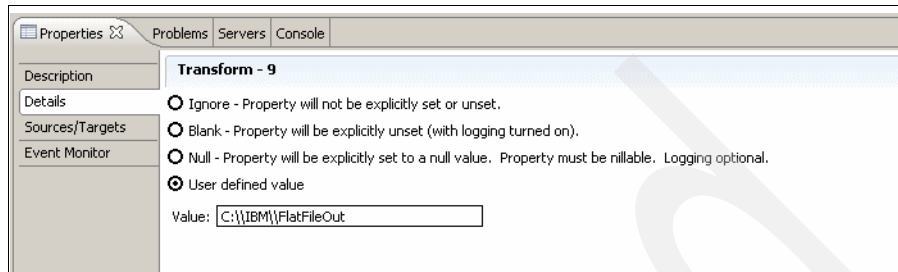


Figure 2-36 Set file path

3. Create a Custom Assign transformation for `fileName` of the `FlatFileBG` and add the custom Java code as shown in Example 2-2.

**Tip:** As a unique file name is required for each new file created, we will simply suffix the file name with a time stamp using a simple custom transformation.

Example 2-2 Custom Java code

```
SimpleDateFormat sdf = new SimpleDateFormat("MM_dd_yy_hhmmss");
FlatFileBG_FlatFile_fileName =
"ContractorRequest."+sdf.format(Calendar.getInstance().getTime())+".xml";
```

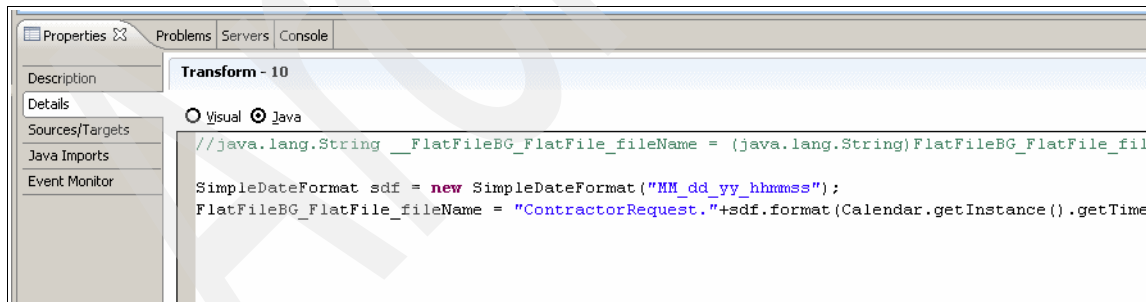


Figure 2-37 Transform file name

4. Add a connection between \_Maintenance BO to variable inputBytes of FlatFileBG with a Custom transform. This will create the contents of the file itself (Figure 2-38).

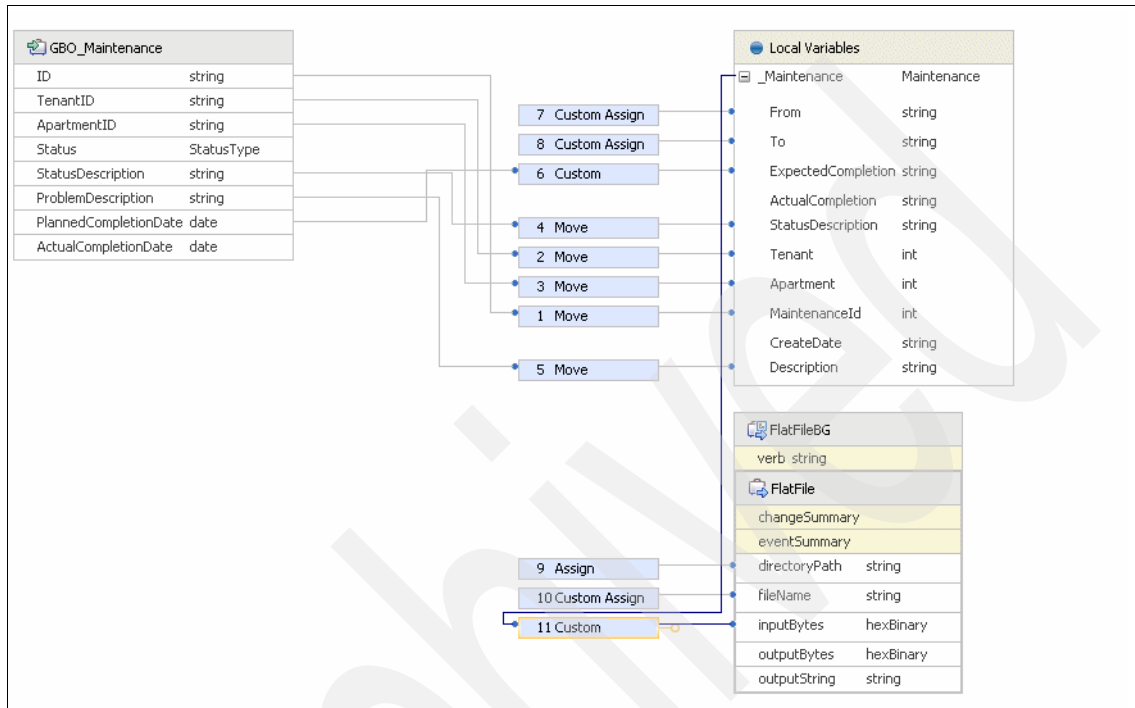


Figure 2-38 Add contents

5. Add the following Java code for this transform (Example 2-3).

**Tip:** Here we have used the BOXMLSerializer API for serialization from BO to XML message and we assign it to the inputBytes of FlatFile BG. The FlatFile Adapter creates the file with the converted XML data. In this way we are mimicking the function of the XML Data Handler in the WebSphere BI adapters.

#### Example 2-3 XML transformation code

```
ByteArrayOutputStream bos=new ByteArrayOutputStream();
BOXMLSerializer xmlSerializerService =(BOXMLSerializer) new
    ServiceManager().locateService("com/ibm/websphere/bo/BOXMLSerializer");
xmlSerializerService.writeDataObject(_Maintenance,
    "http://external.request.redmaint.com","Maintenance",bos);
FlatFileBG_FlatFile_inputBytes = bos.toByteArray();
```

6. Add the related imports in the Java imports, as shown in Example 2-4.

*Example 2-4 Java imports*

```
import java.io.ByteArrayOutputStream;
import com.ibm.websphere.bo.BOXMLSerializer;
import com.ibm.websphere.sca.ServiceManager;
import java.text.SimpleDateFormat;
import java.util.Calendar;
```

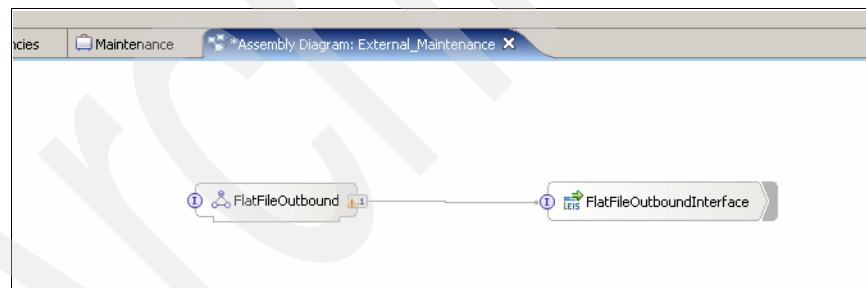
7. Save all changes.

The mapping transformations are now completed for the outbound scenario.

## 2.6 Build the outbound component of External\_Maintenance

We now must assemble the required components:

1. Select the interface map **FlatFileOutbound** and drop it onto the assembly diagram.
2. Wire it to the FlatFileOutboundInterface.
3. Save the changes (Figure 2-39).



*Figure 2-39 Add interface map*

In WebSphere Integration Developer, a published interface from a component can offer its business service to the outside world. Exports have interfaces that are the same as or a subset of the interfaces of the component that they are associated with so that the published service can be called. We expose the flat file outbound of our External\_Maintenance as a service that can be called by other components in an integration solution, using an export.

4. Add an export component to the assembly diagram. We named ours ExternalContractorOutbound.

5. The interface we need here is the ExternalContractor interface in our common MaintenanceLibrary module. Add the interface for this export by clicking at the interface icon at the top of the ExternalContractorOutbound and selecting **ExternalContractor** (Figure 2-40).

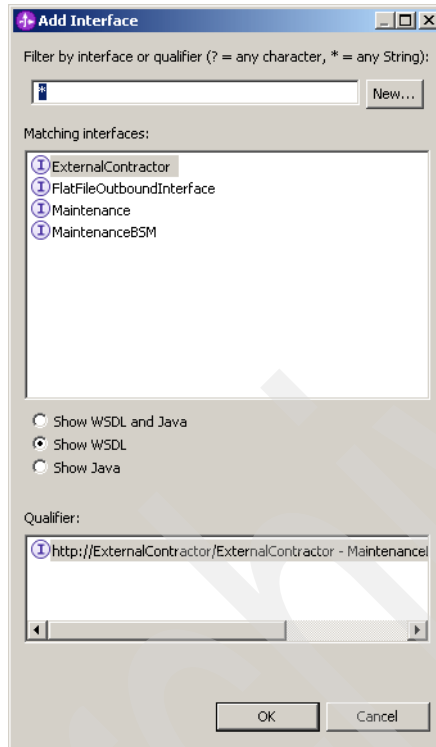


Figure 2-40 Select interface

**Tip:** Imports and exports require binding information, which specifies the means of transporting the data from other modules. An import binding describes the specific way an external service is bound to an import component. An export binding describes the specific way a module's services are made available to clients. An export component with an SCA binding lets us offer a service to other modules.

6. Wire ExternalContractorOutbound to the FlatFileOutbound interface map and generate an SCA binding for the ExternalContractorOutbound (Figure 2-41).

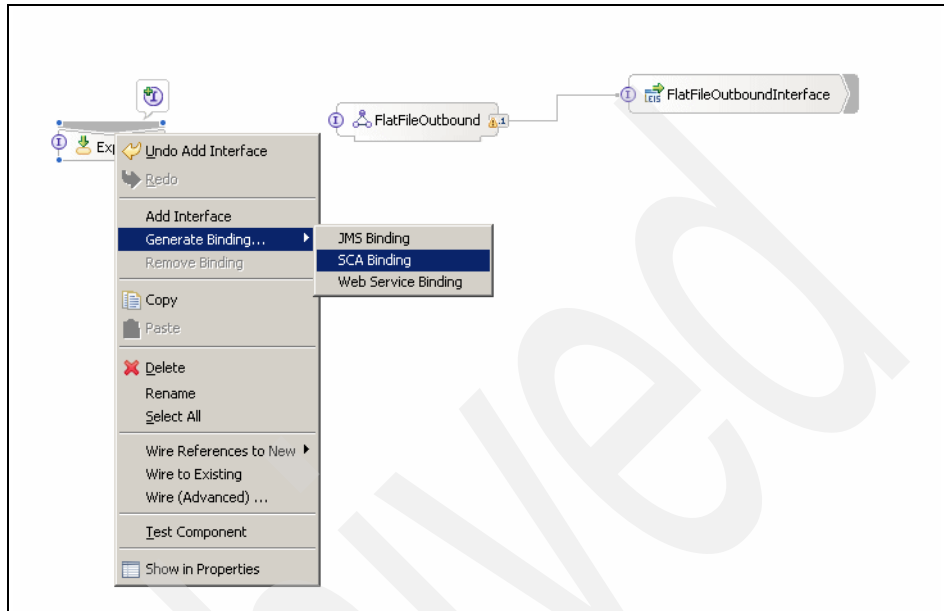


Figure 2-41 Add SCA binding

7. Save the changes (Figure 2-42).

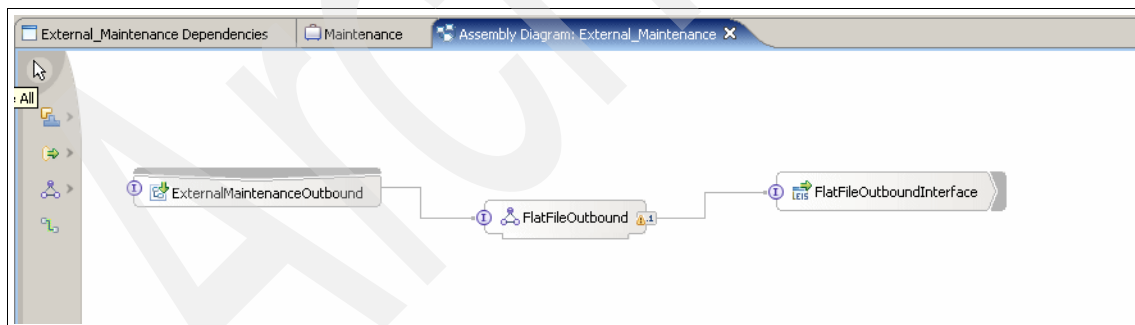


Figure 2-42 Completed assembly diagram

We have one final thing to do prior to testing. There are possibly two different FlatFile.xsd files — one that is shipped within the RAR file and one that is generated by the enterprise service discovery tool. Any references to this file will have a problem with a clash. We must eliminate any potential duplication:

