Java CAPS 6/JBI and OpenESB
Using JBI, Note 4

File to File, with Java logic using Java EE SE

Michael Czapski, June 2008
Rev. 1.1

Release Notes
Both illustrations on Page 13 and Page 14 were incorrect in the original release - WSDL name shown in the illustrations, wsdlEJBwithOTD.wsdl, should have been wsdlJavaLogic.wsdl. As a consequence some names were incorrect. Thanks to Juraj Kazda for spotting this issue.

1 Introduction
This document explores the ability of Java CAPS 6/JBI and OpenESB to expose and execute Java-based logic as a JBI service. It walks through the process of creation, deployment and execution of a simple File-to-File integration solution that reads an XML record from a text file, invokes java logic that operates on that record then writes the XML response record into a file.

The focus is the practice of using JBI components not the theory of JBI.

This document addresses the integration solution developers, not developers of Service Engines or Binding Components.

The project does not use Java CAPS 6 Repository-based components, that's why it is just as good for OpenESB exploration as it is for Java CAPS 6/JBI exploration.

JBI (Java Business Integration) is not discussed to any great extent. JBI artifact names are used in discussion but not elaborated upon. Explanations are provided where necessary to foster understanding of the mechanics of developing integration solutions using JBI technologies in OpenESB and Java CAPS 6/JBI.

Java CAPS 6 and OpenESB are two of a number of toolkits that implement the JBI specification (JSR 208). When I use an expression like "In JBI …" I actually mean "In JBI as implemented in Java CAPS 6 and OpenESB …". The same things may well be implemented differently in other JBI toolkits.

Java CAPS 6 "Revenue Release" is used and shown in illustrations. OpenESB can be used instead however the appearance of components shown in illustrations may vary somewhat.

I use Windows to develop these solutions and make no effort to verify that the solutions will run on other platforms.

2 WSDLs

Java CAPS 6 and OpenESB use WSDL to define message structures and interactions between Binding Components (what in 5.x one would call OTDs and eWays or Adapters) and Service Units (what in 5.x one would call Java Collaborations and eInsight Business Processes). In 5.x WSDL was used for the same things but, unless one wanted to expose an eInsight Business Process as a web service, or consume a web service described by a WSDL, WSDL definitions were effectively invisible to the developer. This made the 5.x toolkit appear simpler to use, and conversely, made OpenESB and Java CAPS 6/JBI appear more complex and appear to require much deeper technical knowledge to work with when compared to Java CAPS 5.x.

In JBI WSDL is used to provide definitions of payload message structures that are exchanged between components and, in case of Binding Components, to provide the means to configure the Binding Component as required by the solution.

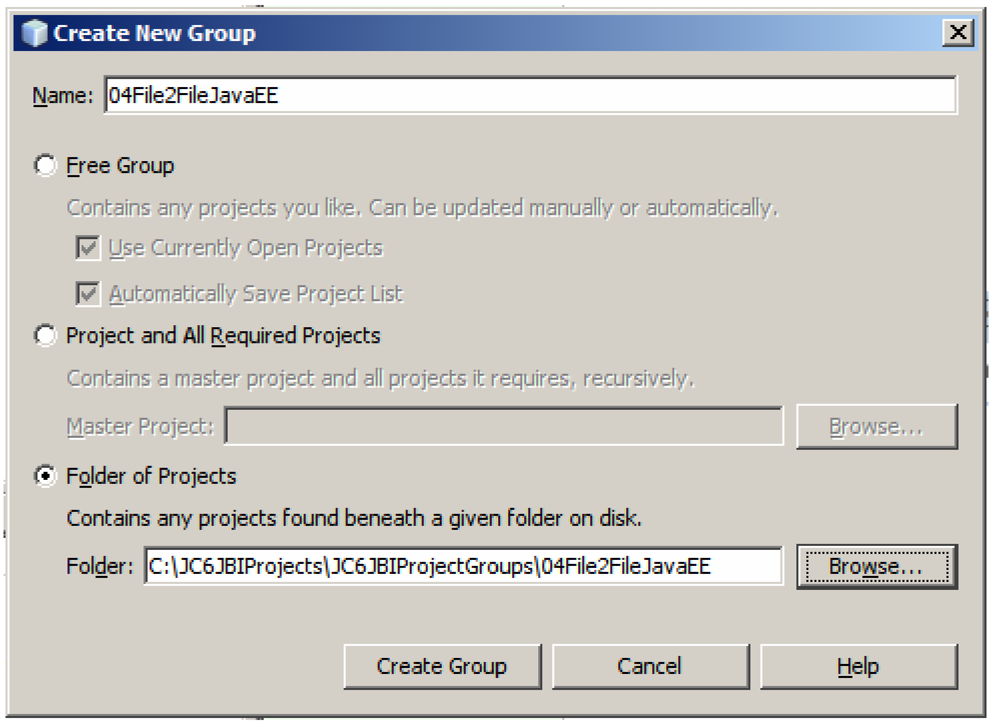
3 Create Project Group 04File2FileJavaEE

As on previous occasions, let’s create a new project group to contain projects we will be building in this Note. The group will be called “04File2FileJavaEE”.

Note

> Let’s make sure that the file system path, where the project group files will be located, does not contain spaces. If it does, we will have issues getting the XML Schema document import to work.

Let’s right-click anywhere in the Projects tab and choose Project Groups -> New Group … to start the wizard. Figure 3-1 shows the dialog box where project group name and file system directory path are specified.


Create New Group

Name: 04File2FileJavaEE

( ) Free Group

Contains any projects you like. Can be updated manually or automatically.

[✓] Use Currently Open Projects

[✓] Automatically Save Project List

( ) Project and All Required Projects

Contains a master project and all projects it requires, recursively.

Master Project:

(●) Folder of Projects

Contains any projects found beneath a given folder on disk.

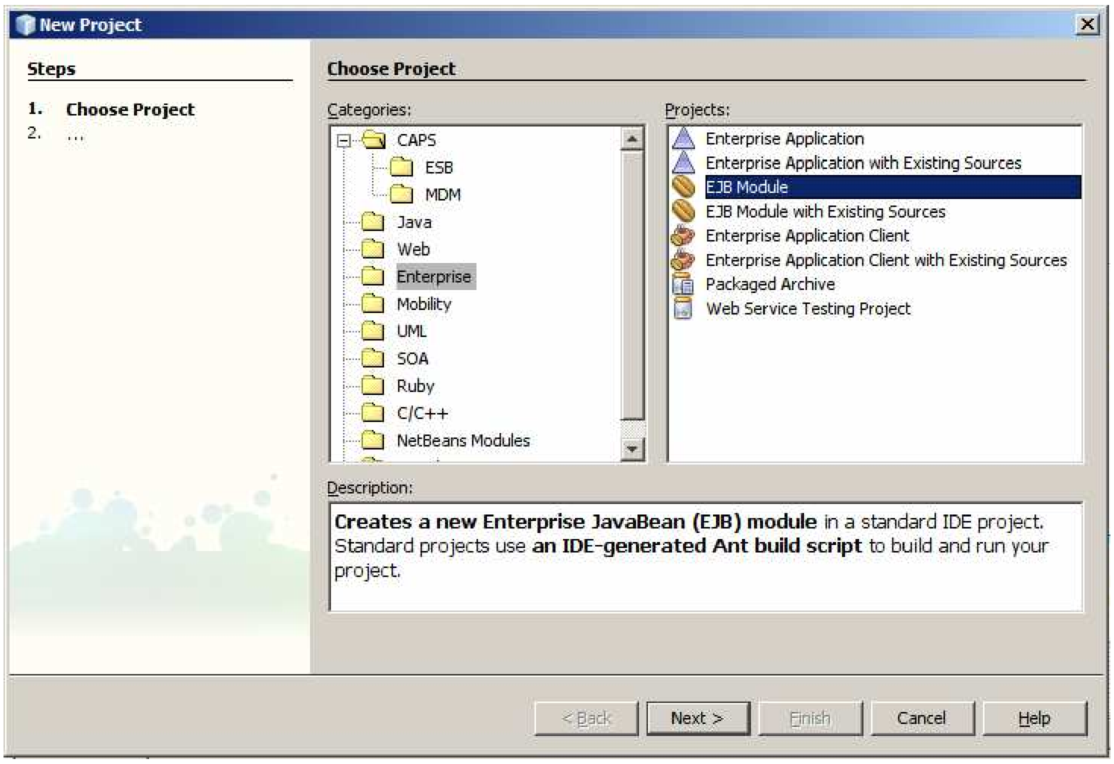
Folder: C:\JC6JBIProjects\JC6JBIProjectGroups\04File2FileJavaEE

Figure 3-1 Creating a new Project Group

4 Java logic

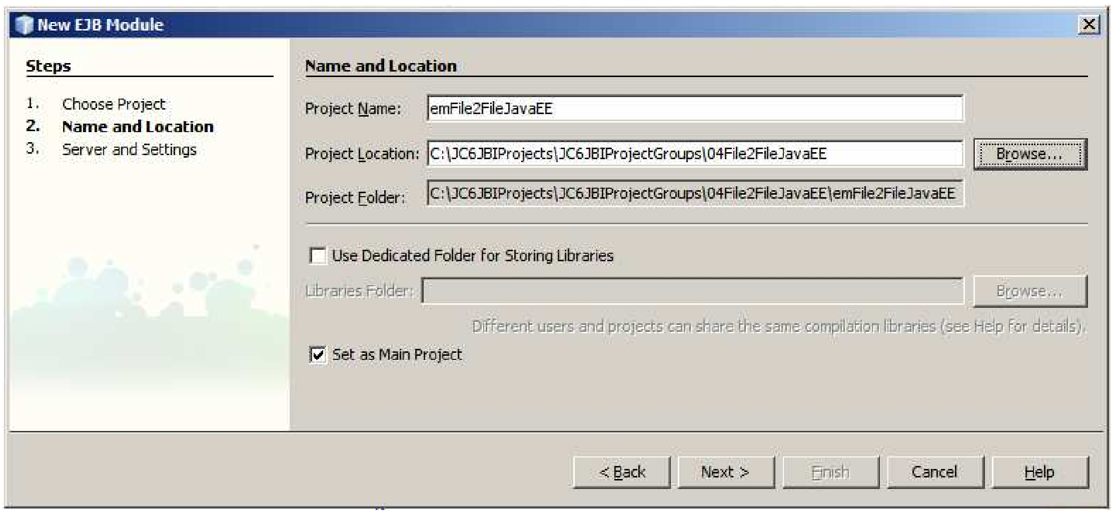
Java logic will be implemented in a Java class annotated as a Web Service. Don't panic – there is no SOAP over HTTP involved. The WSDL, which we will created, will consist of the Interface, or Abstract part, only and will be used, as mentioned in Section 2, to name the service's operation and describe input and output messages. The service itself will be hosted in the JBI container and will be invoked by the NMR.

Let's create a new project, of category Enterprise, of type EJB Module, Figure 4-1, named emFile2FileJavaEE, making sure to select the correct project group folder, Figure 4-2.



Creates a new Enterprise JavaBean (EJB) module in a standard IDE project. Standard projects use **an IDE-generated Ant build script** to build and run your project.

Figure 4-1 Enterprise -> EJB Module project



Project Name: emFile2FileJavaEE

Project Location: C:\JC6JBIProjects\JC6JBIProjectGroups\04File2FileJavaEE

Project Folder: C:\JC6JBIProjects\JC6JBIProjectGroups\04File2FileJavaEE\emFile2FileJavaEE

☐ Use Dedicated Folder for Storing Libraries

Libraries Folder:

Different users and projects can share the same compilation libraries (see Help for details).

☑ Set as Main Project

Figure 4-2 Naming the project and choosing project group folder

Since WSDL is the King, and WSDL uses XML for input and output messages, and using XSD built-in string data type is fairly uninspiring, let's create in our EJB Module folder, emFile2FileJavaEE, an XML Schema document to describe a structure that is slightly more complex then the xsd:string.

Let's create a structure consisting of two element, elString, of type xsd:string and elInteger of type xsd:integer. Notation xsd:xxxx refers to XML Schema built-in data types.

The steps to create a simple XML Schema with two elements is described and illustrated in Figures 4-3 through 4-12.

Right-click on the name of the EJB Module, emFile2FileJavaEE, and choose New -> XML Schema … as shown in Figure 4-3.

