Java CAPS 6 Update 1
Exposing MTOM-capable Java CAPS Classic Web Service

Michael.Czapski@sun.com

February 2009

Contents

1. Introduction

If we overlook the fact that using web services to transfer large payloads is a very stupid idea, we will be faced with the need to implement the optimisation mechanisms to make transfer of large payloads using web services a little less inefficient from the stand point of the size of the over-the-wire data to be transferred.

The standardised, supported mechanism for this is the Message Transmission Optimisation Method (MTOM), http://en.wikipedia.org/wiki/MTOM. Java CAPS Repository-based Web Services don't offer a convenient mechanism to provide MTOM support.

This note walks through the implementation of a Java CAPS Repository-based, eInsight-based web service and the implementation of the EJB-based Web Service Wrapper for this service, which provides support for MTOM. The Note discusses how to exercise the services using the NetBeans web services testing facilities and how to observe on-the-wire message exchanges.

Create a Project Group

I find the NetBeans flat project structure annoying. The Enterprise Designer hierarchical project structure, ported to Java CAPS 6 Classic, was a much more reasonable way of organizing projects and project artefacts. The Project group feature of NetBeans is a poor substitute for that. Be it how it may, I have gotten into a habit of creating project groups so I can collect related projects and open/close related projects in a hit.

So, let’s create a new project group, provide a folder for the projects that belong to the group and crenate new projects for this Note in that group.

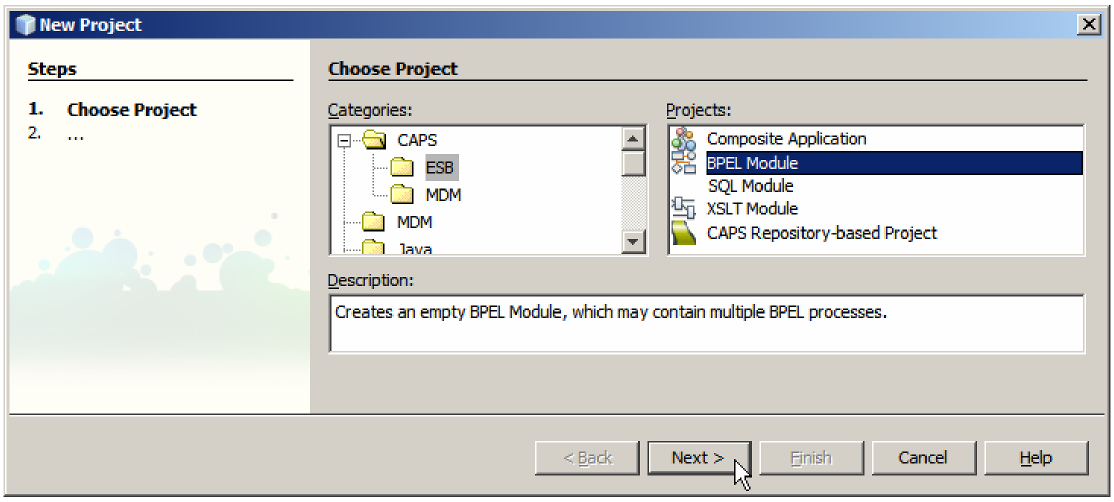
In Java CAPS 6, once you create a project group, the CAPS Components Library project will “disappear”. To “get it back” you can re-connect the repository or hit the “Refresh All” button on the tool bar.

2. WSDL Notes

Java CAPS 6 is fussy about the kind of WSDL it will accept for different kinds of projects. In this Note we will be dealing with EJB-based Web Services and Repository-based Web Services. Empirical experience tells me that the only kind of WSDL that is acceptable to both is a) document/literal WSDL and b) a WSDL with an in-line XML Schema. Another words, don't use an external XML Schema included in the WSDL and expect it to work. It will work in same projects but not in others. In particular, if you create a Classic Web Service, which you wish to invoke form an EJB via a Web Service Reference, the tooling will be unable to find the referenced XML Schema and the result will be a fiasco.

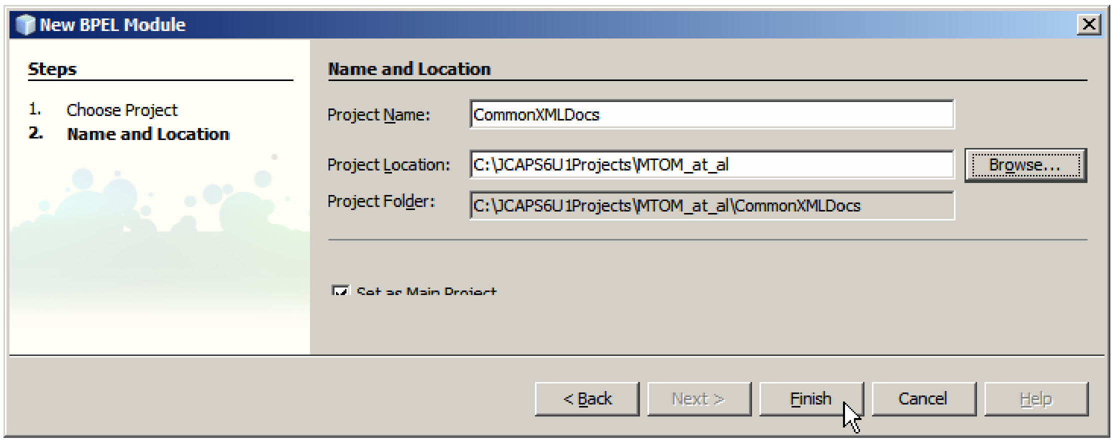
One way I adopted recently goes like this:

1. Create a BPEL 2.0 Module for common WSDLs and XSDs
2. Create an XMLS Schema document using the NetBeans "New->XML Schema" method, with eventual Request and Reply in the XSD as separate Elements.
3. Create a "New->WSDL Document", specifying the appropriate XML Schema elements for Request and Reply messages, and Faults if you have any.
4. Copy the new WSDL document and replace the content of the <types></types> with the entire content of the XML Schema document.



Creates an empty BPEL Module, which may contain multiple BPEL processes.

Figure 2-1 Create a new BPEL Modules



Project Name: CommonXMLDocs
Project Location: C:\JCAPS6U1Projects\MTOM_at_al
Project Folder: C:\JCAPS6U1Projects\MTOM_at_al\CommonXMLDocs

Figure 2-2 Name the module to indicate what objects it contains

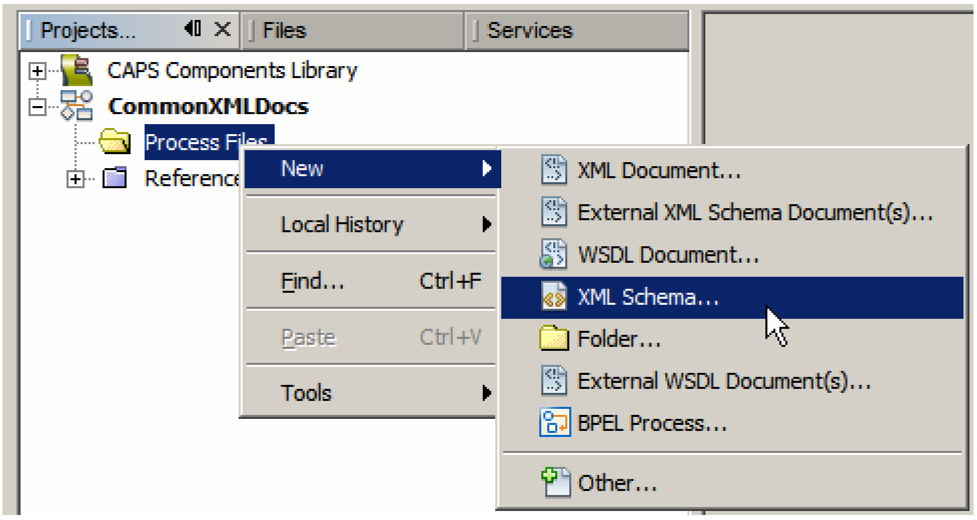
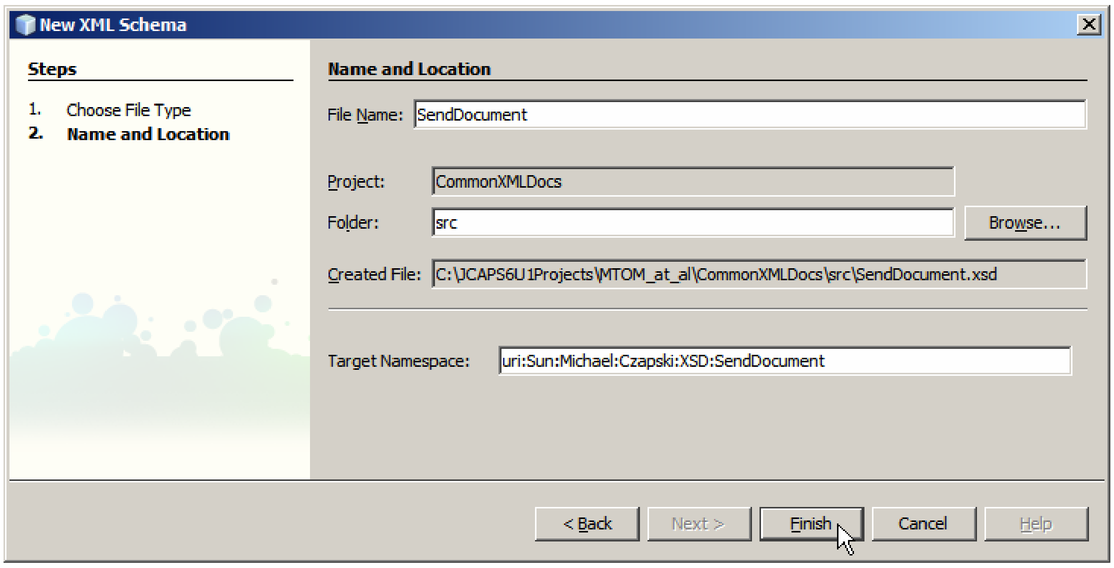