1. Go to the Physical Resources view.



**Note:** If you do not see this, select **Window** → **Show View** → **Physical Resources**.

2. Select the **CWYFF\_FlatFile** project.
3. Select **Connector Properties** → **CWYFF\_FlatFile** → **connectorModule** (Figure 2-43).

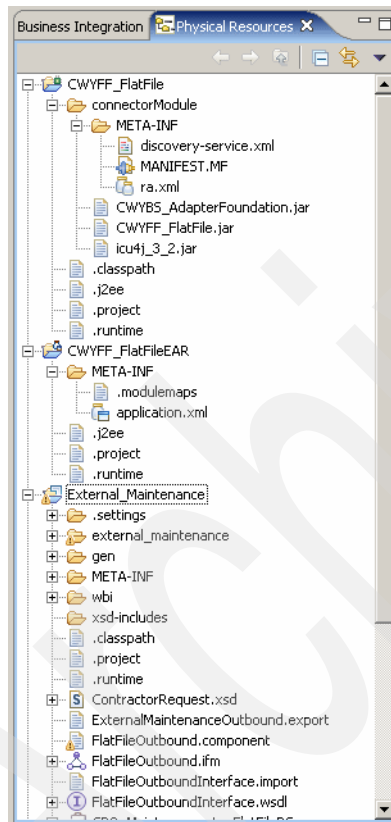


Figure 2-43 Connector module

4. If you see the FlatFile.xsd file, delete it.
5. Save the changes and rebuild the External\_Maintenance project.

The External\_Maintenance outbound file processing component is ready for testing.

## 2.7 Test the outbound file creation

Before commencing the test, ensure that the server is started:

1. Click **Add and Remove Projects**.
2. Select **External\_MaintenanceApp** and add it to configured projects on the server (Figure 2-44).

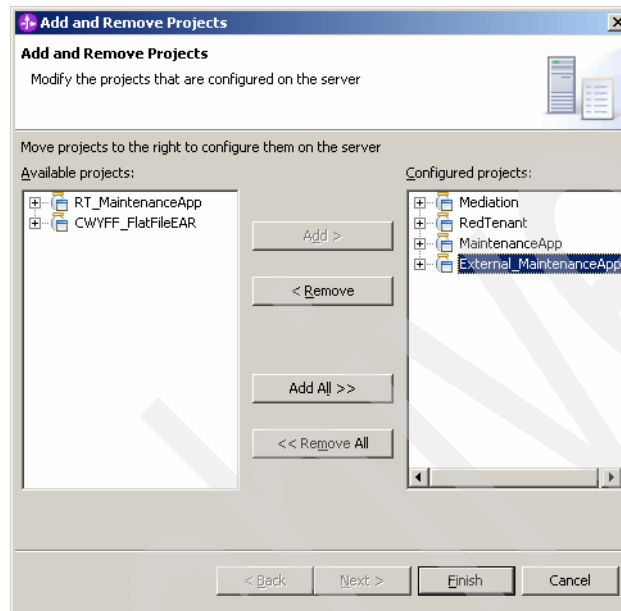


Figure 2-44 Add the project

3. Once the module has been successfully deployed, right-click the **ExternalContractorOutbound** export.

4. Click **Test Component** (Figure 2-45).

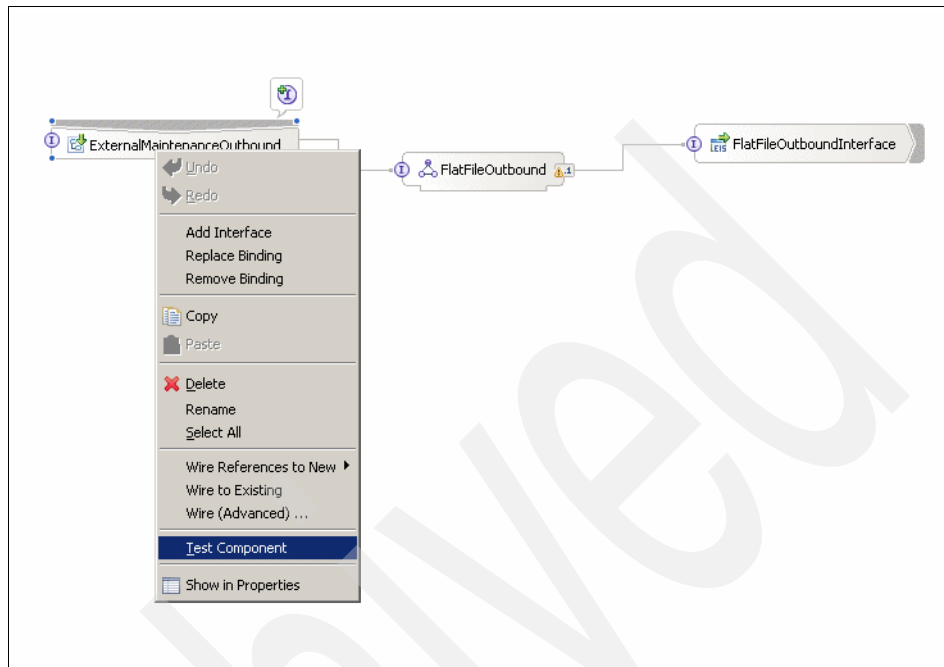


Figure 2-45 Test component

5. This will open a new window with name ExternalContractorOutbound\_Test (Figure 2-46).

**Tip:** Emulators enable you to emulate references, components, or imports in your module.

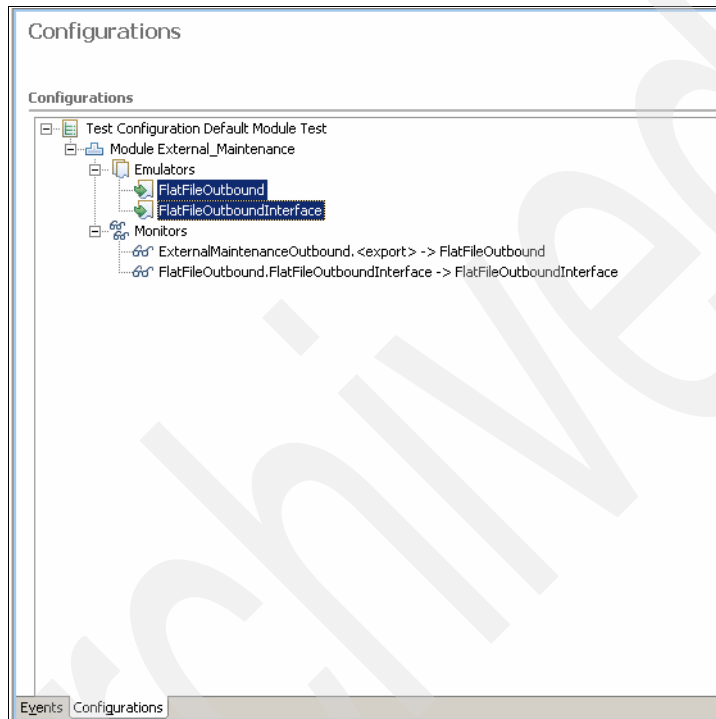


Figure 2-46 Emulators

6. Go to **Configurations** tab and remove the all of the emulators on the Emulator tab (Figure 2-47 on page 53).

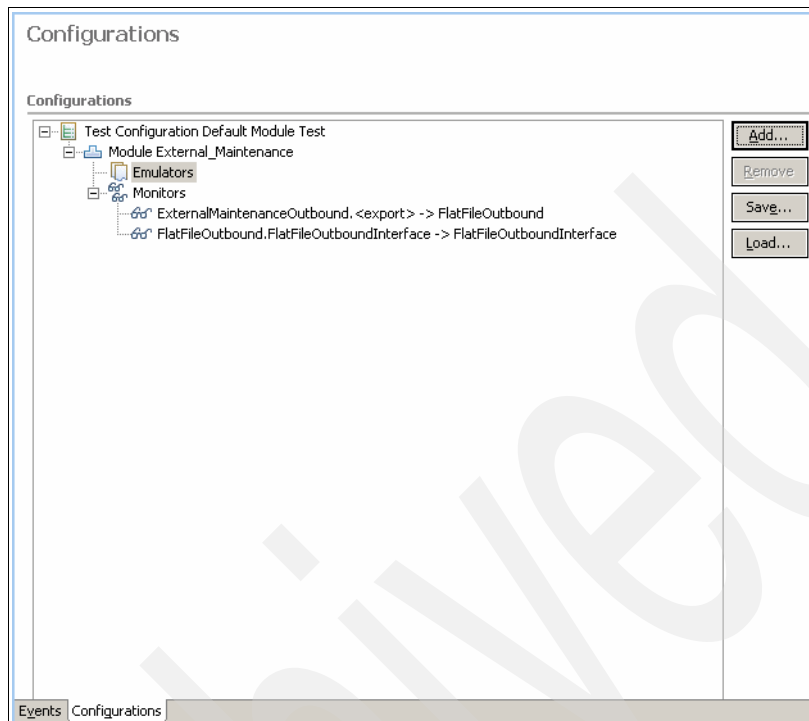


Figure 2-47 Remove emulators

7. Go to the **Events** tab.

8. Fill in some values for the GBO\_Maintenance (as shown in Figure 2-48) and click **Continue**.

[Click here](#) like to invoke. Click Continue to run.