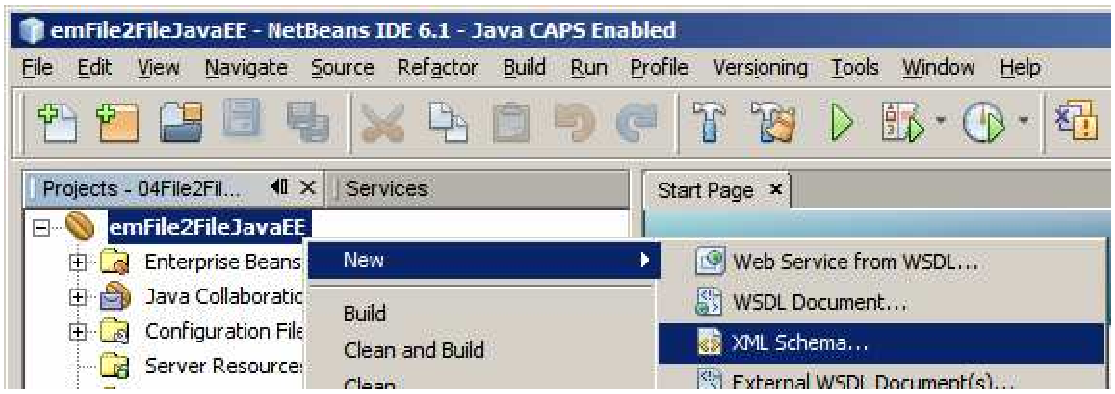
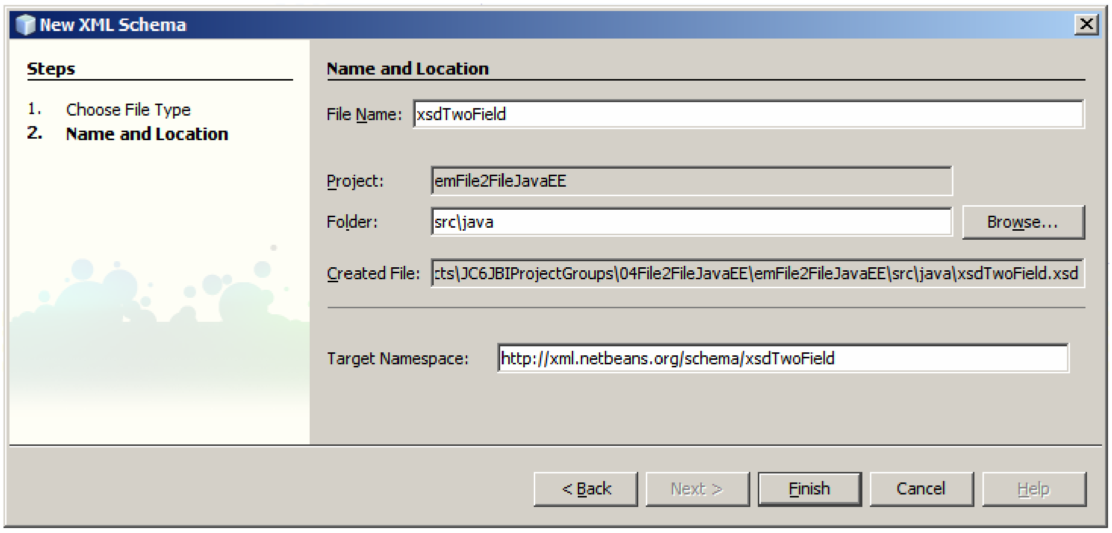


Figure 4-3 Choosing ot create a XML Schema

Let's name the Schema xsdTwoFields, as illustrated in Figure 4-4.


New XML Schema

Steps
1. Choose File Type
2. Name and Location

Name and Location

File Name: xsdTwoField

Project: emFile2FileJavaEE

Folder: src\java

Created File: :cts\JC6JBIProjectGroups\04File2FileJavaEE\emFile2FileJavaEE\src\java\xsdTwoField.xsd

Target Namespace: http://xml.netbeans.org/schema/xsdTwoField

Figure 4-4 Naming the new XML Schema

Make sure the XSD Editor is in the Schema mode, that is that the Schema Tab is selected. Right click on the "Elements" node and choose Add Element … as illustrated in Figure 4-5.

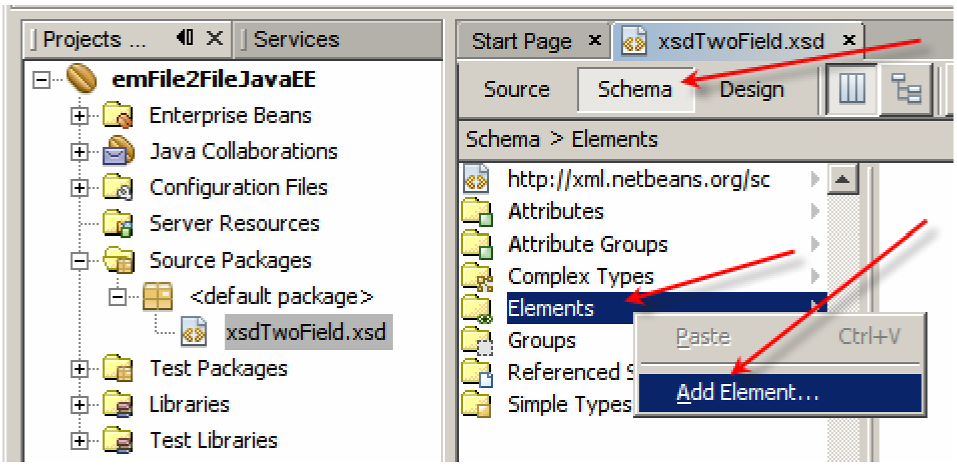
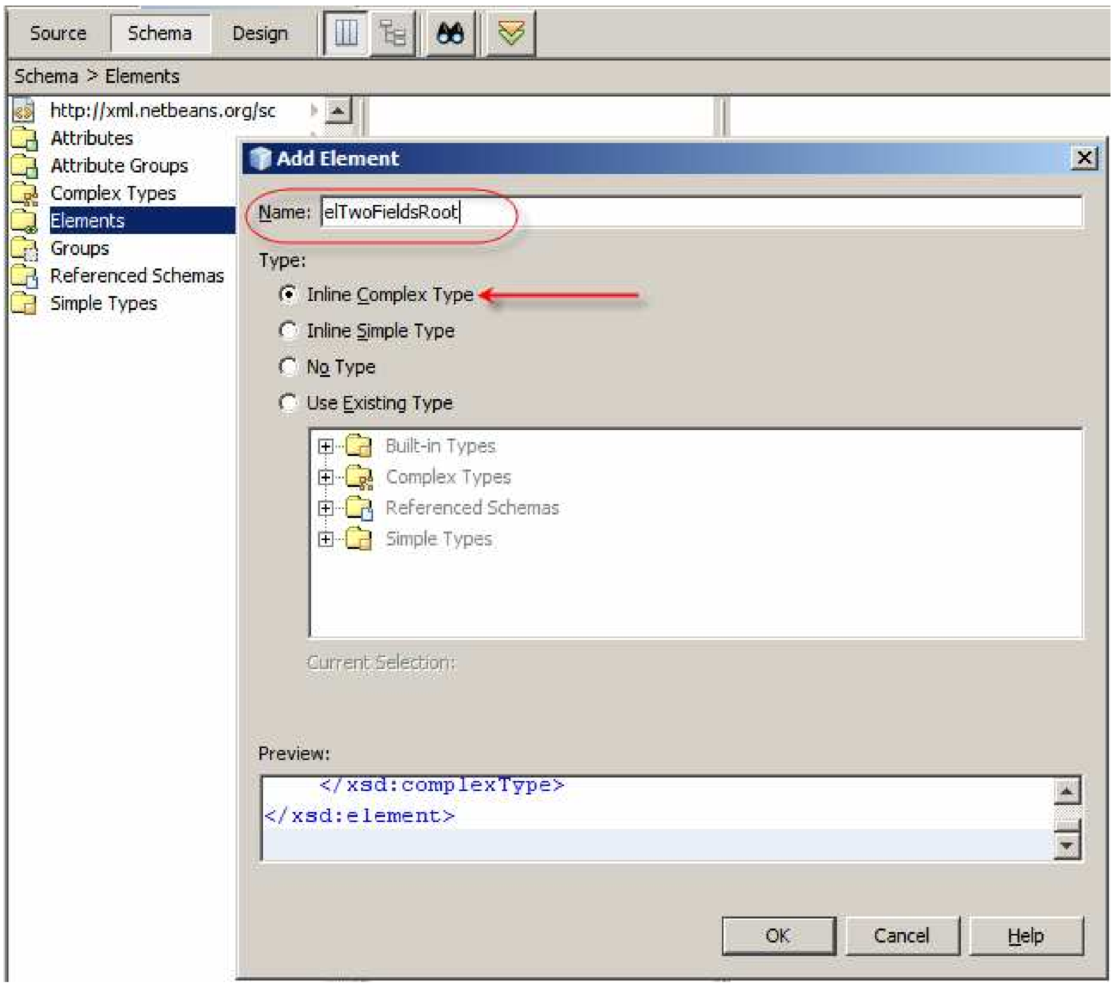


Figure 4-5 Choosing to add a new element

Let's name this element elTwoFieldsRoot and make sure to keep the radio button "Inline Complex Type" selected, as illustrated in Figure 4-6.



Name: elTwoFieldsRoot

Type:

[✓ Inline Complex Type]
Inline Simple Type
No Type
Use Existing Type

Preview:

```
    </xsd:complexType>
</xsd:element>
```

Figure 4-6 Adding root element to the schema

Click on names in successive columns, from left to right, until the "sequence" is shown, then right click on the "sequence" and choose Add -> Element. Figure 4-7 illustrates this.

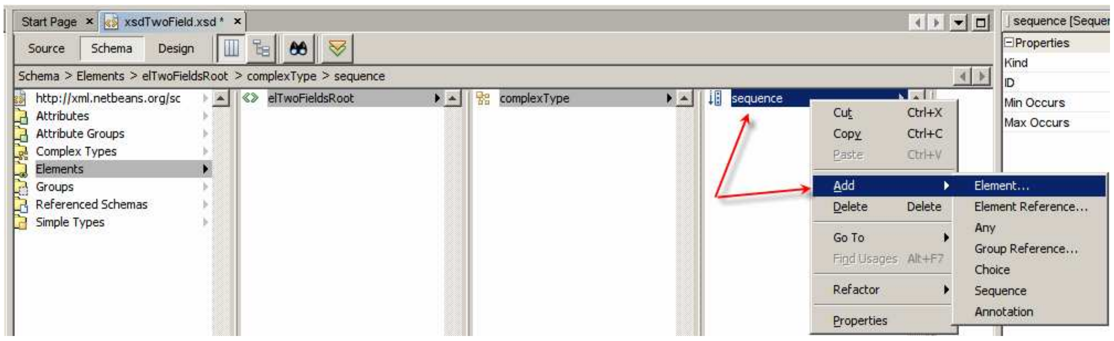
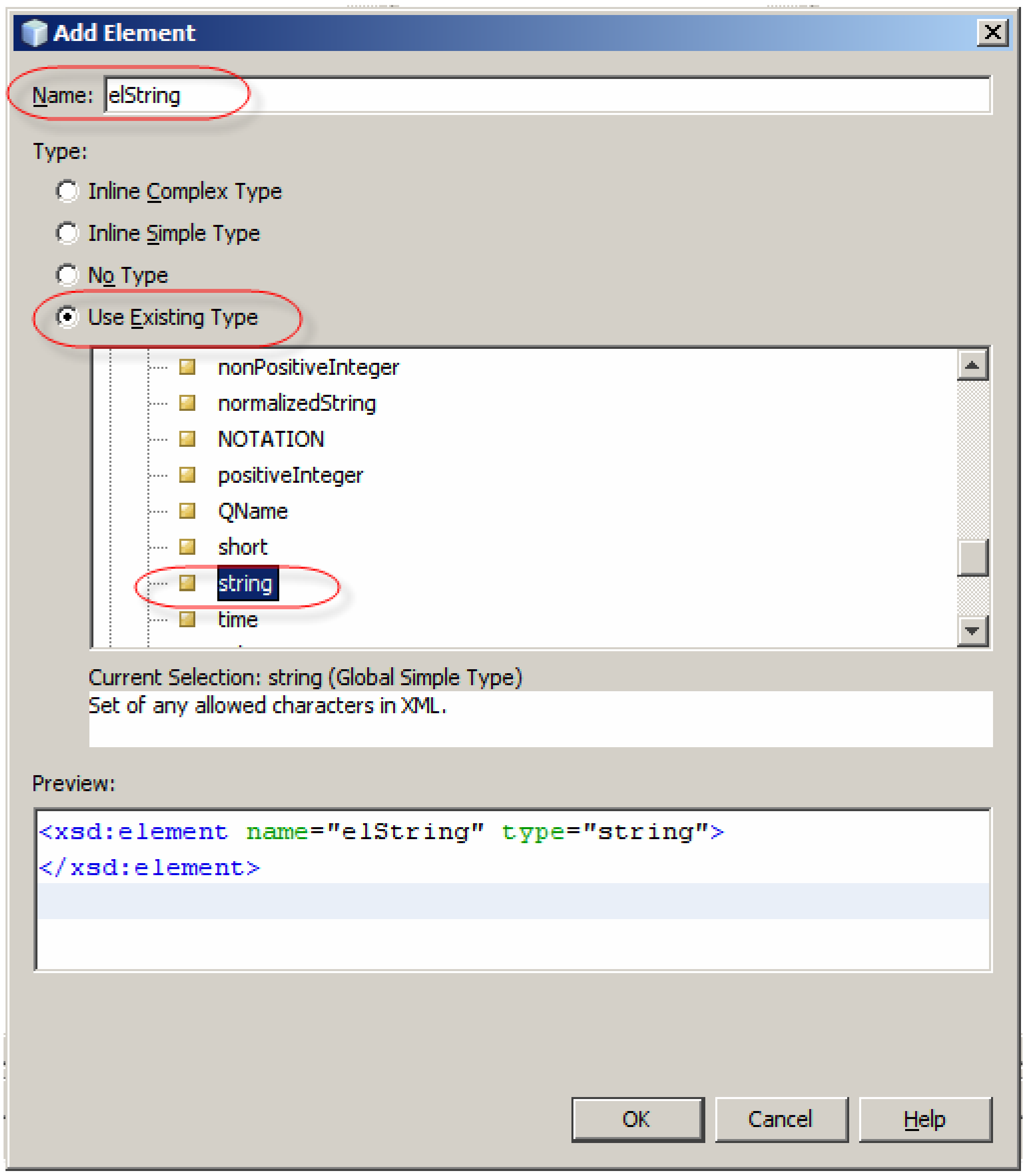