Figure 2-3 Create a New->XML Schema


New XML Schema

Steps

1. Choose File Type
2. **Name and Location**

Name and Location

File Name: SendDocument

Project: CommonXMLDocs

Folder: src

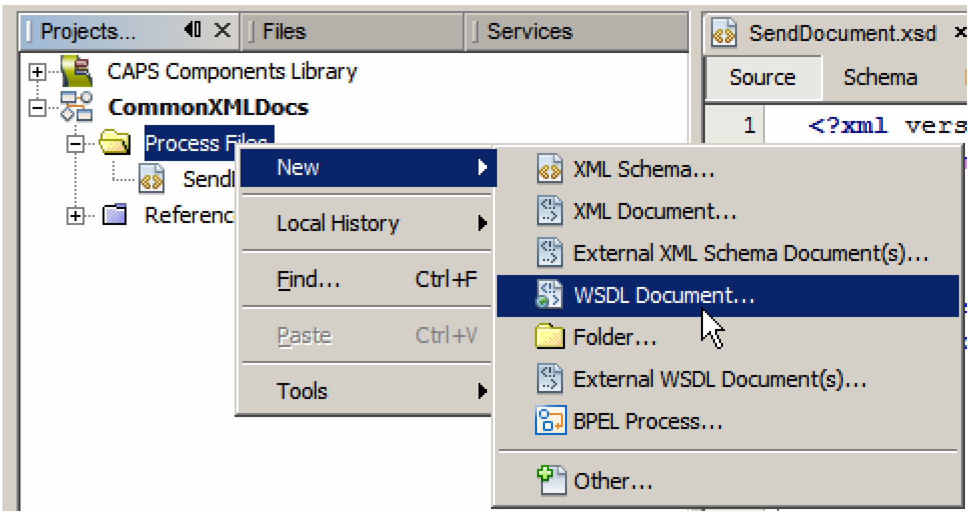
Created File: C:\JCAPS6U1Projects\MTOM_at_al\CommonXMLDocs\src\SendDocument.xsd

Target Namespace: uri:Sun:Michael:Czapski:XSD:SendDocument

Figure 2-4 Name the document and change target namespace value

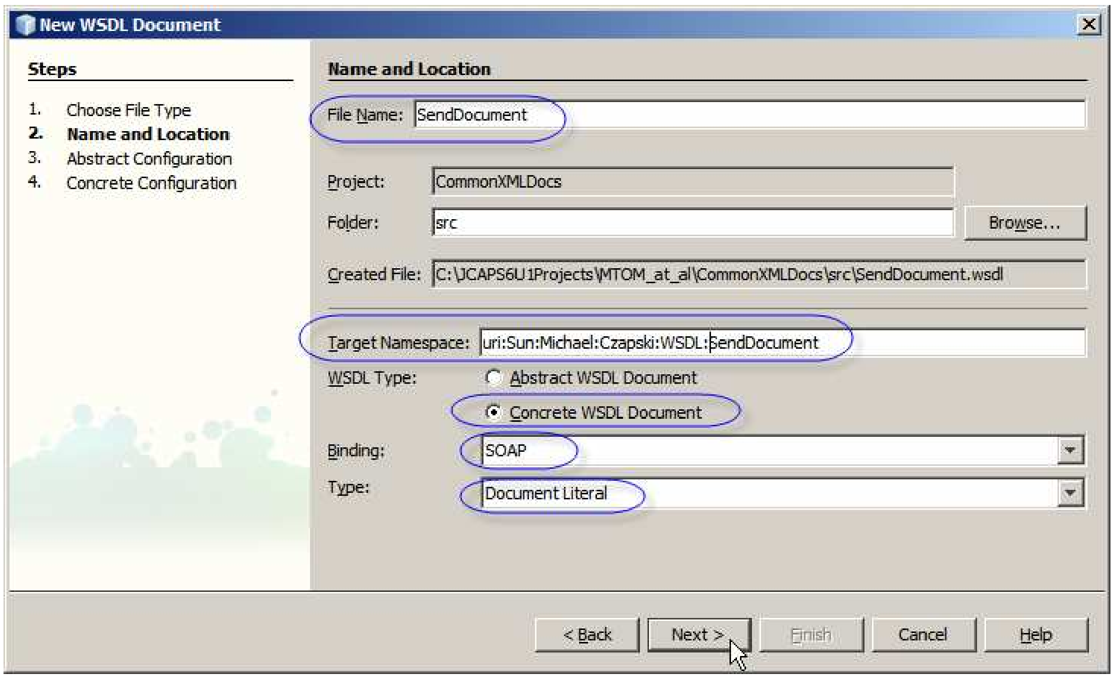
Add the elements to the XSD as required. Listing 3-1 provides the XML Schema document we will use in this Note.