---

► General Properties

▼ Detailed Properties

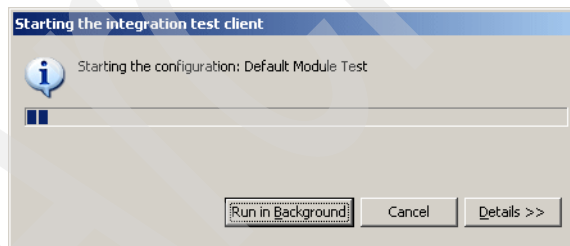
<a href="#">Configuration:</a>	Default Module Test
<a href="#">Module:</a>	External_Maintenance
<a href="#">Component:</a>	ExternalMaintenanceOutbound
<a href="#">Interface:</a>	ExternalContractor
<a href="#">Operation:</a>	send

Initial request parameters

Name	Type	Value
<input type="checkbox"/> sendMaintenance	GBO_Maintenance	
ID	string	23
TenantID	string	100
ApartmentID	string	1
Status	String	X
StatusDescription	string	This is a test for the FLAT FILLING
ProblemDescription	string	No problem - we hope
PlannedCompletionDate	date	2006-31 12
ActualCompletionDate	date	2002-01-01

Figure 2-48 Test maintenance data

9. This starts the default module test (Figure 2-49).



*Figure 2-49 Start test*

10. We now see the execution of the map.
11. The data is sent to FlatFileBG for file creation in the user-defined directory.

12. We then get a successful response back. The flow of execution is shown in Figure 2-50.

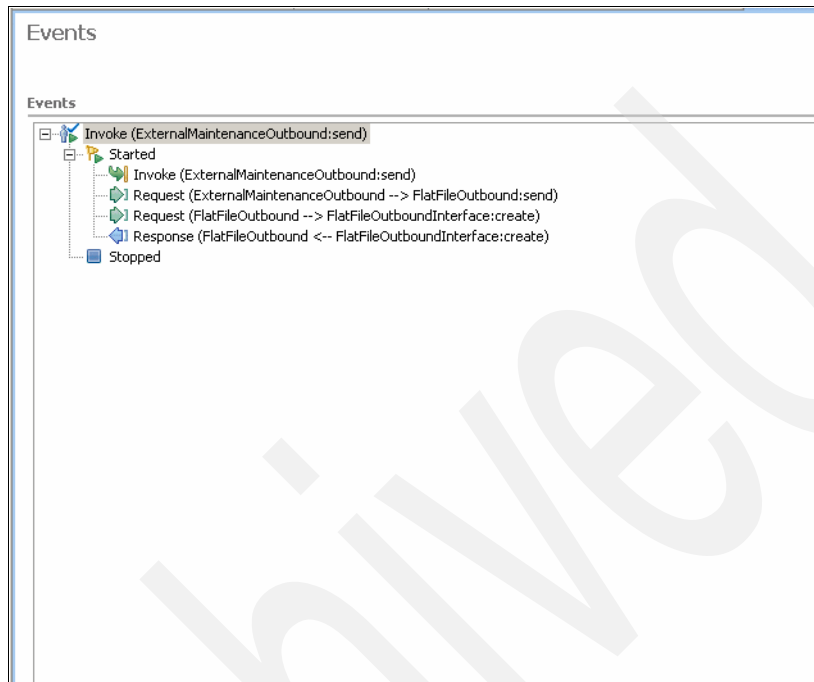


Figure 2-50 Successful execution

13. Now check the ContractorRequest.xml file with creation time, created in the output directory that we configured in the mapping (Figure 2-51).

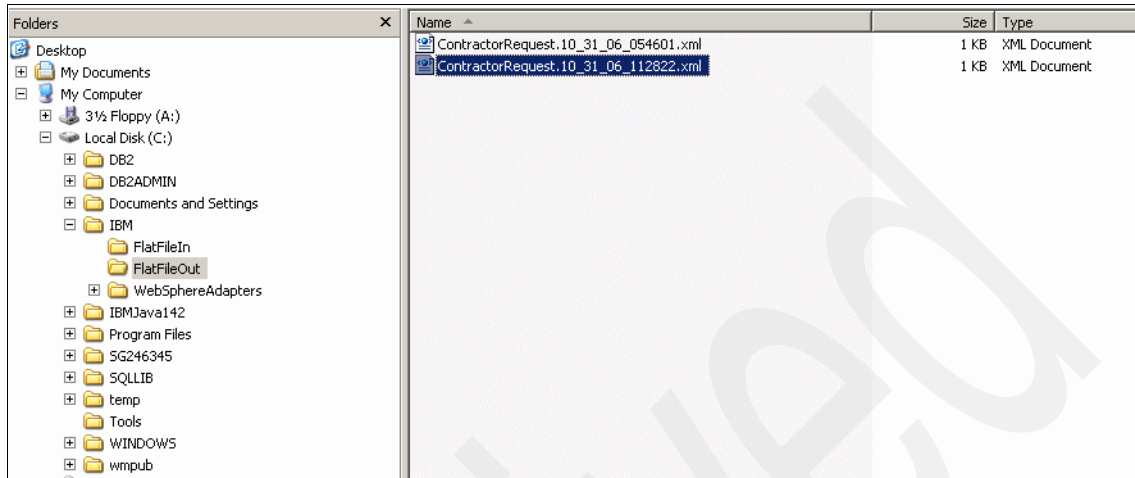


Figure 2-51 New file created

14. Open the ContractorRequest.xml file for verification. This will be processed by the external application. The contents will be similar to Example 2-5. (Verify that the contents match the values that were entered in the test.)

*Example 2-5 XML file output*

```
<?xml version="1.0" encoding="UTF-8"?>
<external:Maintenance xsi:type="external:Maintenance"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:external="http://external.request.redmaint.com">
  <external:From>redmaint</external:From>
  <external:To>redcontractor</external:To>
  <external:ExpectedCompletion>1215835200000</external:ExpectedCompletion>
  <external:StatusDescription>This is a test for the FLAT FILE outbound
</external:StatusDescription>
  <external:Tenant>100</external:Tenant>
  <external:Apartment>1</external:Apartment>
  <external:MaintenanceId>23</external:MaintenanceId>
  <external:Description>No problem - we hope</external:Description>
</external:Maintenance>
```

15. Close the test module without saving it.

16. Remove the ExternalContractorApp project from the server.

We have completed the testing for the outbound file creation. We now move on to the inbound processing.



## Process inbound files

We now move on to the processing of incoming files. In this example the adapter receives an XML file from our external application and deserializes it into a BO (using the BOXMLSerializer API). It is then forwarded to the WebSphere Process Server.

### 3.1 Enterprise service discovery for inbound artifacts

We use the enterprise service discovery wizard again for the configuration of the inbound adapter service:

1. Right-click the **ExternalContractor** module and click **New** → **Enterprise Service discovery** (Figure 3-1).

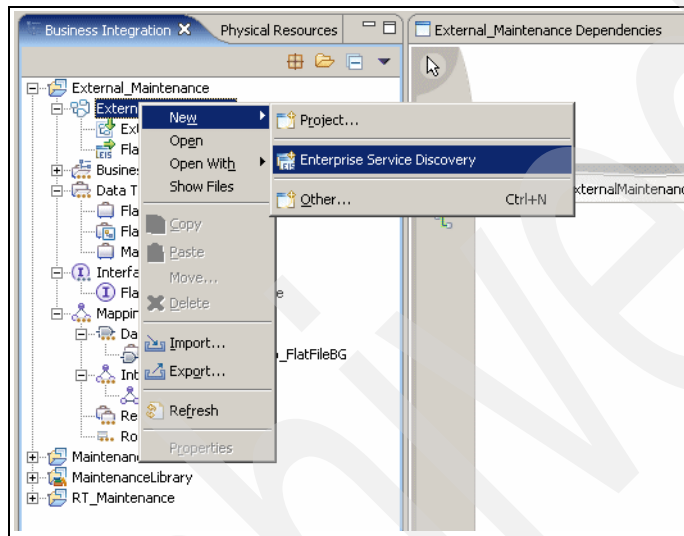


Figure 3-1 Enterprise Service discovery

2. In the New window click **IBM WebSphere Adapter for FlatFile**. Click **Next** (Figure 3-2).

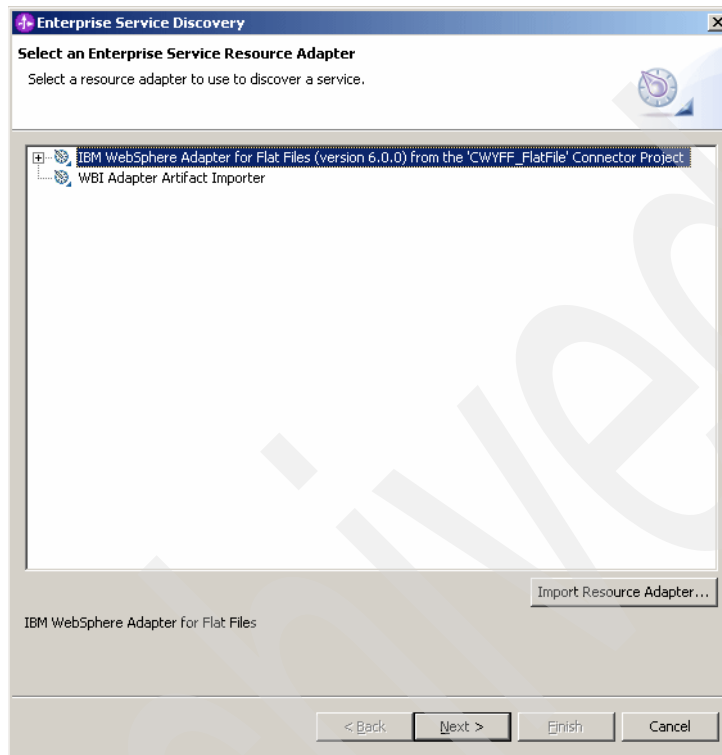


Figure 3-2 Select the Flat File Adapter

3. Select service type **Inbound**. Click **Next** (Figure 3-3).

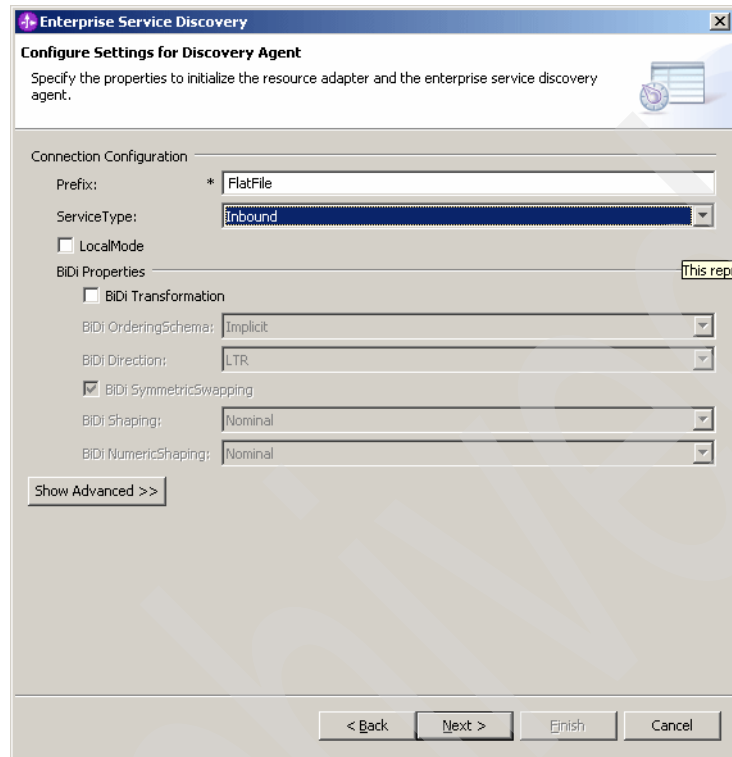


Figure 3-3 Select inbound service

4. Click **Run Query**.
5. Click the **Inbound Artifacts** folder.

6. Click **Next** (Figure 3-4).

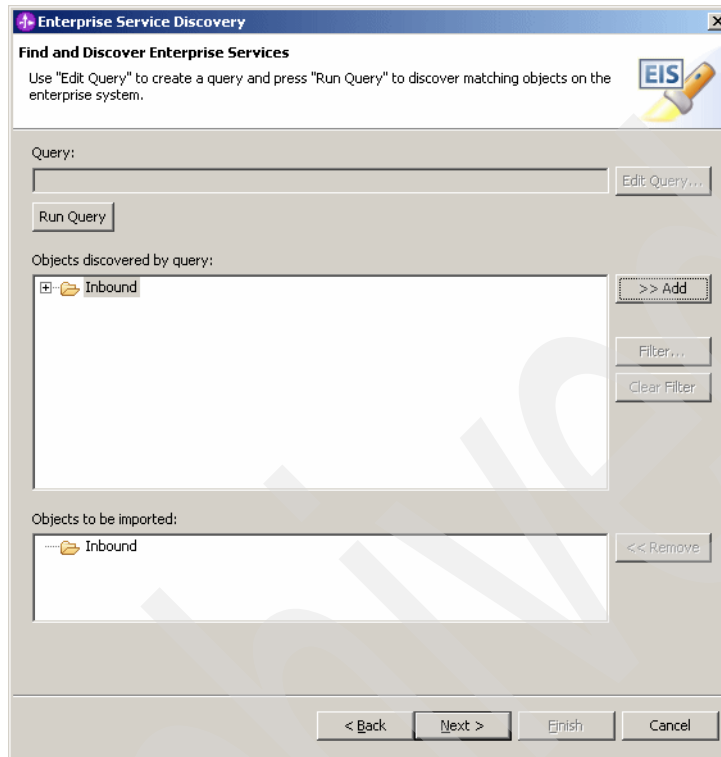


Figure 3-4 Run query

7. Enter a BO location (we chose wbi/asbo). Click **Next** (Figure 3-5).

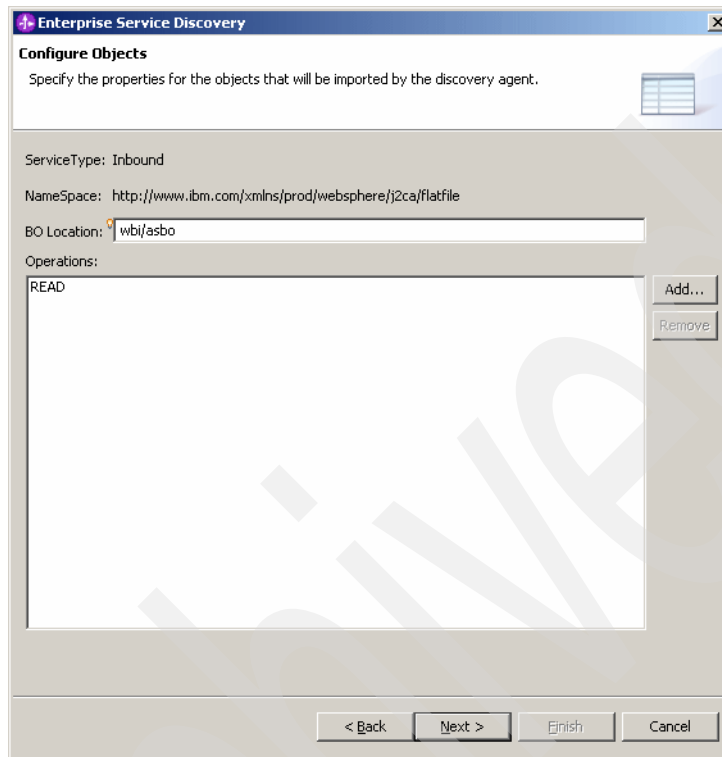
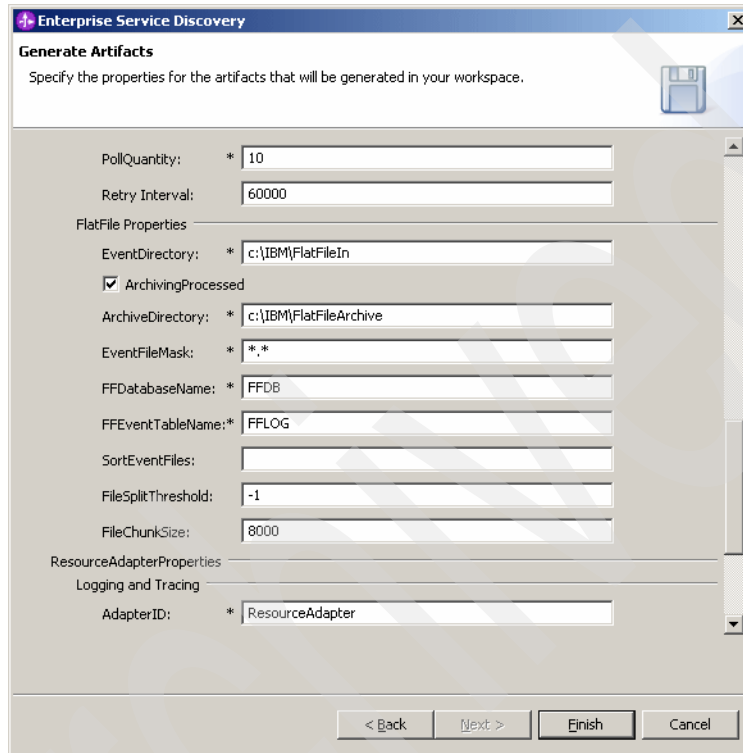


Figure 3-5 Select BO location

8. Select to use the discovered connection properties. Scroll down the window.

9. Enter the EventDirectory location where the inbound files will be arriving (this is the directory that will be polled at runtime). We chose C:\IBM\FlatFileIn and ArchiveDirectory (which is where the files will be moved to after they have been processed). We chose C:\IBM\FlatFileArchive (Figure 3-6).



The image shows a Windows-style dialog box titled "Enterprise Service Discovery". Below the title bar is a section labeled "Generate Artifacts" with the instruction "Specify the properties for the artifacts that will be generated in your workspace." and a floppy disk icon. The dialog contains several input fields and sections:

- PollQuantity:** \* 10
- Retry Interval:** 60000
- FlatFile Properties** (section header)
  - EventDirectory:** \* c:\IBM\FlatFileIn
  - ☒ ArchivingProcessed
  - ArchiveDirectory:** \* c:\IBM\FlatFileArchive
  - EventFileMask:** \* \*.\*
  - FFDatabaseName:** \* FFDB
  - FFEventTableName:** \* FFLOG
  - SortEventFiles:** (empty field)
  - FileSplitThreshold:** -1
  - FileChunkSize:** 8000
- ResourceAdapterProperties** (section header)
  - Logging and Tracing** (section header)
    - AdapterID:** \* ResourceAdapter

At the bottom of the dialog are four buttons: "< Back", "Next >", "Finish", and "Cancel".

Figure 3-6 Enter directory names

10. Click **Finish**. We now see that the FlatFileInboundInterface is added to the assembly diagram.

11. Save all changes (Figure 3-7).

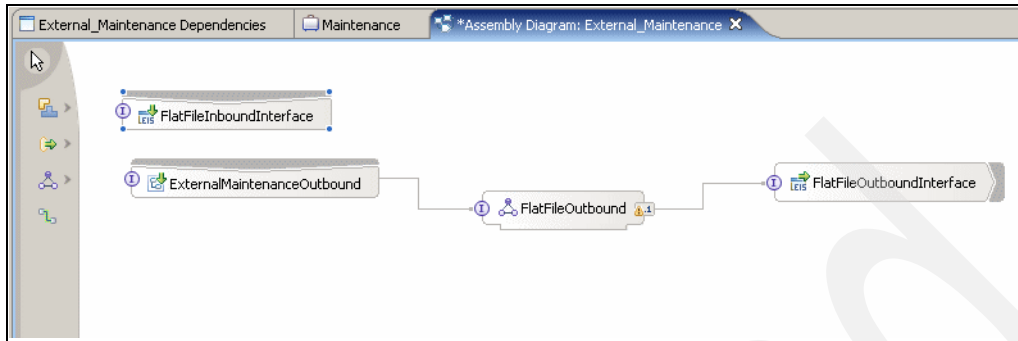


Figure 3-7 Assembly diagram

## 3.2 Create the interface and interface map

We now create an interface for our inbound processing and selection:

1. Right-click **Interfaces** in the ExternalContractor module and click **New** → **Interface**.
2. Enter a new interface name (we entered MaintenanceInbound).
3. Click **Finish** (Figure 3-8).

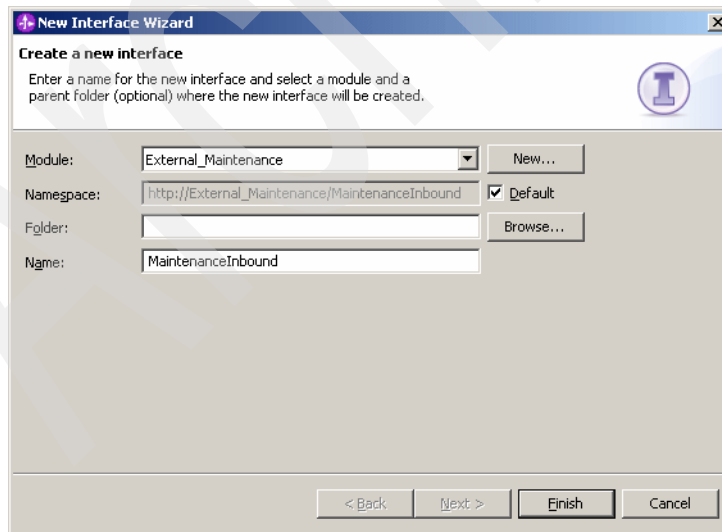


Figure 3-8 New interface



4. In the interface editor, add a one-way operation (we named ours select) (Figure 3-9).

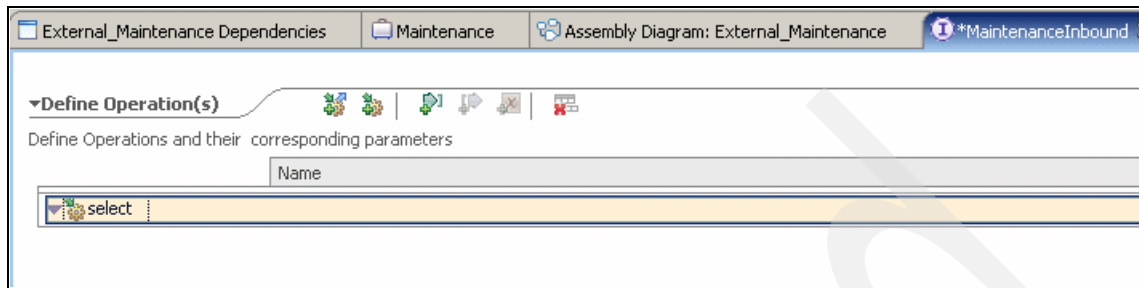


Figure 3-9 New one-way operation

5. Right-click the **select** operation and add an input parameter (we chose readFile) with a type of **GBO\_Maintenance**.
6. Save the new interface (Figure 3-10).