Figure 4-7 Choosing to add an element to the sequence

Name this new element elString, choose the Use Existing Type, scroll through the list of Built-In types and select "string", as shown in Figure 4-8.



Name: elString

Type:

- Inline Complex Type
- Inline Simple Type
- No Type
- [✓ Use Existing Type]

nonPositiveInteger
normalizedString
NOTATION
positiveInteger
QName
short
[✓ string]
time

Current Selection: string (Global Simple Type)
Set of any allowed characters in XML.

Preview:

```
<xsd:element name="elString" type="string">
</xsd:element>
```

Figure 4-8 Adding an element of type xsd:string

Right-click on the "sequence" again and add another element, elInteger, using existing built-in type "integer". Figures 4-9 and 4-10 illustrate this.

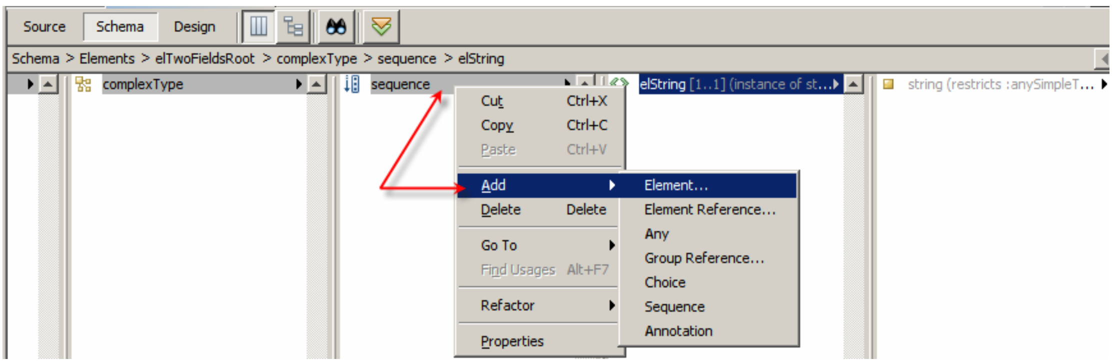
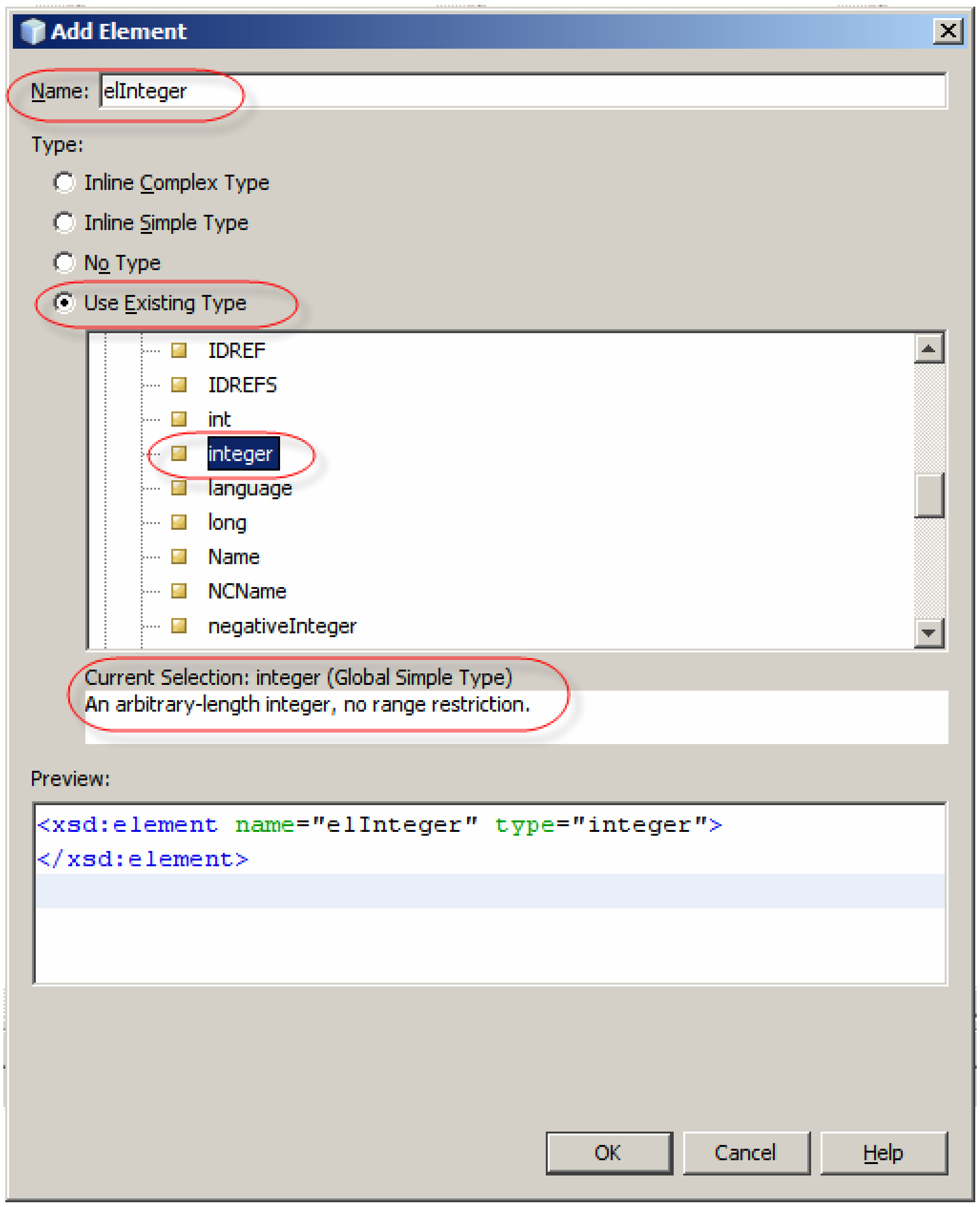


Figure 4-9 Choosing to add another element



Add Element

Name: elInteger

Type:

- Inline Complex Type
- Inline Simple Type
- No Type
- [✓ Use Existing Type]

Current Selection: integer (Global Simple Type)
An arbitrary-length integer, no range restriction.

Preview:

```
<xsd:element name="elInteger" type="integer">
</xsd:element>
```

Figure 4-10 Adding a xsd:integer type – this will become java.math.BigInteger

The complete XSD Schema is shown in Schema mode in Figure 4-11 and in Source mode in Figure 4-12.

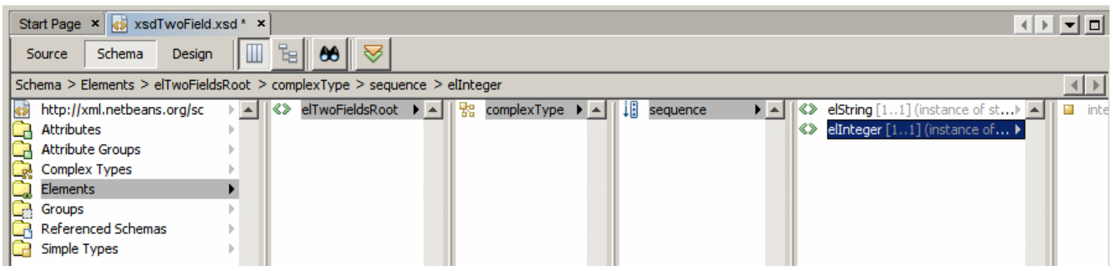
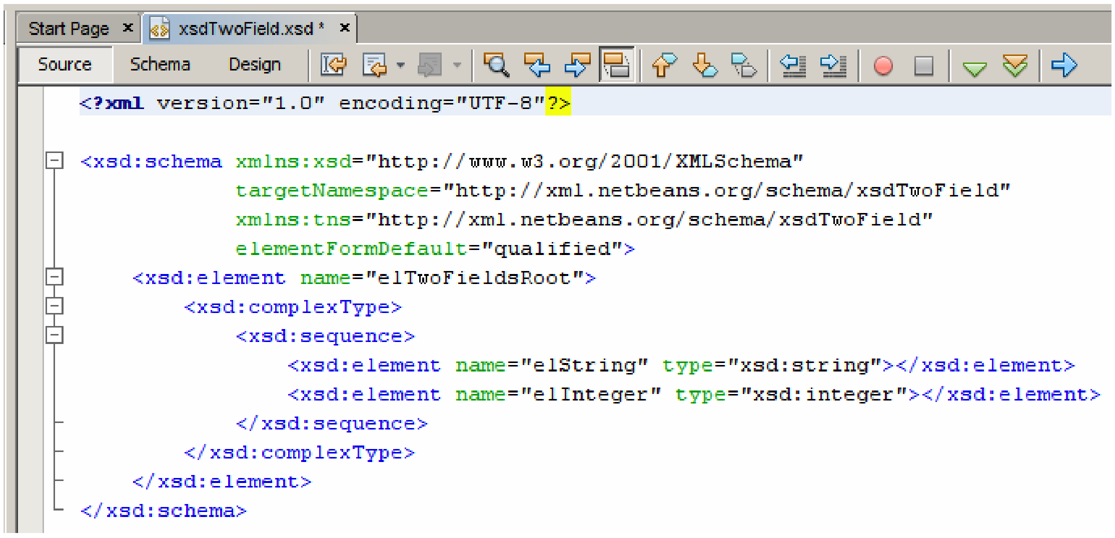


Figure 4-11 XML Schema in Schema mode



```
<?xml version="1.0" encoding="UTF-8"?>

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            targetNamespace="http://xml.netbeans.org/schema/xsdTwoField"
            xmlns:tns="http://xml.netbeans.org/schema/xsdTwoField"
            elementFormDefault="qualified">
    <xsd:element name="elTwoFieldsRoot">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="elString" type="xsd:string"></xsd:element>
                <xsd:element name="elInteger" type="xsd:integer"></xsd:element>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
</xsd:schema>
```

Figure 4-12 XML Schema in Source mode

Our XML Schema is now ready. We can create a WSDL to describe the interface between the EJB Module we are creating and the rest of the world.