Now create a New->WSDL document.



```
1    <?xml vers
```

Figure 2-5 Create a New->WSDL Document



File Name: SendDocument

Project: CommonXMLDocs

Folder: src

Created File: C:\JCAPS6U1Projects\MTOM_at_al\CommonXMLDocs\src\SendDocument.wsdl

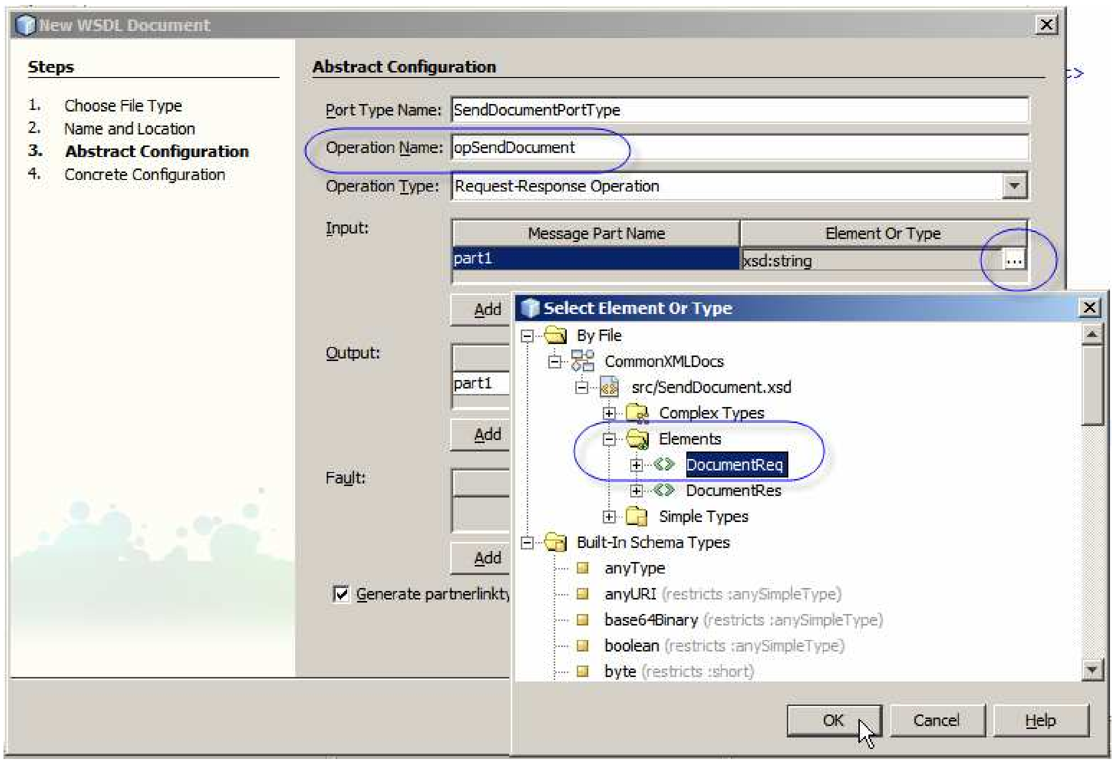
Target Namespace: uri:Sun:Michael:Czapski:WSDL:SendDocument

WSDL Type: Abstract WSDL Document [✓ Concrete WSDL Document]

Binding: SOAP

Type: Document Literal

Figure 2-6 Name the WSDL, specify target namespace and Binding information


New WSDL Document

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

Abstract Configuration

Port Type Name: SendDocumentPortType
Operation Name: opSendDocument
Operation Type: Request-Response Operation
Input:

Message Part Name	Element Or Type
part1	xsd:string

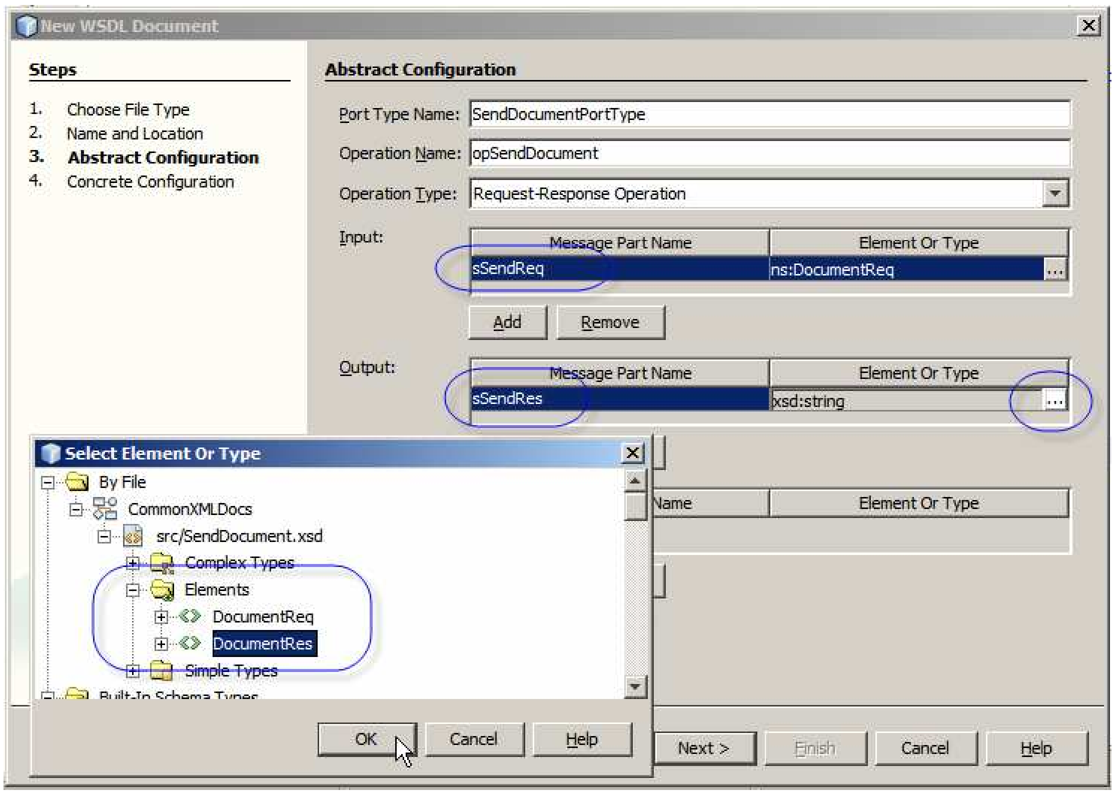
Output: part1

Fault:

☑ Generate partnerlinkty

Select Element Or Type
- By File
  - CommonXMLDocs
    - src/SendDocument.xsd
      - Complex Types
      - Elements
        - DocumentReq
        - DocumentRes
      - Simple Types
- Built-In Schema Types
  - anyType
  - anyURI (restricts :anySimpleType)
  - base64Binary (restricts :anySimpleType)
  - boolean (restricts :anySimpleType)
  - byte (restricts :short)

Figure 2-7 Name the Operation, choose Request (Input) type and name it


New WSDL Document

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

Abstract Configuration

Port Type Name: SendDocumentPortType
Operation Name: opSendDocument
Operation Type: Request-Response Operation
Input:

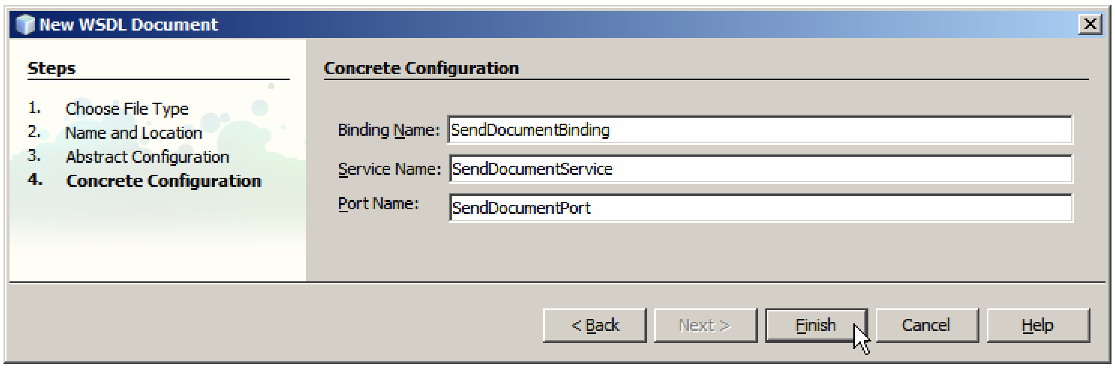
Message Part Name	Element Or Type
sSendReq	ns:DocumentReq

Output:

Message Part Name	Element Or Type
sSendRes	xsd:string

Select Element Or Type
- By File
  - CommonXMLDocs
    - src/SendDocument.xsd
      - Complex Types
      - Elements
        - DocumentReq
        - DocumentRes
      - Simple Types
- Built-In Schema Types