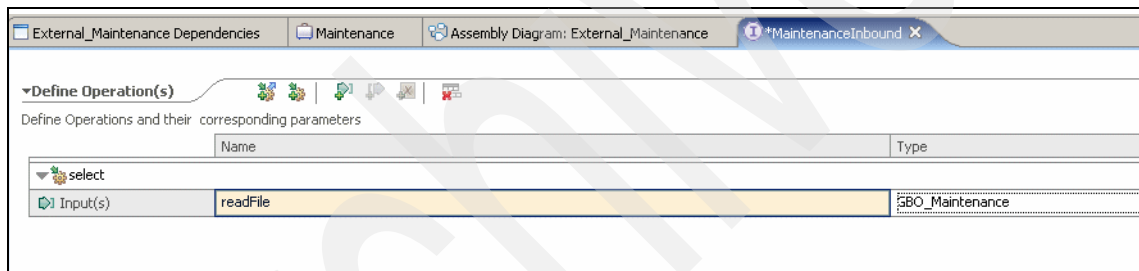
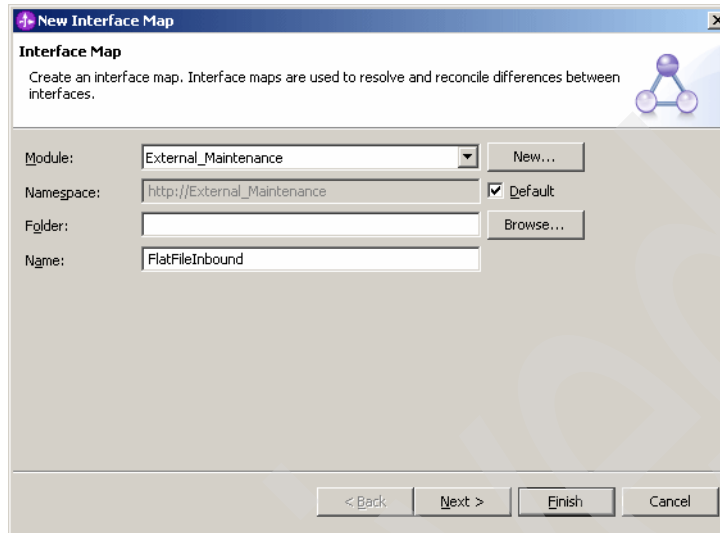


Figure 3-10 Add input

We now create a new interface map for conversion of ASBO to GBO:

1. Create a new interface map.

2. Enter a name (we chose FlatFileInbound). Click **Next** (Figure 3-11).



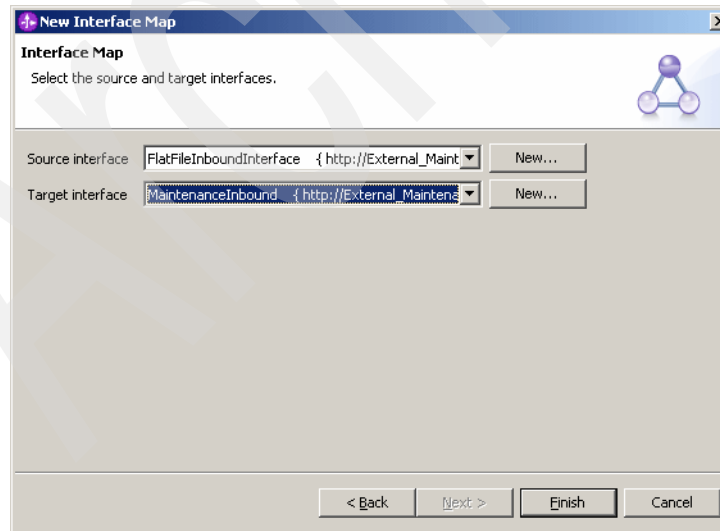
The 'New Interface Map' dialog box is shown. It has a title bar with a plus icon and a close button. Below the title bar is a section titled 'Interface Map' with a description: 'Create an interface map. Interface maps are used to resolve and reconcile differences between interfaces.' To the right of the text is a small icon of three nodes connected by lines. Below this is a form with the following fields and buttons:

- Module:** A dropdown menu showing 'External\_Maintenance' and a 'New...' button.
- Namespace:** A text box containing 'http://External\_Maintenance' and a checked 'Default' checkbox.
- Folder:** An empty text box and a 'Browse...' button.
- Name:** A text box containing 'FlatFileInbound'.

At the bottom of the dialog are four buttons: '< Back', 'Next >', 'Finish', and 'Cancel'.

Figure 3-11 New interface map

3. Select **FlatFileInboundInterface** as the source interface and **MaintenanceInbound** as the target interface.
4. Click **Finish** (Figure 3-12).



The 'New Interface Map' dialog box is shown again, but now it has a different title bar and content. The title bar has a plus icon and a close button. Below the title bar is a section titled 'Interface Map' with a description: 'Select the source and target interfaces.' To the right of the text is a small icon of three nodes connected by lines. Below this is a form with the following fields and buttons:

- Source interface:** A dropdown menu showing 'FlatFileInboundInterface' and a 'New...' button.
- Target interface:** A dropdown menu showing 'MaintenanceInbound' and a 'New...' button.

At the bottom of the dialog are four buttons: '< Back', 'Next >', 'Finish', and 'Cancel'.

Figure 3-12 Source and target interfaces

5. Create operation mapping between the source (READ operation) and target (select) interfaces (Figure 3-13).

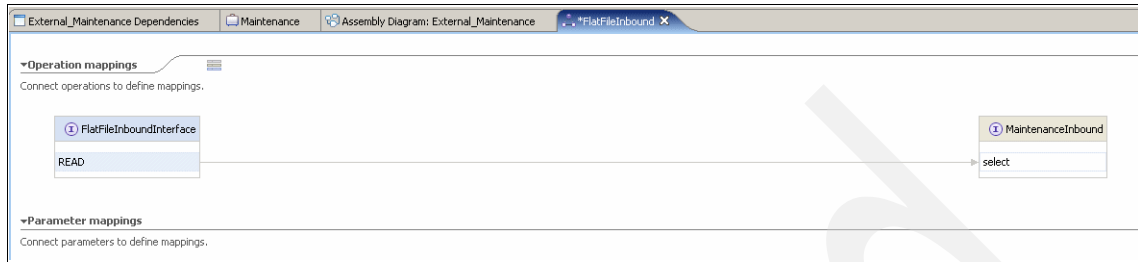


Figure 3-13 Create operation mapping

6. Create parameter mapping between source (READInputFlatFileBG) and target (readFile GBO\_Maintenance) parameters.
7. Select the mapping type of **map** from the General tab in the Properties view (Figure 3-14).

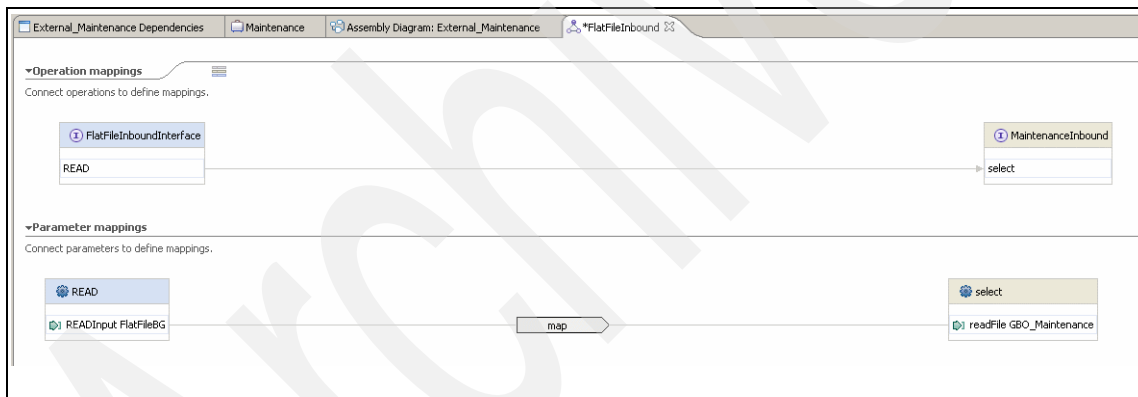


Figure 3-14 Select mapping type

8. Click **New** Business Object Map on the Advanced tab in the Properties view.

9. In the new window enter a business object map name (we chose FlatFileBG\_to\_GBO\_Maintenance). Click **Next** (Figure 3-15).

**New Business Object Map**

**Business Object Map**  
Create a new business object map. Business object maps are used to transform data between source and target business objects.

Module:  New...

Namespace:  ☒ Default

Folder:  Browse...

Name:

< Back Next > Finish Cancel

Figure 3-15 New business object map

10. Select input (**FlatFileBG**) and output (**GBO\_Maintenance**) types for data mapping.

11. Click **Finish** (Figure 3-16).

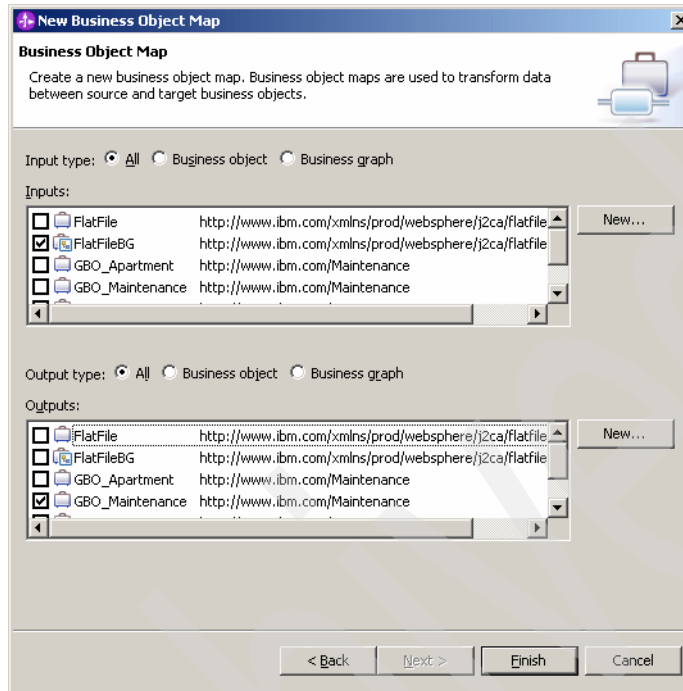


Figure 3-16 Select data types

12. The business object map is created. Save the interface map.
13. Open the new map for editing. As before, we use a variable on the type maintenance to hold the incoming data.
14. Add the new variable, name the variable (we chose `_Maintenance`), and select **Business Object Type**.
15. Browse business object type and select the **Maintenance** data type with qualifier `http://external.request.redmaint.com`.

16. Click **OK** (Figure 3-17).

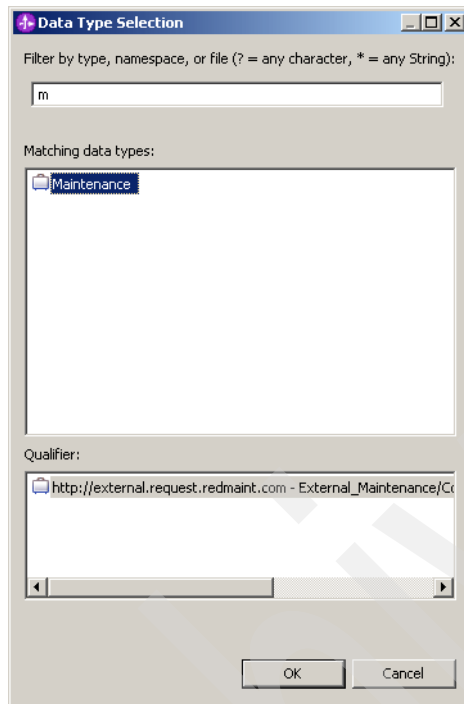


Figure 3-17 Select data type

17. Create a custom mapping between the inputBytes of FlatFileBG and the \_Maintenance object. This custom mapping will deserialize the XML response message to a business object by using the BOXMLSerializer API provided by the WebSphere Process Server (Figure 3-18).