Let’s create a new WSDL, wsdlJavaLogic, Figure 4-13, import the XML Schema we just created, Figures 4-14 and 4-15, then name the WSDL operation opJavaLogic, the input message sIStruct, and the output message sOStruc, Figure 4-16. Before completing the wizard we need to make sure that the input and output messages are of the correct type, elTwoFieldsRoot, which we obtained when we imported the XML Schema created earlier. Figures 4-17 and 4-18 illustrate the steps in choosing the type for the input message. The steps are the same for the output message, as is the message type.

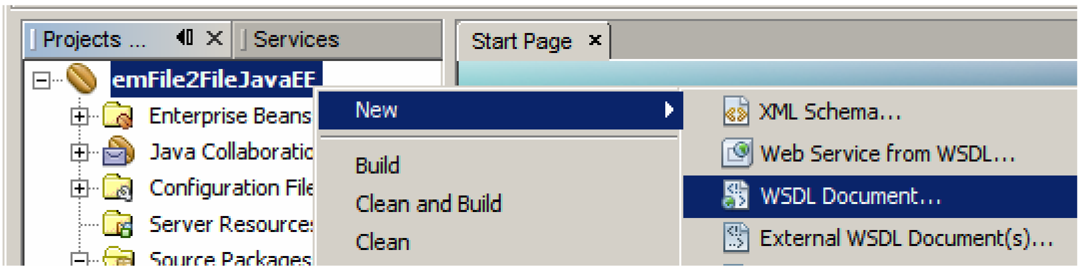
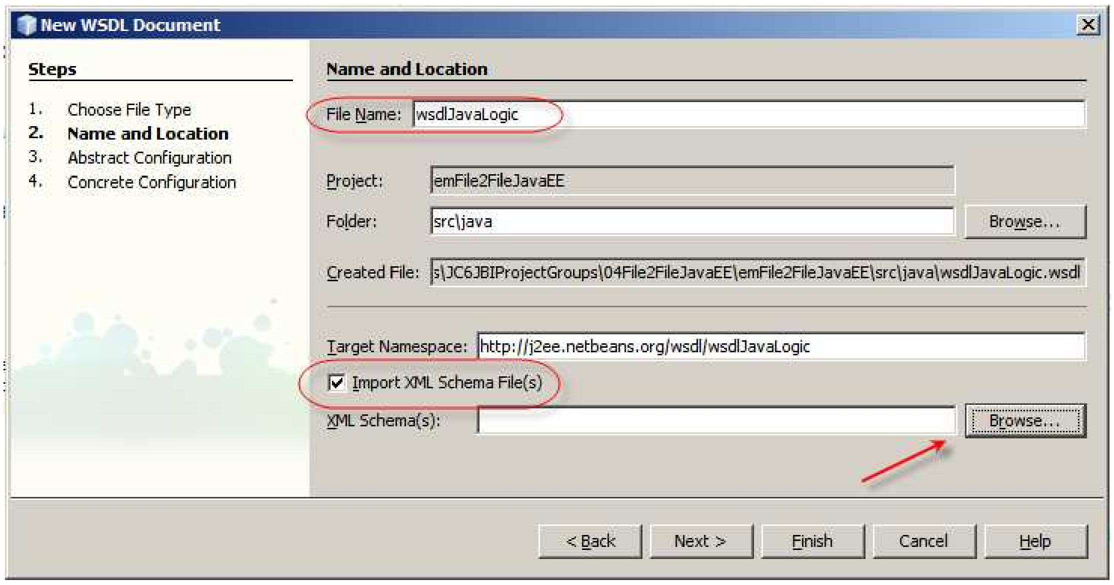


Figure 4-13 Choosing to create a new WSDL document


New WSDL Document

Steps

1. Choose File Type
2. **Name and Location**
3. Abstract Configuration
4. Concrete Configuration

Name and Location

File Name: wsdlJavaLogic

Project: emFile2FileJavaEE

Folder: src\java

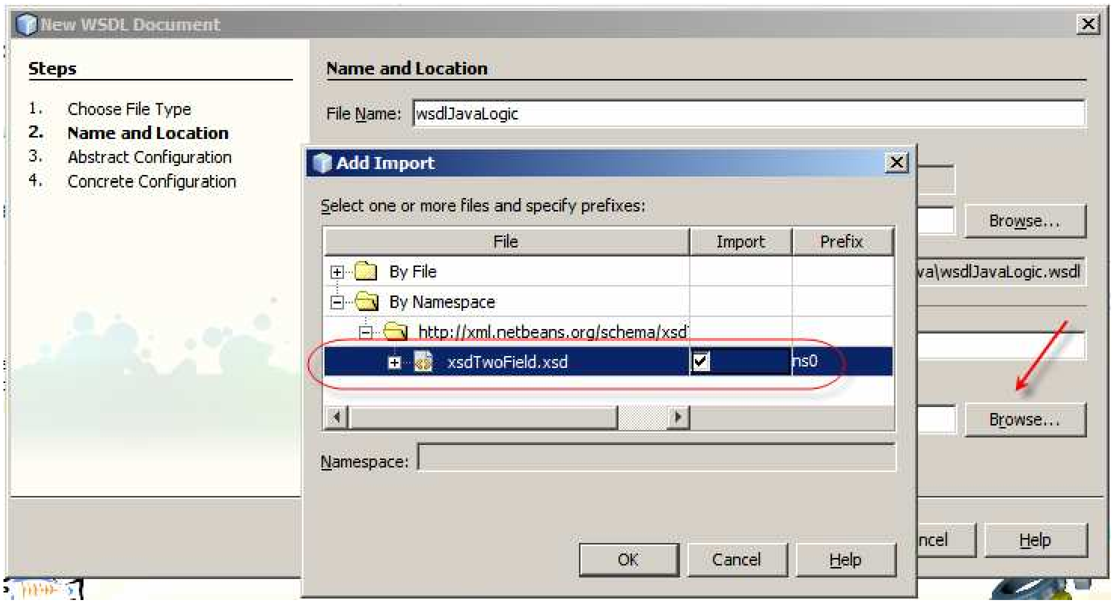
Created File: s\JC6JBIProjectGroups\04File2FileJavaEE\emFile2FileJavaEE\src\java\wsdlJavaLogic.wsdl

Target Namespace: http://j2ee.netbeans.org/wsdl/wsdlJavaLogic

☑ Import XML Schema File(s)

XML Schema(s):

Figure 4-14 Naming the WSDL and starting import of the XML Schema


New WSDL Document

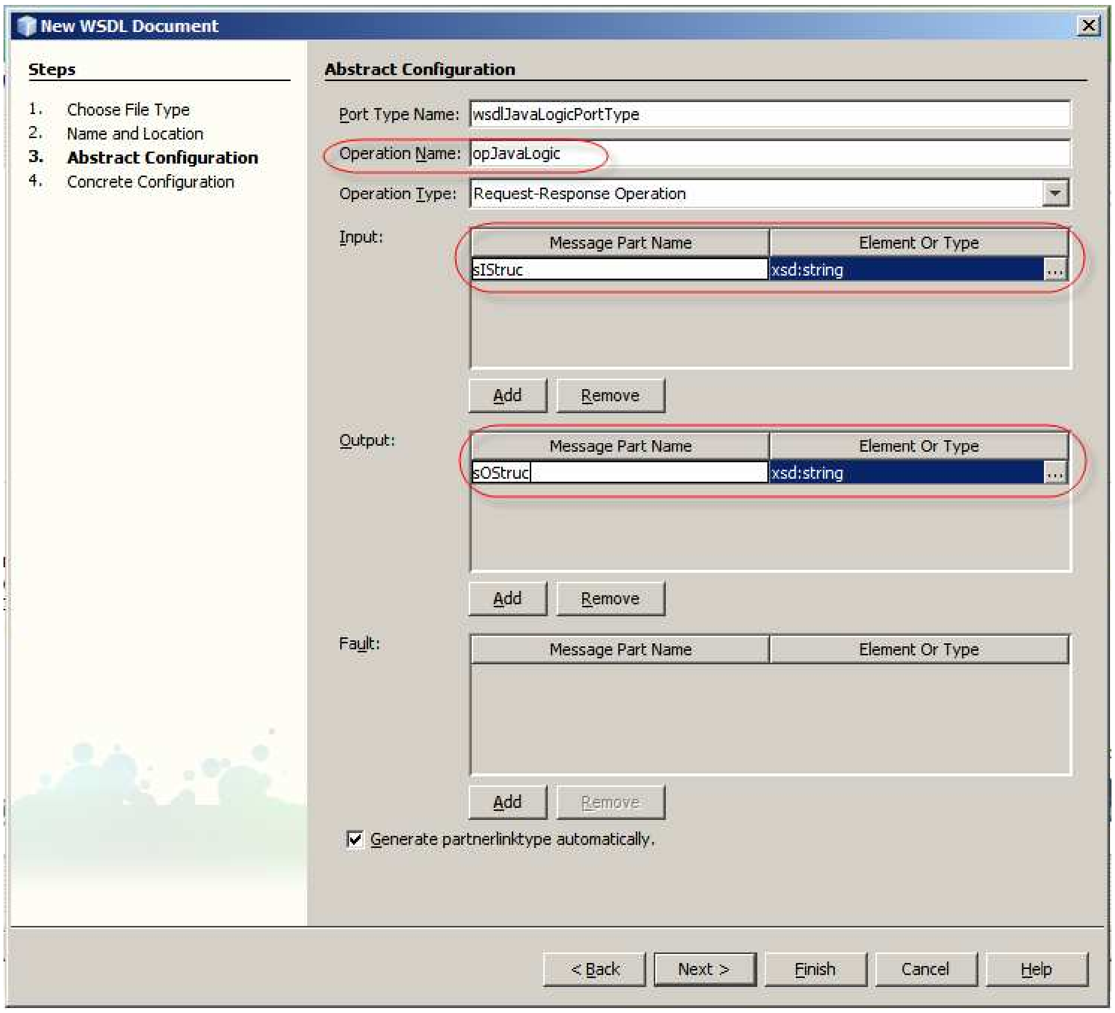
Add Import

Select one or more files and specify prefixes:

File	Import	Prefix
By File		
By Namespace		
http://xml.netbeans.org/schema/xsd		
xsdTwoField.xsd	✓	ns0

Namespace:

Figure 4-15 Choosing XML Schema to import



Port Type Name: wsdlJavaLogicPortType

Operation Name: opJavaLogic

Operation Type: Request-Response Operation

Input:

Message Part Name	Element Or Type
sIStruc	xsd:string

Output:

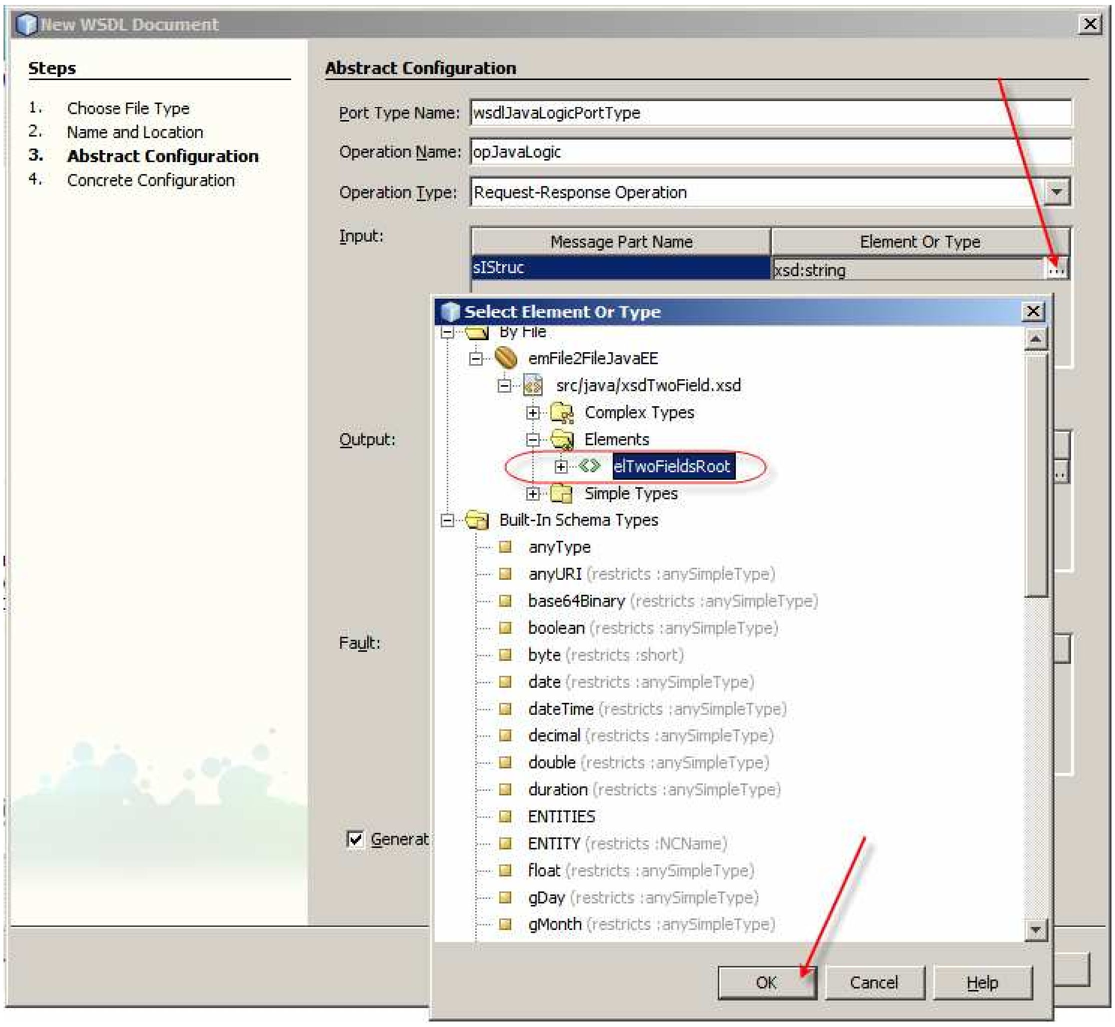
Message Part Name	Element Or Type
sOStruc	xsd:string

Fault:

Message Part Name	Element Or Type

[✓] Generate partnerlinktype automatically.

Figure 4-16 Naming the operation and messages


New WSDL Document

Steps

1. Choose File Type
2. Name and Location
3. **Abstract Configuration**
4. Concrete Configuration

Abstract Configuration

Port Type Name: wsdlJavaLogicPortType

Operation Name: opJavaLogic

Operation Type: Request-Response Operation

Input:

Message Part Name	Element Or Type
sIStruc	xsd:string

Output:

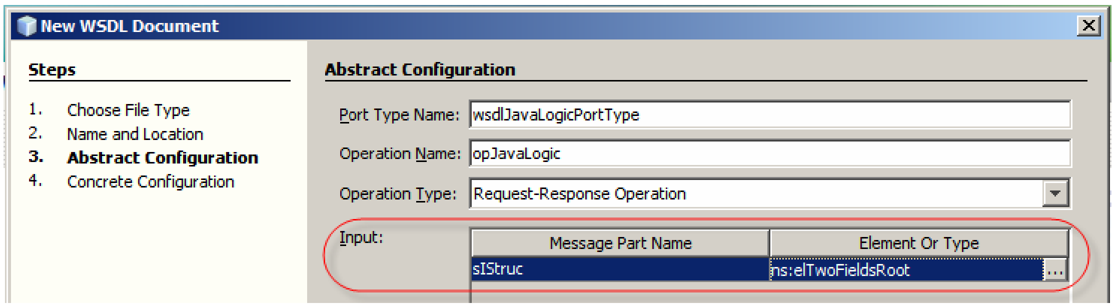
Fault:

☑ Generat

Select Element Or Type

- By File
  - emFile2FileJavaEE
    - src/java/xsdTwoField.xsd
      - Complex Types
      - Elements
        - elTwoFieldsRoot
      - Simple Types
- Built-In Schema Types
  - anyType
  - anyURI (restricts :anySimpleType)
  - base64Binary (restricts :anySimpleType)
  - boolean (restricts :anySimpleType)
  - byte (restricts :short)
  - date (restricts :anySimpleType)
  - dateTime (restricts :anySimpleType)
  - decimal (restricts :anySimpleType)
  - double (restricts :anySimpleType)
  - duration (restricts :anySimpleType)
  - ENTITIES
  - ENTITY (restricts :NCName)
  - float (restricts :anySimpleType)
  - gDay (restricts :anySimpleType)
  - gMonth (restricts :anySimpleType)

Figure 4-17 Choosing the message type for the input message


New WSDL Document

Steps

1. Choose File Type
2. Name and Location
3. **Abstract Configuration**
4. Concrete Configuration

Abstract Configuration

Port Type Name: wsdlJavaLogicPortType

Operation Name: opJavaLogic

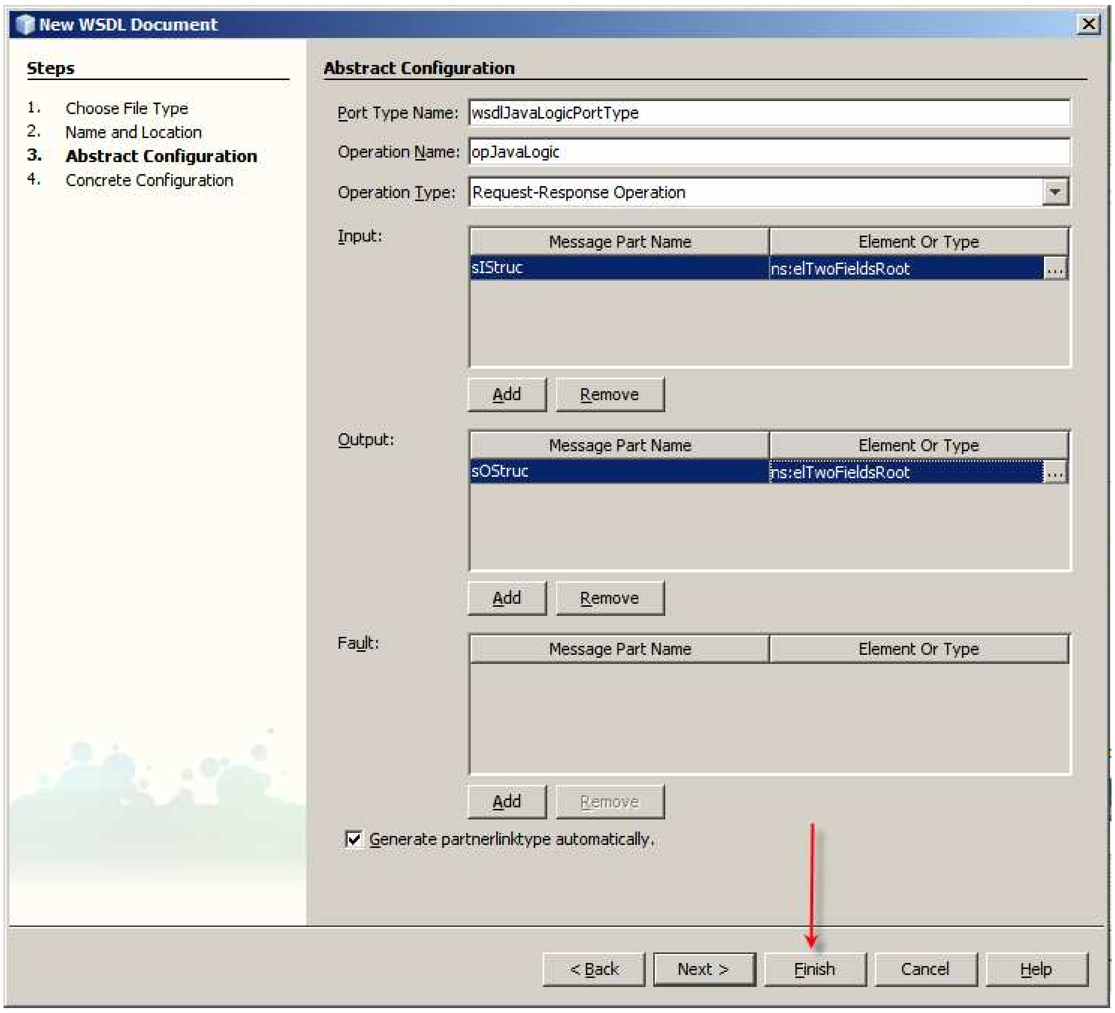
Operation Type: Request-Response Operation

Input:

Message Part Name	Element Or Type
sIStruc	ns:elTwoFieldsRoot

Figure 4-18 Input message now has the correct type

This completes configuration of the Abstract part of the WSDL. We are not interested in the Concrete part so we will complete the Wizard by clicking the “Finish” button, Figure 4-19. Clicking the “Next>” button would have lead us to the part of the wizard that aids in specification of the concrete part of the WSDL – we will not do that here.


New WSDL Document

Steps

1. Choose File Type
2. Name and Location
3. **Abstract Configuration**
4. Concrete Configuration

Abstract Configuration

Port Type Name: wsdlJavaLogicPortType

Operation Name: opJavaLogic

Operation Type: Request-Response Operation

Input:

Message Part Name	Element Or Type
sIStruc	ns:elTwoFieldsRoot

Output:

Message Part Name	Element Or Type
sOStruc	ns:elTwoFieldsRoot

Fault:

Message Part Name	Element Or Type

[✓] Generate partnerlinktype automatically.

Figure 4-19 Completing the Abstract part of the wizard

The process we have just gone through produced the interface specification to which our Java logic service will conform.

Let's now create the service implementation.

Right-click on the name of the EJB Module, emFile2FileJavaEE, then choose News -> Web Service form WSDL … as shown in Figure 4-20.

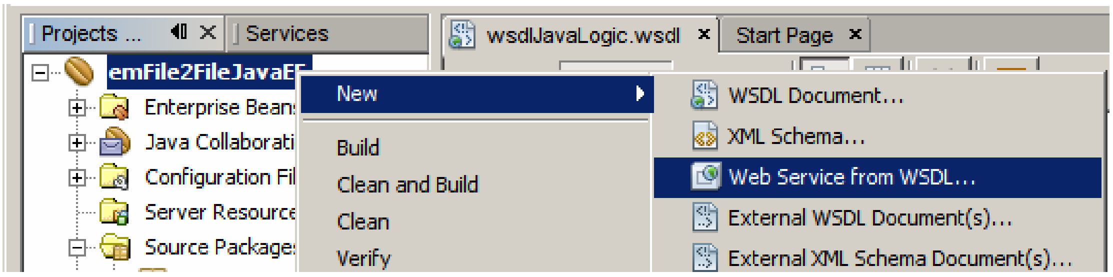
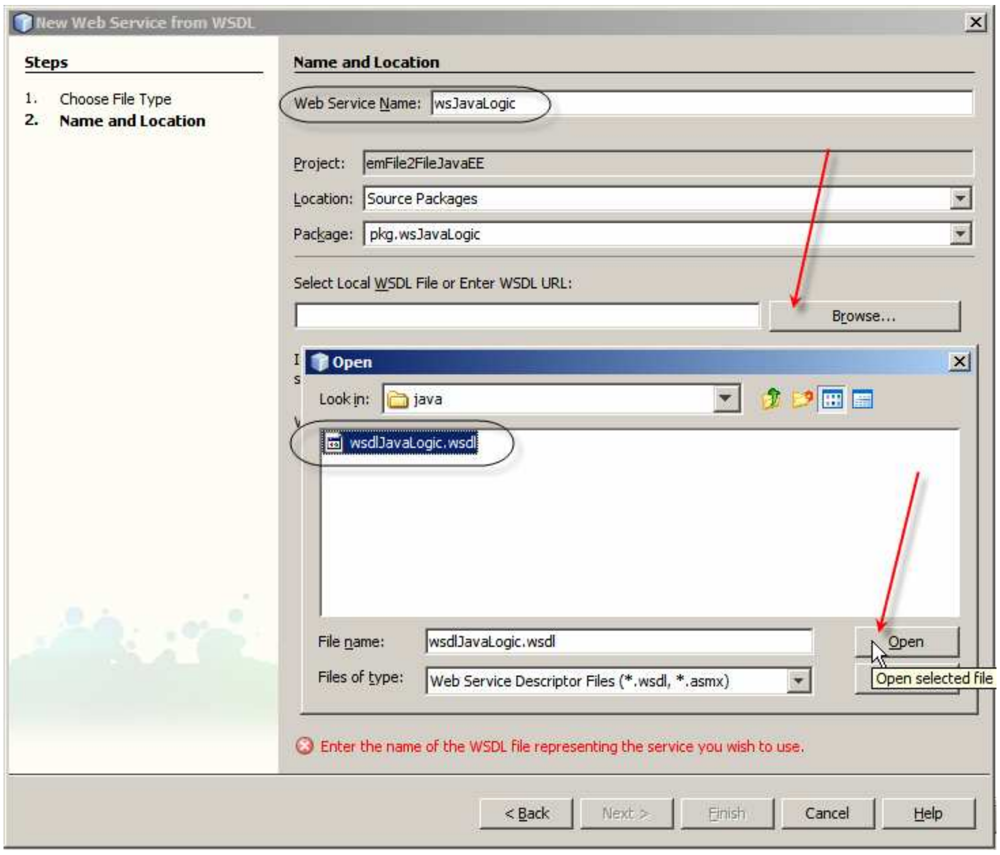


Figure 4-20 Choosing to create a Web Service from WSDL

Let's name the service wsJavaLogic, browse to the WSDL we just created and choose it, click Open to select the WSDL then click Finish to complete the wizard and have the skeleton Java class created, as shown in Figures 4-21, 4-22 and 4-23.