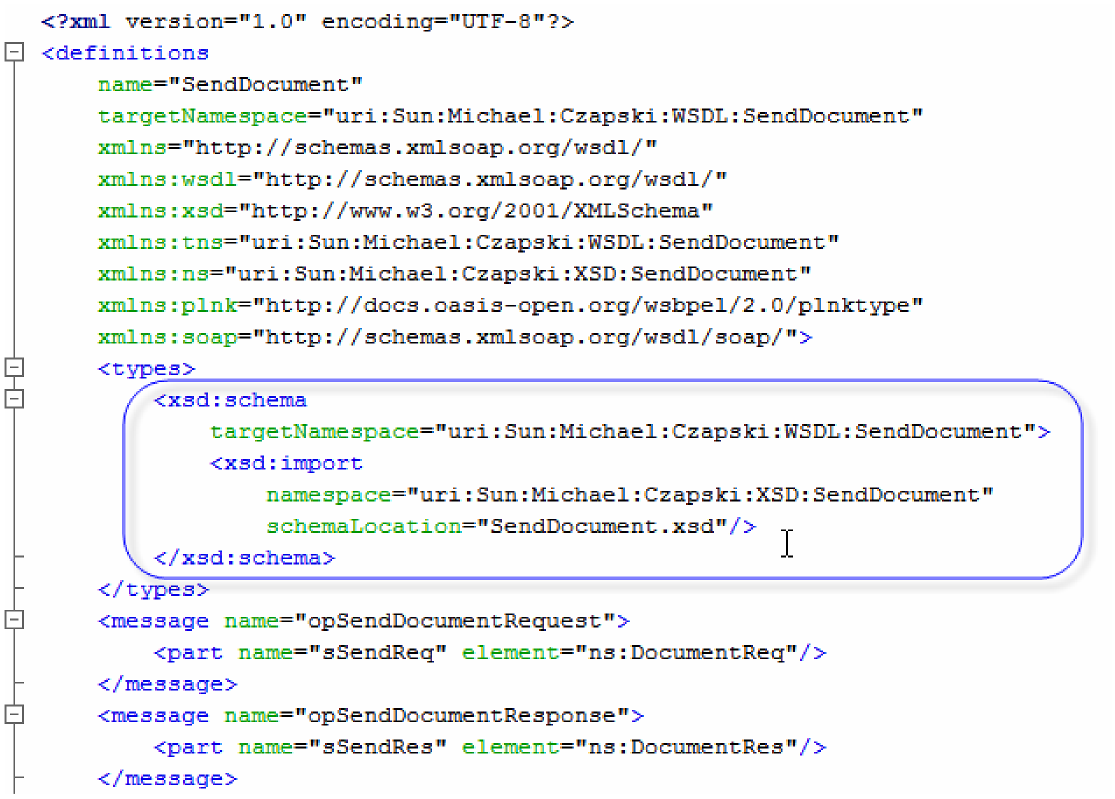
Figure 2-8 Name Input and Output (Response) and choose type



Binding Name: SendDocumentBinding
Service Name: SendDocumentService
Port Name: SendDocumentPort

Figure 2-9 Accept default to finish

Slightly re-formatted for readability, in Source mode, the “interesting” WSDL fragment is shown in Figure 3-10.



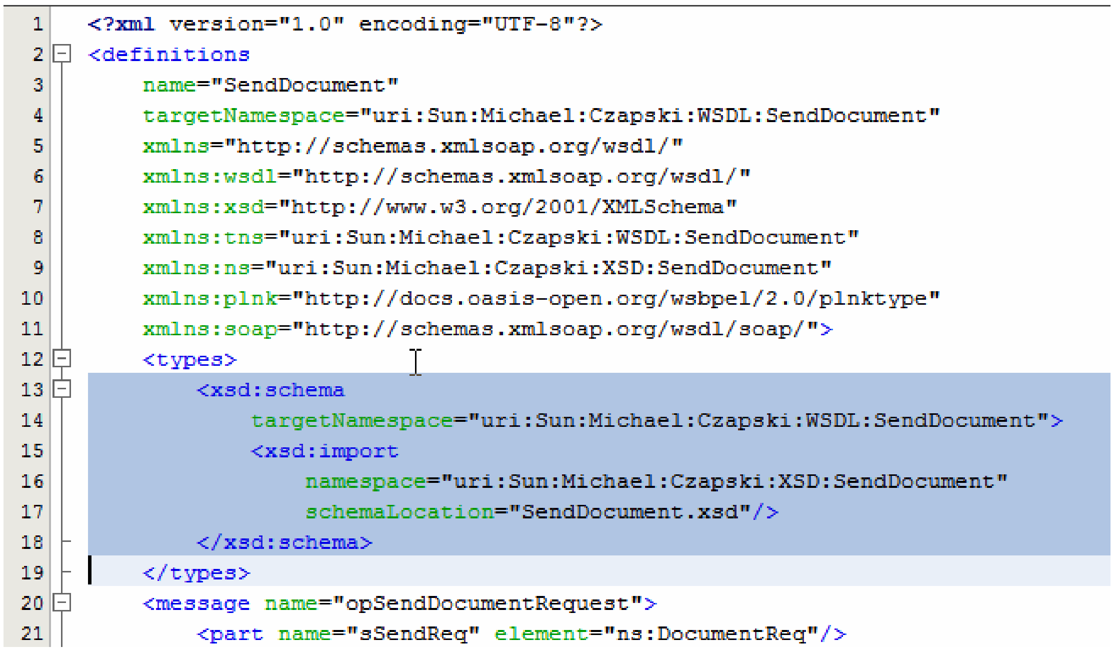
```
<?xml version="1.0" encoding="UTF-8"?>
<definitions
    name="SendDocument"
    targetNamespace="uri:Sun:Michael:Czapski:WSDL:SendDocument"
    xmlns="http://schemas.xmlsoap.org/wsdl/"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:tns="uri:Sun:Michael:Czapski:WSDL:SendDocument"
    xmlns:ns="uri:Sun:Michael:Czapski:XSD:SendDocument"
    xmlns:plnk="http://docs.oasis-open.org/wsbpel/2.0/plnktype"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/">
    <types>
        <xsd:schema
            targetNamespace="uri:Sun:Michael:Czapski:WSDL:SendDocument">
            <xsd:import
                namespace="uri:Sun:Michael:Czapski:XSD:SendDocument"
                schemaLocation="SendDocument.xsd"/>
        </xsd:schema>
    </types>
    <message name="opSendDocumentRequest">
        <part name="sSendReq" element="ns:DocumentReq"/>
    </message>
    <message name="opSendDocumentResponse">
        <part name="sSendRes" element="ns:DocumentRes"/>
    </message>
```