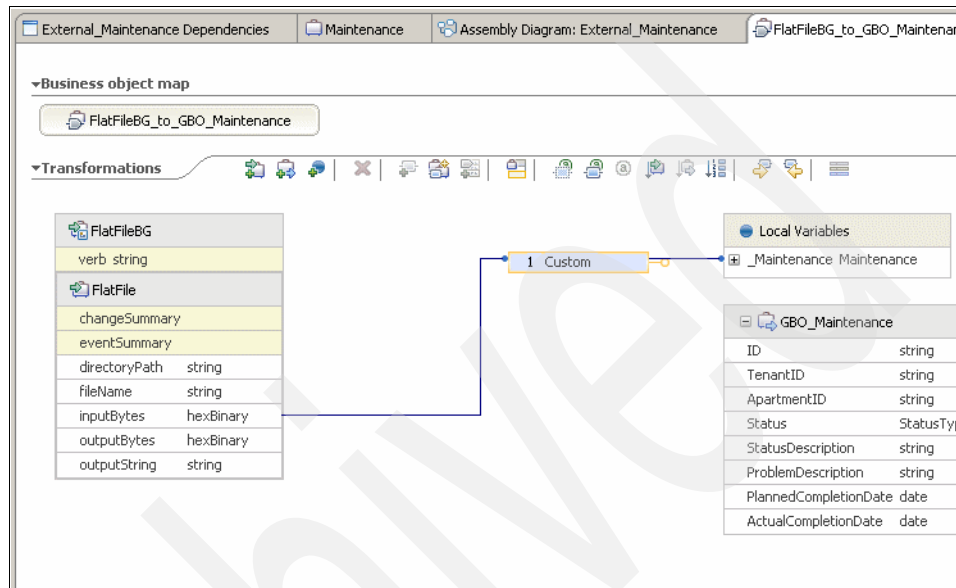


Figure 3-18 Select custom transformation

18. Add the Java code (as shown in Example 3-1) to perform the deserialization of data. (This code can also be found in Appendix A, “Additional material” on page 87.)

*Example 3-1 API code*

```
byte[] __FlatFileBG_FlatFile_inputBytes =
(byte[])FlatFileBG_FlatFile_inputBytes;

BOXMLSerializer xmlSerializerService = (BOXMLSerializer) new
    ServiceManager().locateService("com/ibm/websphere/bo/BOXMLSerializer");

ByteArrayInputStream input = new
    ByteArrayInputStream(__FlatFileBG_FlatFile_inputBytes);

BOXMLDocument boXmlFlatFile = xmlSerializerService.readXMLDocument(input);
_Maintenance = boXmlFlatFile.getDataObject();
```

19. Add the Java imports as shown in Example 3-2.

*Example 3-2 Java imports*

```
import java.io.ByteArrayInputStream;
import com.ibm.websphere.sca.ServiceManager;
import com.ibm.websphere.bo.BOXMLSerializer;
import com.ibm.websphere.bo.BOXMLDocument;
```

20. Save all changes.

21. Create the *move* transformations shown in Figure 3-19.

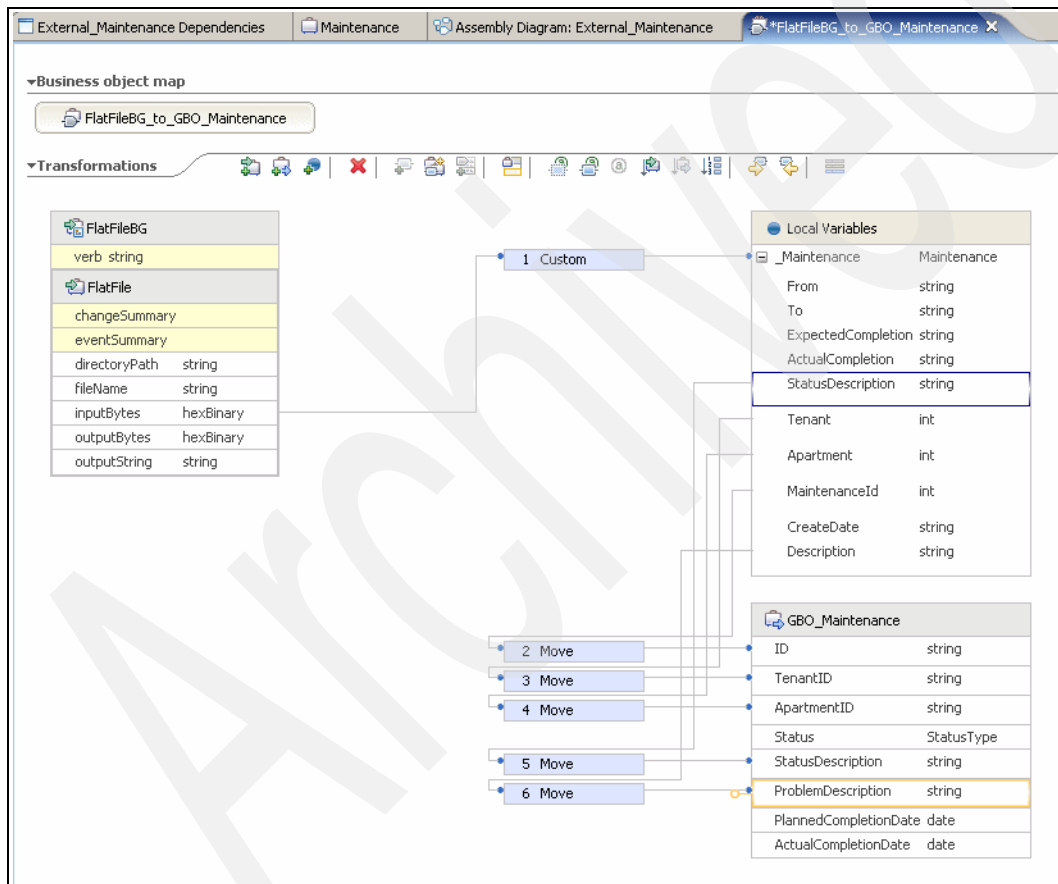


Figure 3-19 Move transformations

We now must create custom transformations for the dates. The dates that are sent in the XML file are string data, which represents the date in milliseconds. We must transform this string data to a date representation (which is the



reverse of what we did in 2.5, “Create interface map” on page 31). There are two transformations that must be performed, as our external contractor may have updated either of these values:

- ExpectedCompletion → PlannedCompletionDate
- ActualCompletion → ActualCompletionDate

22. Create a custom transformation between ExpectedCompletion and PlannedCompletionDate.

23. Using the visual snippet editor, we check that the ExpectedCompletion has a value and therefore requires transformation (that is, as it is a string, having a length that is not equal to zero). See Figure 3-20.

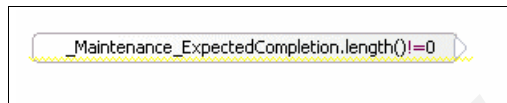


Figure 3-20 Check date exists

24. Add a choice to the palette (we only need to process the date if it exists, and therefore our condition is true).

25. Drag the \_Maintenance\_Expected Completion into the true condition using an expression (as this is the input to the transformation).

26. We then convert the string data to long data in readiness for converting to a date. Select the **parseLong(string)** method, as shown in Figure 3-21. Drop this in to the true condition.

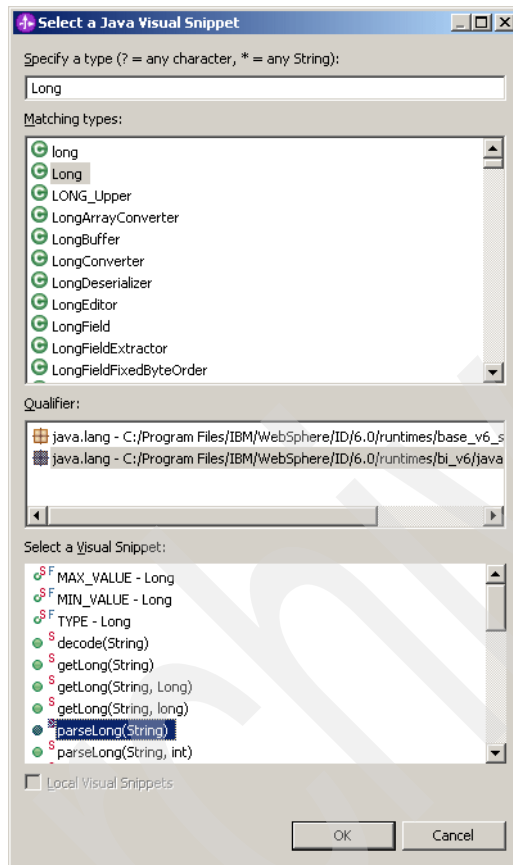


Figure 3-21 *parseLong(string)*

27. Select the **Date(long)** method, as shown in Figure 3-22, and drop this in to the true condition.

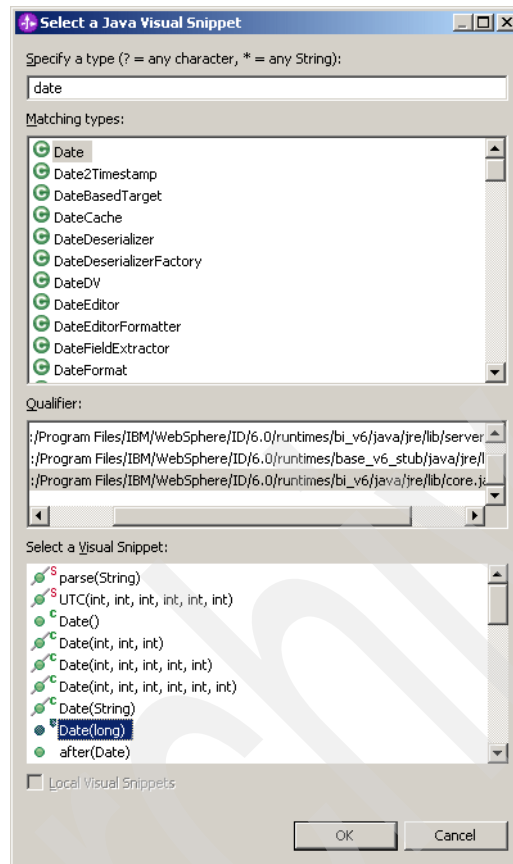


Figure 3-22 Date(long)

28. Add the PlannedCompletionDate of GBO\_Maintenance to the true condition using an expression.

29. Wire the conditional processing logic together, as shown in Figure 3-23.

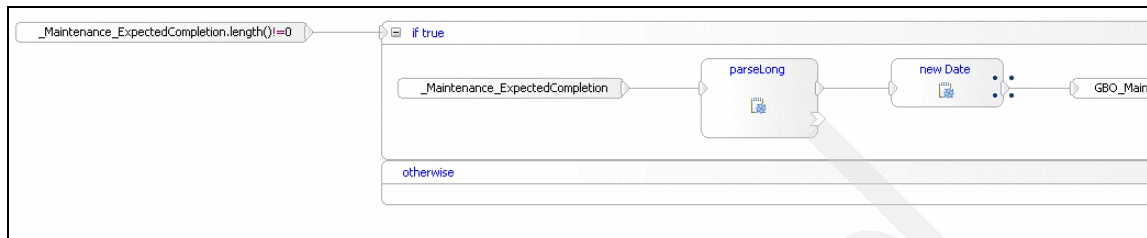


Figure 3-23 Transformation

30. Using the same steps, create a custom transform between ActualCompletion and ActualCompletionDate, as shown in Figure 3-24.