New Web Service from WSDL

Steps
1. Choose File Type
2. Name and Location

Name and Location

Web Service Name: wsJavaLogic

Project: emFile2FileJavaEE

Location: Source Packages

Package: pkg.wsJavaLogic

Select Local WSDL File or Enter WSDL URL:

Open

Look in: java

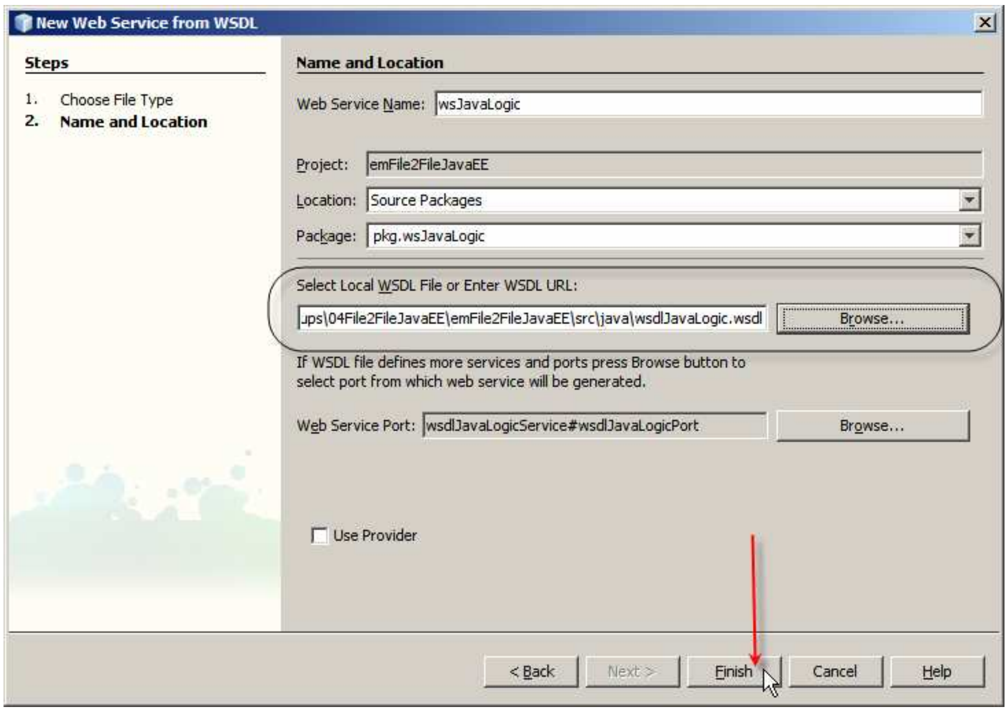
wsdlJavaLogic.wsdl

File name: wsdlJavaLogic.wsdl

Files of type: Web Service Descriptor Files (*.wsdl, *.asmx)

Enter the name of the WSDL file representing the service you wish to use.

Figure 4-21 Naming the service and choosing the WSDL


New Web Service from WSDL

Steps
1. Choose File Type
2. Name and Location

Name and Location

Web Service Name: wsJavaLogic

Project: emFile2FileJavaEE

Location: Source Packages

Package: pkg.wsJavaLogic

Select Local WSDL File or Enter WSDL URL:

.ps\04File2FileJavaEE\emFile2FileJavaEE\src\java\wsdlJavaLogic.wsdl

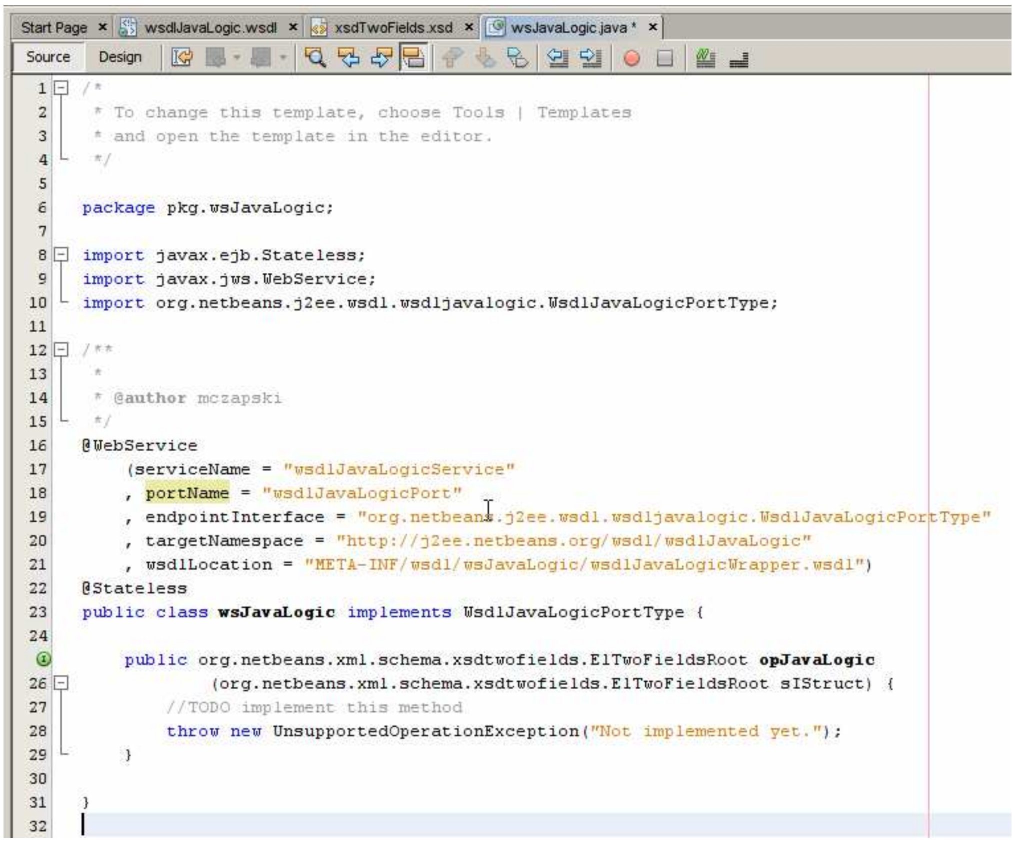
If WSDL file defines more services and ports press Browse button to select port from which web service will be generated.

Web Service Port: wsdlJavaLogicService#wsdlJavaLogicPort

[ ] Use Provider

Figure 4-22 Completing the Web Service form WSDL wizard

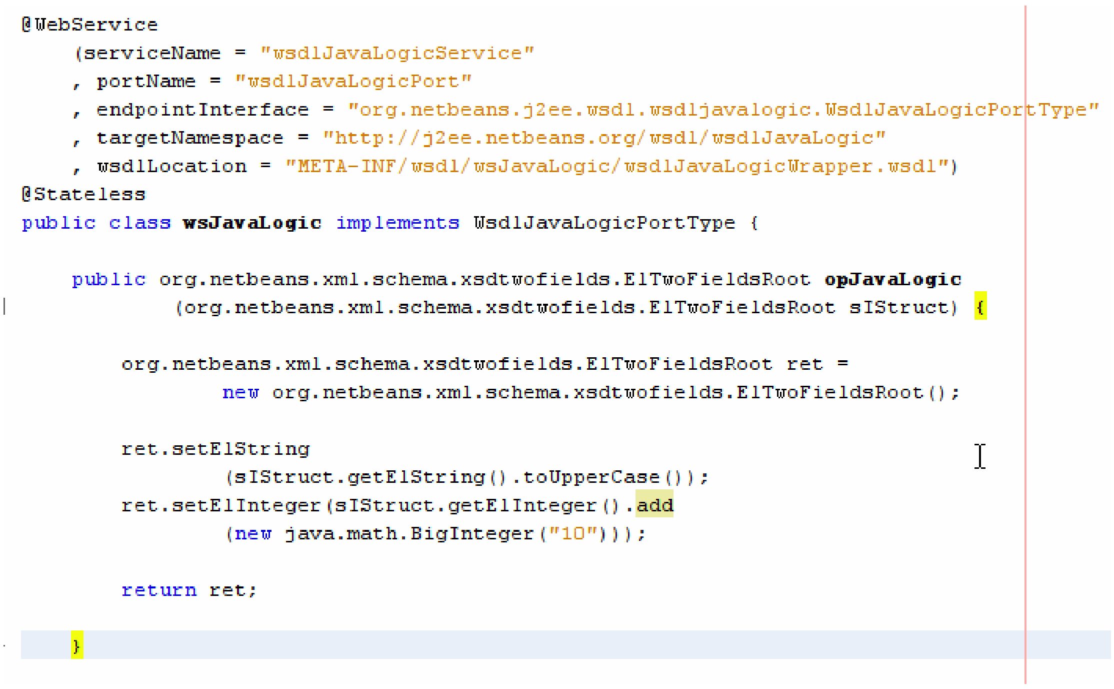
In Source mode, the skeleton Java class, with source reformatted for readability, will look similar to that shown in Figure 4-23.



```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */

package pkg.wsJavaLogic;

import javax.ejb.Stateless;
import javax.jws.WebService;
import org.netbeans.j2ee.wsdl.wsdljavalogic.WsdlJavaLogicPortType;

/**
 *
 * @author mczapski
 */
@WebService
    (serviceName = "wsdlJavaLogicService"
    , portName = "wsdlJavaLogicPort"
    , endpointInterface = "org.netbeans.j2ee.wsdl.wsdljavalogic.WsdlJavaLogicPortType"
    , targetNamespace = "http://j2ee.netbeans.org/wsdl/wsdlJavaLogic"
    , wsdlLocation = "META-INF/wsdl/wsJavaLogic/wsdlJavaLogicWrapper.wsdl")
@Stateless
public class wsJavaLogic implements WsdlJavaLogicPortType {

    public org.netbeans.xml.schema.xsdtwofields.ElTwoFieldsRoot opJavaLogic
            (org.netbeans.xml.schema.xsdtwofields.ElTwoFieldsRoot sIStruct) {
        //TODO implement this method
        throw new UnsupportedOperationException("Not implemented yet.");
    }

}
```

Figure 4-23 Java source of the new logic service

Let’s add some trivial logic like convert the content of the string element elString, to upper case and add 10 to the value of the integer element elInteger . The logic is shown in Figure 4-24.



```
@WebService
    (serviceName = "wsdlJavaLogicService"
    , portName = "wsdlJavaLogicPort"
    , endpointInterface = "org.netbeans.j2ee.wsdl.wsdljavalogic.WsdlJavaLogicPortType"
    , targetNamespace = "http://j2ee.netbeans.org/wsdl/wsdlJavaLogic"
    , wsdlLocation = "META-INF/wsdl/wsJavaLogic/wsdlJavaLogicWrapper.wsdl")
@Stateless
public class wsJavaLogic implements WsdlJavaLogicPortType {

    public org.netbeans.xml.schema.xsdtwofields.ElTwoFieldsRoot opJavaLogic
            (org.netbeans.xml.schema.xsdtwofields.ElTwoFieldsRoot sIStruct) {

        org.netbeans.xml.schema.xsdtwofields.ElTwoFieldsRoot ret =
                new org.netbeans.xml.schema.xsdtwofields.ElTwoFieldsRoot();

        ret.setElString
                (sIStruct.getElString().toUpperCase());
        ret.setElInteger(sIStruct.getElInteger().add
                (new java.math.BigInteger("10")));

        return ret;

    }
```

Figure 4-24 Trivial processing logic

One can easily imagine both much more complex message structures, possibly different for the input message and for the output message, ands much more complex logic that operates on these structures. The method is the same.