Figure 2-10 WSDL fragment

Note the content of the <types></types> tag, with XML Schema import.

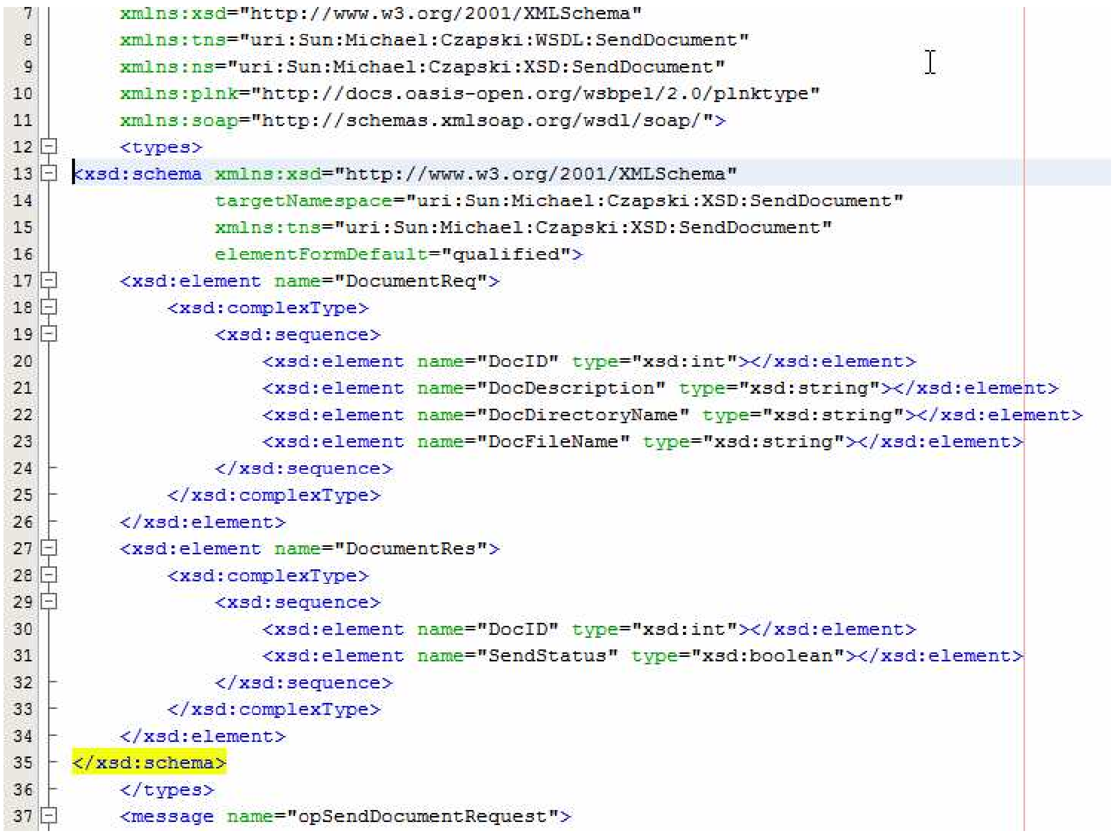
Copy the entire WSDL and paste as a new WSDL wit a name perhaps suffixed with some constant that tells you that this WSDL has the XSD embedded rather then imported.