Figure 3-24 Actual date transformation

31. We now must add a custom transformation between ActualCompletion and Status. The external contractor does not send a status value to us, as the determination of status is made by us. For this transformation we must implement logic that will determine the status based on whether the ActualCompletion date has been set. If the date has been set, this means that the work is complete and we set the status to C (completed). If not we set the status to P (in progress).

32. Using an expression, check whether the ActualCompletion exists (as we did previously).

33. Add a condition:

- If the date that exists evaluates to true, set the status to C.
- Otherwise, set the status to P.

This is shown in Figure 3-25.

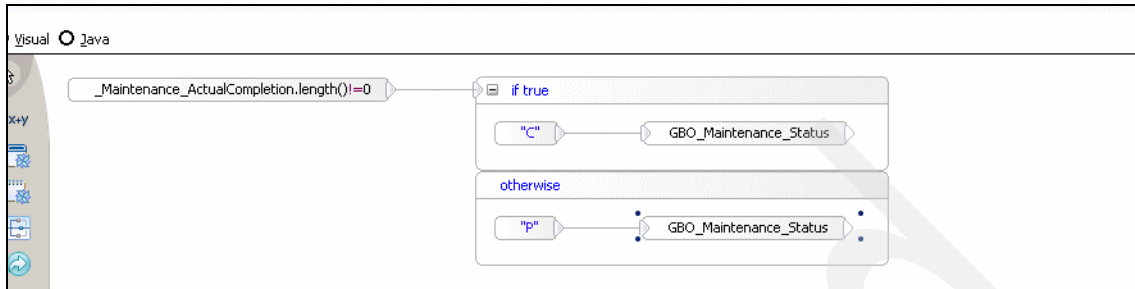


Figure 3-25 Set status

34. Save all changes.

We are now ready to assemble the inbound components.

### 3.3 Build the inbound processing of External\_Maintenance

In the assembly diagram:

1. Select the interface map **FlatFileInbound** and drag it to the assembly diagram.
2. Wire to the FlatFileInboundInterface.
3. Save all changes (Figure 3-26).

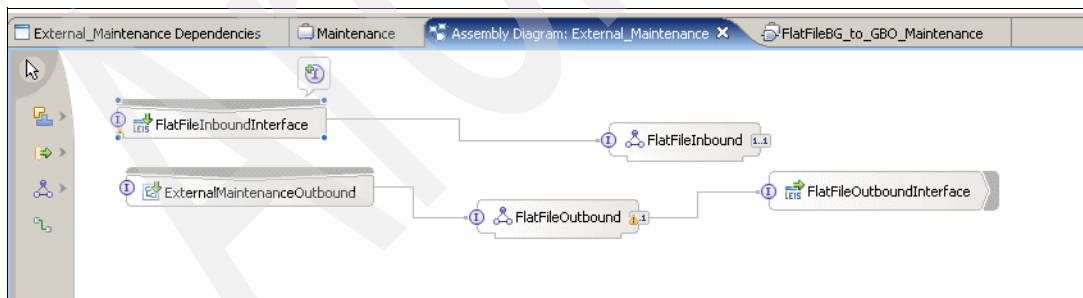


Figure 3-26 Assembly diagram

We are now ready to test the inbound processing.

## 3.4 Test the inbound file processing

There are two ways to test the inbound processing:

- ▶ Test using a Java component.
- ▶ Test by attaching to the module.

We explore both of these options.

### 3.4.1 Testing using a Java component

We start with the Java component:

1. Add a Java component to the assembly diagram (Figure 3-27).

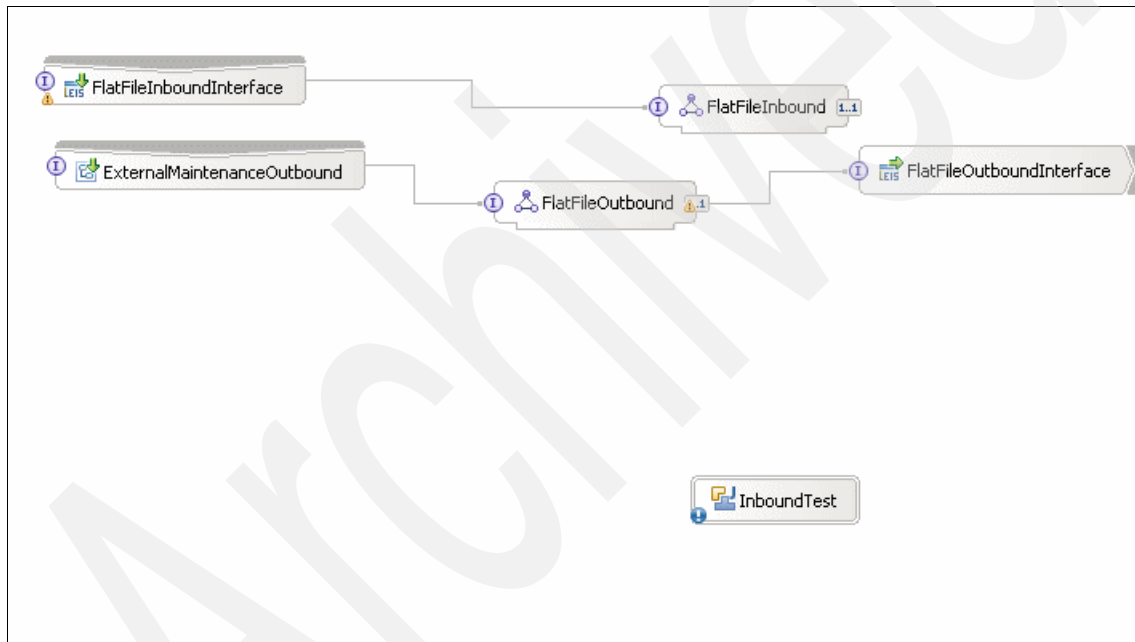


Figure 3-27 Add a Java component

2. Give the component a name (we chose InboundTest).

3. Add the interface MaintenanceInbound to the InboundTest Java component (Figure 3-28).

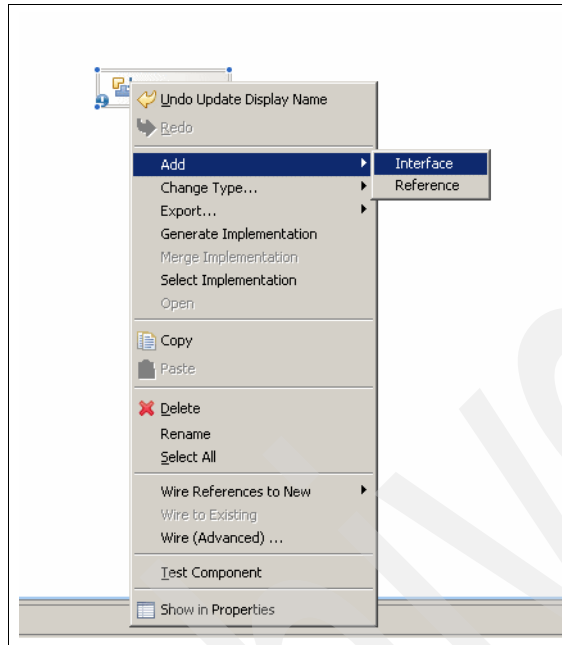


Figure 3-28 Add an interface

4. Wire the FlatFileInbound interface map to the InboundTest component.

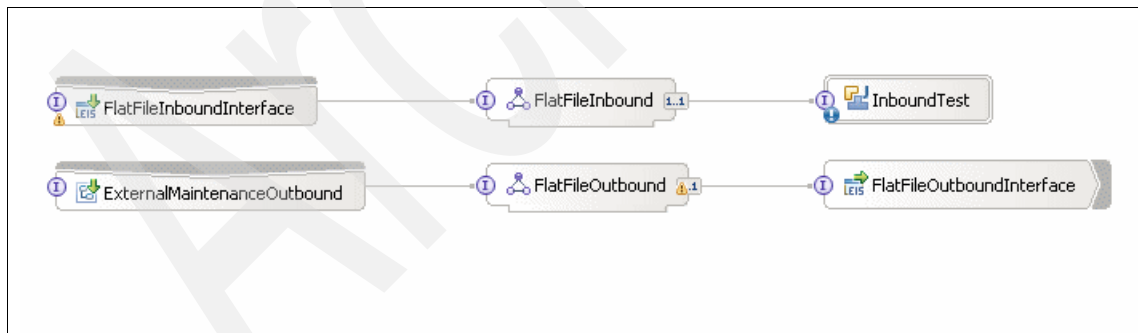


Figure 3-29 Wire interface map to component

5. Right-click the **InboundTest** component and select **Generate Implementation** (Figure 3-30).

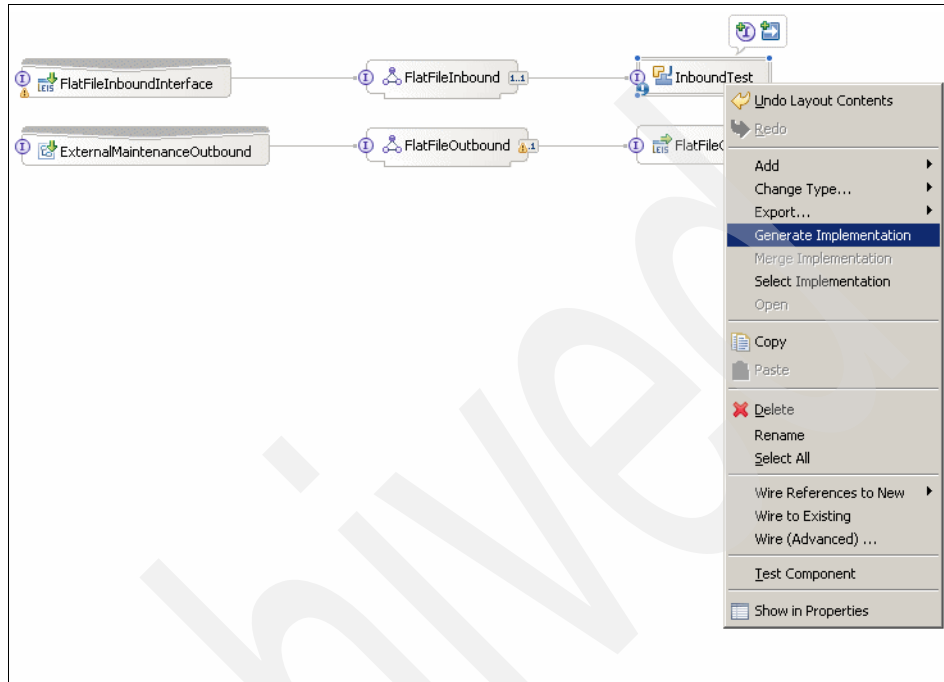


Figure 3-30 Generate implementation

6. Select **Default Package** and click **OK**.
7. We see the newly created TestImpl.java file with method select(DataObject readFile).
8. Add the code (shown in Example 3-3) in the select method to display polled and serialized values on the server console. (The code is also available in Appendix A, “Additional material” on page 87.)