Let’s build the EJB Module, emFile2FileJavaEE, by right-clicking on the module name and choosing Build, as shown in Figure 4-25.

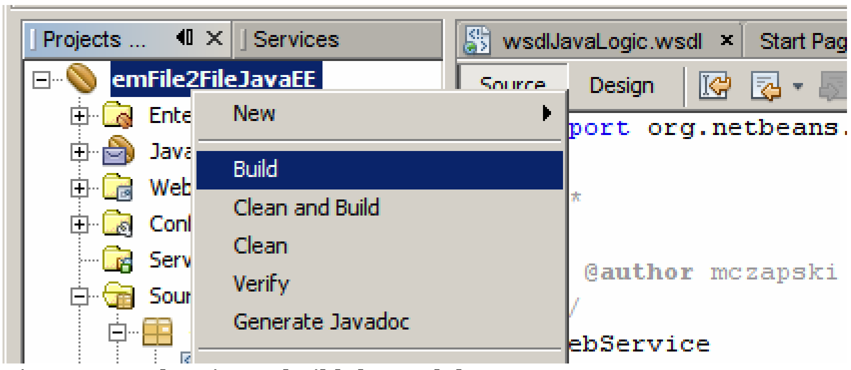


Figure 4-25 Choosing to build the module

The Java logic module is now ready to use in a Composite Application, which will be developed next.

5 File In to File Out with Java Logic

The module developed in Section 4 is ready to be used in a Composite Application. Let’s create a simple composite application wherein a File BC reads the content of a file, which has to be an XML message conforming to the XML Schema also developed in Section 4. The EJB Module is a request/reply service so the File BC will have to be configured to both receive the content of a file to be used as input and to write the service response to a file, using a single File BC.

Let’s create a new Composite Application project, caFiel2FileJavaEE. Use CAPS -> ESB -> Composite Application project type. This has been done in all previous Notes in this series so the specific steps will not be repeated.

Let’s now drag the EJB Module, emFile2FileJavaEE, onto the CASA Editor canvas, to the Service Assembly’s JBI Modules ‘swim line’. The result should appear as shown in Figure 5-1.

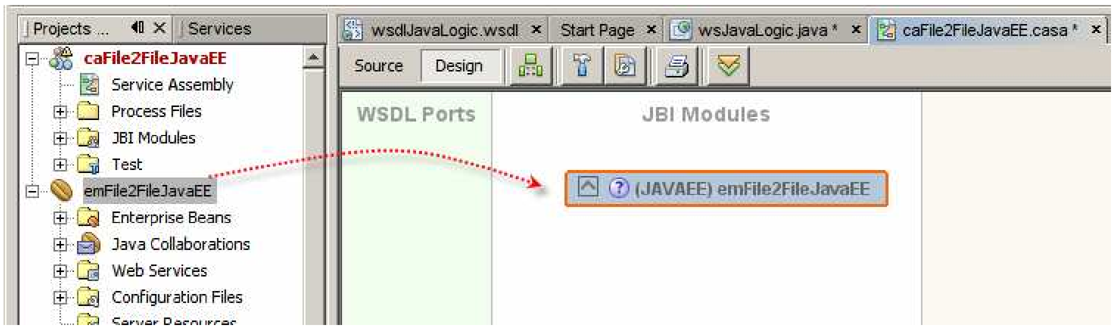


Figure 5-1 Adding EJB Module to the JBI Service Assembly

Let’s now drag the File BC form the palette to the WSDL Ports swim line, as shown in Figure 5-2.

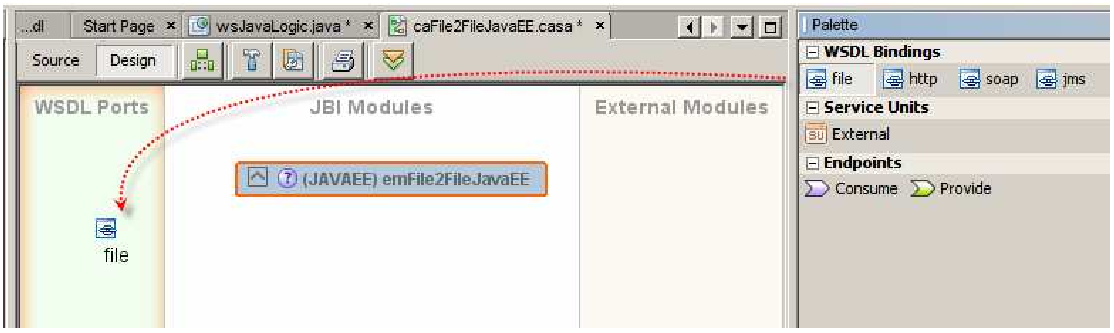


Figure 5-2 Adding File BC to the Service Assembly

Let's Build the Composite Application project then click and drag from the Consume connector of the File BC to the Provide connector of the EJB module as shown in Figure 5-3.

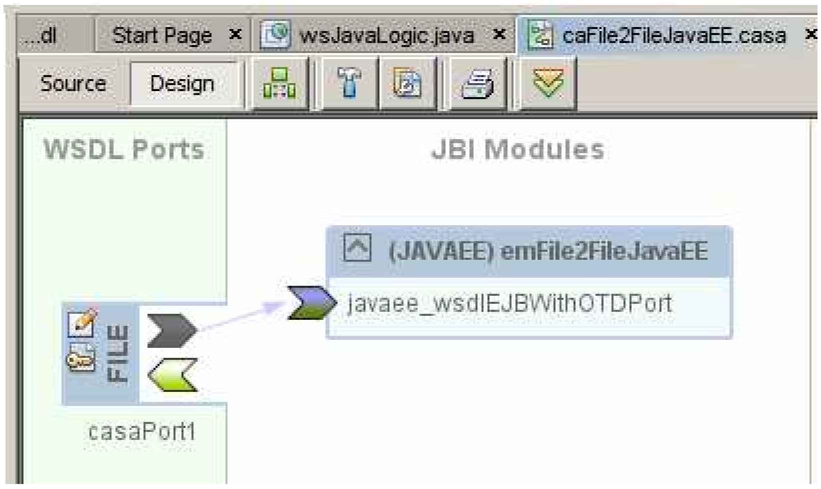


Figure 5-3 Connecting consumer to provider

Let's double-click on the caFile2FileJavavEE WSDL, created when we added the File BC to the Service Assembly, shown in Figure 5-4, to open it for configuration.

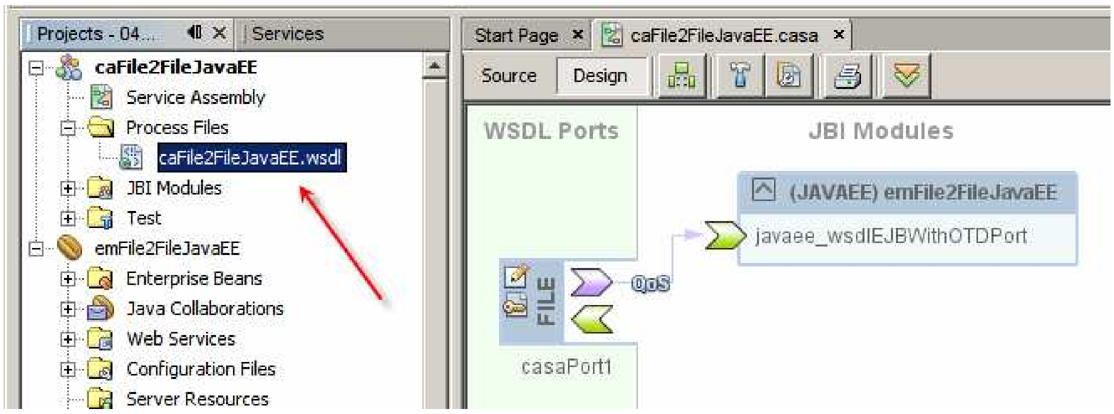
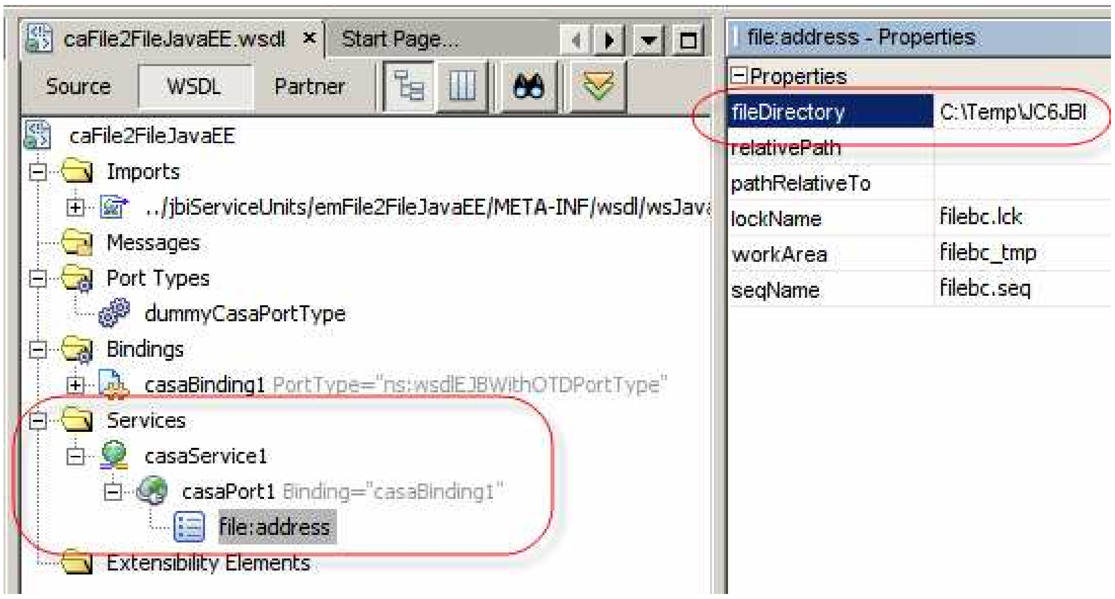


Figure 5-4 File BC WSDL in the Composite Application project

As on previous occasions, let's select the Services -> casaService1 -> casaPort1 -> file:address node of the WSDL and configure the fileDirectory property to point to the file system directory where the input file will be found and where the output file will be written. Figure 5-5 illustrates this.


caFile2FileJavaEE.wsdl | Start Page...
Source | WSDL | Partner

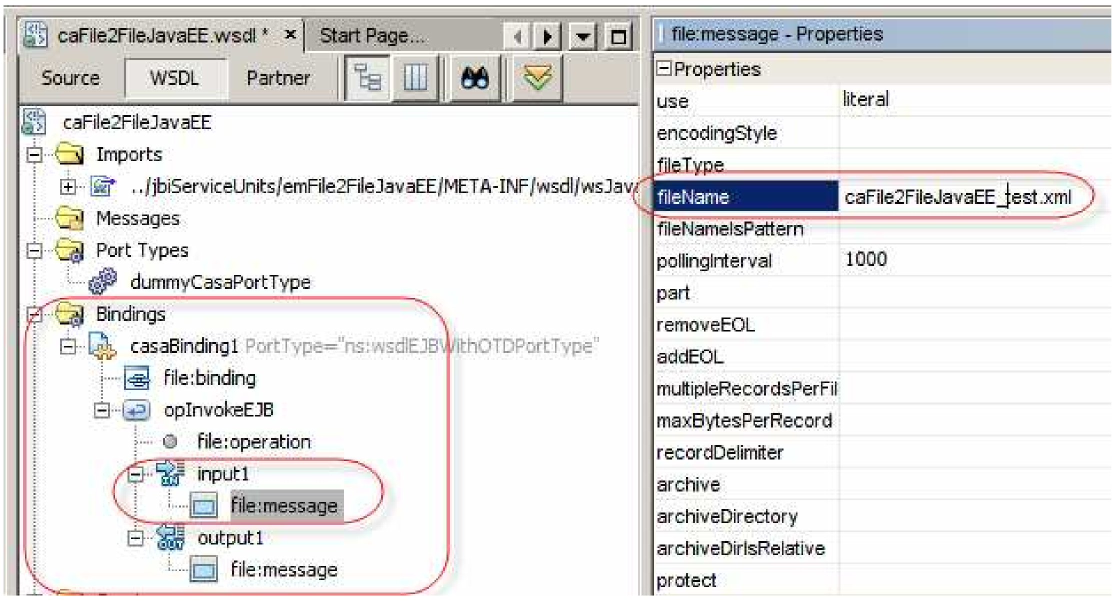
```
caFile2FileJavaEE
  Imports
    ../jbiServiceUnits/emFile2FileJavaEE/META-INF/wsdl/wsJav
  Messages
  Port Types
    dummyCasaPortType
  Bindings
    casaBinding1 PortType="ns:wsdlEJBWithOTDPortType"
  Services
    casaService1
      casaPort1 Binding="casaBinding1"
        file:address
  Extensibility Elements
```

file:address - Properties

Properties	
fileDirectory	C:\Temp\JC6JBI
relativePath	
pathRelativeTo	
lockName	filebc.lck
workArea	filebc_tmp
seqName	filebc.seq

Figure 5-5 Configuring file directory

The input file will be different from the output file even though we are using the same File BC. Let’s configure the name of the input file. Let’s prefix the default name, test.xml, with the name of the composite application to obtain a unique name. Figure 5-6 illustrates this.


caFile2FileJavaEE.wsdl * | Start Page...
Source | WSDL | Partner

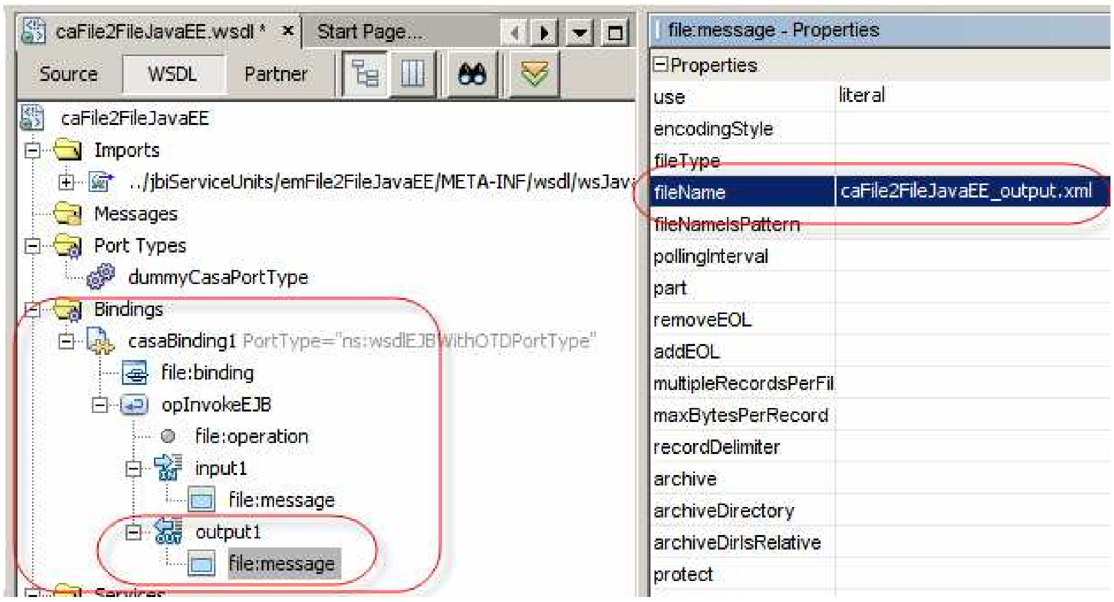
```
caFile2FileJavaEE
  Imports
    ../jbiServiceUnits/emFile2FileJavaEE/META-INF/wsdl/wsJav
  Messages
  Port Types
    dummyCasaPortType
  Bindings
    casaBinding1 PortType="ns:wsdlEJBWithOTDPortType"
      file:binding
      opInvokeEJB
        file:operation
        input1
          file:message
        output1
          file:message
```

file:message - Properties

Properties	
use	literal
encodingStyle	
fileType	
fileName	caFile2FileJavaEE_test.xml
fileNameIsPattern	
pollingInterval	1000
part	
removeEOL	
addEOL	
multipleRecordsPerFil	
maxBytesPerRecord	
recordDelimiter	
archive	
archiveDirectory	
archiveDirIsRelative	
protect	

Figure 5-6 Naming the input file

Let’s do the same to the output file, see Figure 5-7.



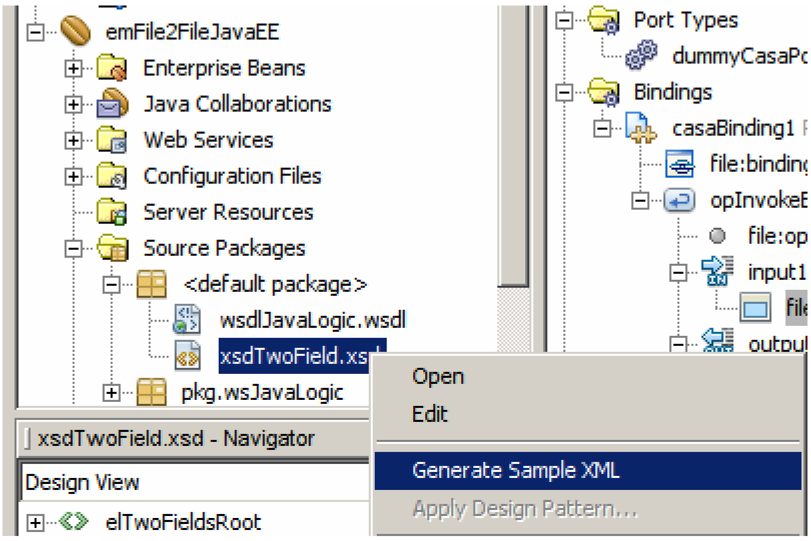
```
caFile2FileJavaEE
  Imports
    ../jbiServiceUnits/emFile2FileJavaEE/META-INF/wsdl/wsJav
  Messages
  Port Types
    dummyCasaPortType
  Bindings
    casaBinding1 PortType="ns:wsdlEJBWithOTDPortType"
      file:binding
      opInvokeEJB
        file:operation
        input1
          file:message
        output1
          file:message
```

Properties	
use	literal
encodingStyle	
fileType	
fileName	caFile2FileJavaEE_output.xml
fileNamesPattern	
pollingInterval	
part	
removeEOL	
addEOL	
multipleRecordsPerFil	
maxBytesPerRecord	
recordDelimiter	
archive	
archiveDirectory	
archiveDirIsRelative	
protect	

Figure 5-7 Naming output file

The composite application is now complete. Let's build and deploy it. Since this was done in every Note so far the steps will not be repeated.

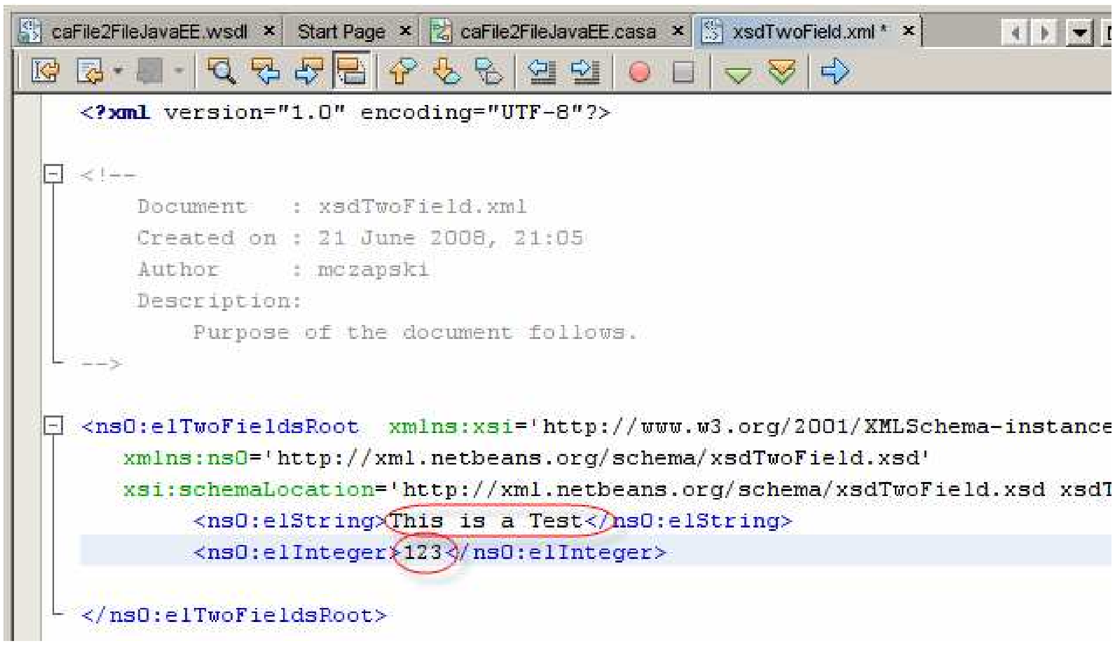
The input file, caFile2FileJavaEE_test.xml, must exist and must contain valid XML Instance document conforming to the XML Schema, xsdTwoFields, developed in Section 4. Let's create an instance document using NetBeans facilities. Locate the xsdTwoFields.xsd in project folder hierarchy emFile2FileJavaEE/Source Packages/<default package>, right-click its name and choose Generate Sample XML. See Figure 5-8.



```
emFile2FileJavaEE
  Enterprise Beans
  Java Collaborations
  Web Services
  Configuration Files
  Server Resources
  Source Packages
    <default package>
      wsdlJavaLogic.wsdl
      xsdTwoField.xsd
    pkg.wsJavaLogic

elTwoFieldsRoot

Port Types
  dummyCasaPc
Bindings
  casaBinding1
    file:binding
    opInvokeE
      file:op
      input1
        File
      outpul
```

Figure 5-8 Choosing to generate a XML Instance Document

Accept defaults in the dialog box that follows and modify the resulting XML Instance document to read as shown in Figure 5-9.



```
<?xml version="1.0" encoding="UTF-8"?>

<!--
    Document   : xsdTwoField.xml
    Created on : 21 June 2008, 21:05
    Author     : mczapski
    Description:
        Purpose of the document follows.
-->

<ns0:elTwoFieldsRoot  xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance
   xmlns:ns0='http://xml.netbeans.org/schema/xsdTwoField.xsd'
   xsi:schemaLocation='http://xml.netbeans.org/schema/xsdTwoField.xsd xsdT
        <ns0:elString>This is a Test</ns0:elString>
        <ns0:elInteger>123</ns0:elInteger>

</ns0:elTwoFieldsRoot>
```

Figure 5-9 XML Instance document with test data

This document is created in the file system directory associated with the project group. We need it in the directory nominated as the input file directory in the File BC configuration, in this case C:\Temp\JC6JBI, and we need it to be named as specified in the file:message fielName property on the input side, in this case caFile2FileJavaEE_test.xml. Let’s copy the file from where it was created to the appropriate directory and rename it. Once the file is renamed it will be picked up within a second and processed. The result will be written to the file named caFile2FileJavaEE_output.xml in the same directory as the input file.

Listing 5-1 shows the output of the run with the sample input shown in Figure 5-9.

```
<elTwoFieldsRoot
xmlns="http://xml.netbeans.org/schema/xsdTwoFieldsx">
<elString>THIS IS A TEST</elString>
<elInteger>133</elInteger>
</elTwoFieldsRoot>
```
Listing 5-1 Output of execution with sample input from Figure 5-9

The string element value was converted to upper case and the integer element value is 10 more then the input was.

6 Summary

This document walked the reader, step-by-step, through the process of creating and exercising a Java CAPS 6/JBI (or OpenESB) basic File to File integration solution that used Java to implement transformation logic. The solution used only JBI and JEE components. It operated on a file containing a single XML record. It demonstrated how arbitrary Java logic can be created and incorporated into JBI Composite Applications.