Example 3-3 Java test code

```
public void select(DataObject readFile) {
    //TODO Needs to be implemented.
    System.out.println("***** Received new Data Object*****");
    System.out.println("*****");
    System.out.println("ID:"+readFile.getString("ID"));
    System.out.println("TenantID:"+readFile.getString("TenantID"));
    System.out.println("ApartmentID:"+readFile.getString("ApartmentID"));
    System.out.println("Status:"+readFile.getString("Status"));
    System.out.println("Status Description:"+readFile.getString("StatusDescription"));
    System.out.println("Problem Description:"+readFile.getString("ProblemDescription"));
}
```



```

        System.out.println("Planned Completion
Date:"+readFile.getString("PlannedCompletionDate"));
        System.out.println("Actual Completion
Date:"+readFile.getString("ActualCompletionDate"));
        System.out.println("*****");
    }

```

---

9. Save all changes.
10. Start the server.
11. Add the External\_MaintenanceApp project to the server.
12. Check for a message that the module has successfully started and has started polling (shown in Example 3-4).

#### *Example 3-4 Application and polling started*

---

```

Preparing to start EJB jar: External_MaintenanceEJB.jar
Starting EJB jar: External_MaintenanceEJB.jar
com.ibm.j2ca.base.WBIResourceAdapter startPolling() CWYBS0011I: Polling started
Loading Web Module: External_MaintenanceWeb.
Web Module External_MaintenanceWeb has been bound to default_host[*:9080,*:80,*:9443].
com.ibm.j2ca.extension.eventmanagement.EndpointRecovery recoverInProgressEvents CWYBS0500I:
Completed recovery
Application started: External_MaintenanceApp

```

---

13. Put the UpdateContractorResponse.xml file (there is a sample file in Appendix A, "Additional material" on page 87) in the event directory (C:\IBM\FlatFileIn). The adapter should be polling this directory, based on our configuration from 3.1, "Enterprise service discovery for inbound artifacts" on page 58.
14. Our test sample is shown in Example 3-5.

#### *Example 3-5 Test inbound file*

---

```

<?xml version="1.0" encoding="UTF-8"?>
<external:Maintenance xsi:type="external:Maintenance"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:external="http://external.request.redmaint.com">
  <external:From>redcontractor</external:From>
  <external:To>redmaint</external:To>
  <external:ExpectedCompletion>1150000000000</external:ExpectedCompletion>
  <external:ActualCompletion></external:ActualCompletion>
  <external:StatusDescription>This is a test for the FLAT FILE INBOUND - sent from the
contractors</external:StatusDescription>
  <external:Tenant>100</external:Tenant>
  <external:Apartment>1</external:Apartment>
  <external:MaintenanceId>23</external:MaintenanceId>
  <external:CreateDate>Today</external:CreateDate>

```

```
<external:Description>This is a description</external:Description>
</external:Maintenance>
```

---

This file will be picked up by the Flat File Adapter and output will be displayed on the console. Check that this displays deserialized values on the console and that the status has been set correctly to C or P. Our output is displayed in Example 3-6. Note that we also see the message telling us that an event delivery has occurred. In the context of an adapter, this means that something has happened that now requires some action.

#### *Example 3-6 Sample output*

---

```
ResourceAdapt A com.ibm.j2ca.extension.eventmanagement.EventSender deliverEvent() CWYBS0505I:
Event delivery occurred.
SystemOut      0 ***** Received new Data Object*****
SystemOut      0 *****
SystemOut      0 ID:23
SystemOut      0 TenantID:100
SystemOut      0 ApartmentID:1
SystemOut      0 Status:P
SystemOut      0 Status Description:This is a test for the FLAT FILE INBOUND - sent from the
contractors
SystemOut      0 Problem Description:This is a description
SystemOut      0 Planned Completion Date:2006-06-11
SystemOut      0 Actual Completion Date:null
SystemOut      0 *****
```

---

15. Try sending a sample to check that the status of complete will be correctly set according to the ActualCompletion (as seen in Example 3-7).

#### *Example 3-7 Test for completed*

---

```
<?xml version="1.0" encoding="UTF-8"?>
<external:Maintenance xsi:type="external:Maintenance"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:external="http://external.request.redmaint.com">
  <external:From>redcontractor</external:From>
  <external:To>redmaint</external:To>
  <external:ExpectedCompletion>1150000000000</external:ExpectedCompletion>
  <external:ActualCompletion>1155000000000</external:ActualCompletion>
  <external:StatusDescription>This is a test for the FLAT FILE INBOUND - sent from the
contractors</external:StatusDescription>
  <external:Tenant>100</external:Tenant>
  <external:Apartment>1</external:Apartment>
  <external:MaintenanceId>23</external:MaintenanceId>
  <external:CreateDate>Today</external:CreateDate>
  <external:Description>Testing now for COMPLETION STATUS</external:Description>
</external:Maintenance>
```

---

16. Check the output on the console (Example 3-8).

*Example 3-8 Console output for completed status*

---

```
ResourceAdapt A com.ibm.j2ca.extension.eventmanagement.EventSender deliverEvent() CWYBS0505I:
Event delivery occurred.
SystemOut      0 ***** Received new Data Object*****
SystemOut      0 *****
SystemOut      0 ID:23
SystemOut      0 TenantID:100
SystemOut      0 ApartmentID:1
SystemOut      0 Status:C
SystemOut      0 Status Description:This is a test for the FLAT FILE INBOUND - sent from the
contractors
SystemOut      0 Problem Description:Testing now for COMPLETION STATUS
SystemOut      0 Planned Completion Date:2006-06-11
SystemOut      0 Actual Completion Date:2006-08-08
SystemOut      0 *****
```

---

Go to the archive directory (C:\IBM\FlatFileArchive) and verify that the two input files have been moved to the archive location. Note that file names have been suffixed with:

- ▶ A time and date stamp
- ▶ The word PROCESSED

Our test is successful. We can now test using another method.

### 3.4.2 Test by attaching to the module

From the assembly diagram:

1. Right-click the **External\_Maintenance** module and click **Test** → **Attach** (Figure 3-31).

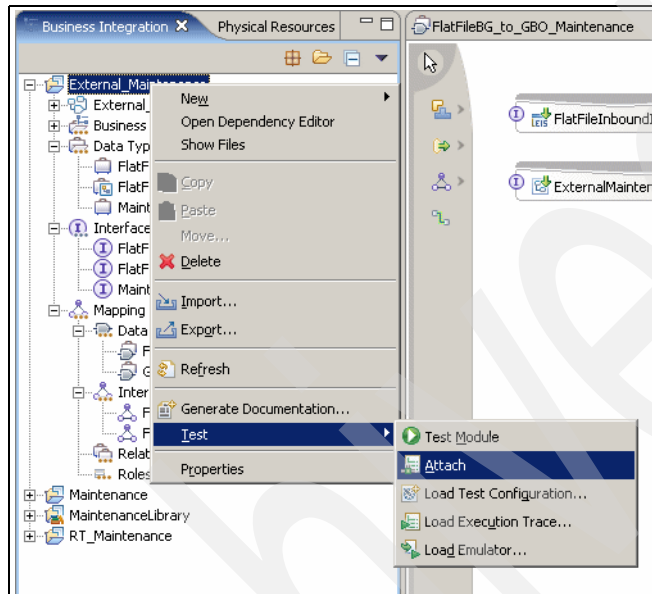


Figure 3-31 Attach

2. In the Events and Attach window, click **Continue** (Figure 3-32).

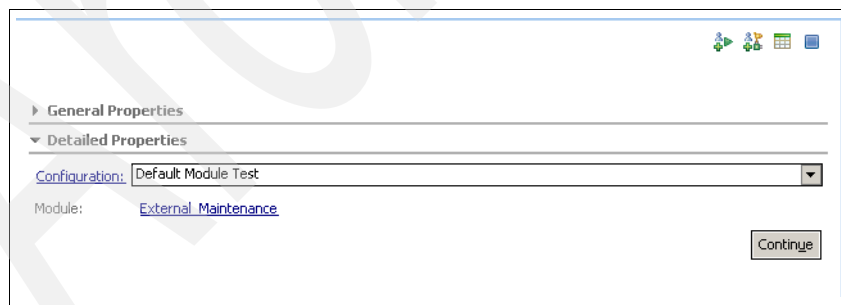


Figure 3-32 Events window

- Put a sample message into the C:\IBM\FlatFileIn directory. The adapter will poll this directory and the Attach window will receive the polled data as the converted business object (similar to that shown in Figure 3-33).

*Figure 3-33 Test results*

*Figure 3-34 Data object contents*

We can verify the results by comparing them to the output in the system log shown in Example 3-9.

*Example 3-9 Attach test in console*

---

```
ResourceAdapt A com.ibm.j2ca.extension.eventmanagement.EventSender deliverEvent() CWYBS0505I:
Event delivery occurred.
SystemOut    0 ***** Received new Data Object*****
SystemOut    0 *****
SystemOut    0 ID:23
SystemOut    0 TenantID:100
SystemOut    0 ApartmentID:1
SystemOut    0 Status:P
SystemOut    0 Status Description:This is an ATTACH TEST for the FLAT FILE INBOUND
SystemOut    0 Problem Description:This is a TEST
SystemOut    0 Planned Completion Date:2006-10-04
SystemOut    0 Actual Completion Date:null
SystemOut    0 *****
```

---

The External\_Maintenance module has now been successfully tested for both inbound and outbound file processing.

At this point we are now be ready to integrate this with any other processing, simply by removing the Java component and replacing it with an import component with an SCA binding for the module to which we want to pass.

## Additional material

This Redpaper refers to additional material that can be downloaded from the Internet as described below.

### Locating the Web material

The Web material associated with this Redpaper is available in softcopy on the Internet from the IBM Redbooks Web server. Point your Web browser to:

<ftp://www.redbooks.ibm.com/redbooks/REDP4235>

Alternatively, you can go to the IBM Redbooks Web site at:

[ibm.com/redbooks](http://ibm.com/redbooks)

Select the **Additional materials** and open the directory that corresponds with the redbook form number, REDP4235.

### Using the Web material

The additional Web material that accompanies this Redpaper includes the following files:

<i>File name</i>	<i>Description</i>
<b>REDP4235.zip</b>	Supporting materials and examples

## System requirements for downloading the Web material

The following system configuration is recommended:

<b>Hard disk space:</b>	20 MB minimum
<b>Operating System:</b>	Windows®
<b>Memory:</b>	1 GB

## How to use the Web material

Create a subdirectory (folder) on your workstation, and unzip the contents of the Web material zip file into this folder.







**Redpaper**

# A Simple Example: Using the WebSphere Adapter for Flat Files

**Integrate file-based  
applications into  
your SOA**

**Process inbound and  
outbound files**

**Explore serialization  
and deserialization  
of files**

WebSphere Adapters Version 6.0 provide a service-oriented approach to integration with Enterprise Information Systems (EIS).

WebSphere Adapters are compliant with J2EE Connector Architecture (JCA 1.5). JCA is the J2EE standard for EIS connectivity. EIS Import and EIS Export provide SCA components with the uniform view of the services external to the module. This allows components to communicate with a variety of external EIS systems using the consistent SCA programming model. WebSphere Adapters are assembled in WebSphere Integration Developer from imported RAR files and then exported as an Enterprise Application Archive (EAR) file and deployed on WebSphere ESB.

In this IBM Redpaper, we demonstrate a simple example of how to configure and use the WebSphere Adapter for Flat Files to create and read files for use in a service-oriented solution.

**INTERNATIONAL  
TECHNICAL  
SUPPORT  
ORGANIZATION**

**BUILDING TECHNICAL  
INFORMATION BASED ON  
PRACTICAL EXPERIENCE**

IBM Redbooks are developed by the IBM International Technical Support Organization. Experts from IBM, Customers and Partners from around the world create timely technical information based on realistic scenarios. Specific recommendations are provided to help you implement IT solutions more effectively in your environment.

**For more information:**  
[ibm.com/redbooks](http://ibm.com/redbooks)