Now open the XML Schema, copy its content and replace the content of the <types></types> tag with it.



```
<?xml version="1.0" encoding="UTF-8"?>
<definitions
    name="SendDocument"
    targetNamespace="uri:Sun:Michael:Czapski:WSDL:SendDocument"
    xmlns="http://schemas.xmlsoap.org/wsdl/"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:tns="uri:Sun:Michael:Czapski:WSDL:SendDocument"
    xmlns:ns="uri:Sun:Michael:Czapski:XSD:SendDocument"
    xmlns:plnk="http://docs.oasis-open.org/wsbpel/2.0/plnktype"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/">
    <types>
        <xsd:schema
            targetNamespace="uri:Sun:Michael:Czapski:WSDL:SendDocument">
            <xsd:import
                namespace="uri:Sun:Michael:Czapski:XSD:SendDocument"
                schemaLocation="SendDocument.xsd"/>
        </xsd:schema>
    </types>
    <message name="opSendDocumentRequest">
        <part name="sSendReq" element="ns:DocumentReq"/>
```

Figure 2-11 Select the content of the <types></types> tag



```
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:tns="uri:Sun:Michael:Czapski:WSDL:SendDocument"
    xmlns:ns="uri:Sun:Michael:Czapski:XSD:SendDocument"
    xmlns:plnk="http://docs.oasis-open.org/wsbpel/2.0/plnktype"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/">
    <types>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            targetNamespace="uri:Sun:Michael:Czapski:XSD:SendDocument"
            xmlns:tns="uri:Sun:Michael:Czapski:XSD:SendDocument"
            elementFormDefault="qualified">
    <xsd:element name="DocumentReq">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="DocID" type="xsd:int"></xsd:element>
                <xsd:element name="DocDescription" type="xsd:string"></xsd:element>
                <xsd:element name="DocDirectoryName" type="xsd:string"></xsd:element>
                <xsd:element name="DocFileName" type="xsd:string"></xsd:element>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="DocumentRes">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="DocID" type="xsd:int"></xsd:element>
                <xsd:element name="SendStatus" type="xsd:boolean"></xsd:element>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
</xsd:schema>
    </types>
    <message name="opSendDocumentRequest">
```

Figure 2-12 Paste the XML Schema source in its place and Format document

This produces a WSDL that will work in all kinds of projects and an XML Schema that can be used where WSDL is not required, for example to create XSD-based OTDs.

The XML Schema we will use in this Note looks like that presented in Listing 3-1.

Listing 2-1 Send Document XML Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            targetNamespace="uri:Sun:Michael:Czapski:XSD:SendDocument"
            xmlns:tns="uri:Sun:Michael:Czapski:XSD:SendDocument"
            elementFormDefault="qualified">
    <xsd:element name="DocumentReq">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="DocID" type="xsd:int"></xsd:element>
                <xsd:element name="DocDescription" type="xsd:string"></xsd:element>
                <xsd:element name="DocDirectoryName" type="xsd:string"></xsd:element>
                <xsd:element name="DocFileName" type="xsd:string"></xsd:element>
                <xsd:element name="DocBody"
                             type="xsd:base64Binary" minOccurs="0"></xsd:element>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="DocumentRes">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="DocID" type="xsd:int"></xsd:element>
                <xsd:element name="SendStatus" type="xsd:boolean"></xsd:element>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
</xsd:schema>
```

The original WSDL is presented in Listing 3-2

Figure 2-2 Original WSDL

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions
    name="SendDocument"
    targetNamespace="uri:Sun:Michael:Czapski:WSDL:SendDocument"
    xmlns="http://schemas.xmlsoap.org/wsdl/"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:tns="uri:Sun:Michael:Czapski:WSDL:SendDocument"
    xmlns:ns="uri:Sun:Michael:Czapski:XSD:SendDocument"
    xmlns:plnk="http://docs.oasis-open.org/wsbpel/2.0/plnktype"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/">
    <types>
        <xsd:schema
            targetNamespace="uri:Sun:Michael:Czapski:WSDL:SendDocument">
            <xsd:import
                namespace="uri:Sun:Michael:Czapski:XSD:SendDocument"
                schemaLocation="SendDocument.xsd"/>
        </xsd:schema>
    </types>
    <message name="opSendDocumentRequest">
        <part name="sSendReq" element="ns:DocumentReq"/>
    </message>
    <message name="opSendDocumentResponse">
        <part name="sSendRes" element="ns:DocumentRes"/>
    </message>
    <portType name="SendDocumentPortType">
        <operation name="opSendDocument">
            <input name="input1" message="tns:opSendDocumentRequest"/>
            <output name="output1" message="tns:opSendDocumentResponse"/>
        </operation>
    </portType>
    <binding name="SendDocumentBinding" type="tns:SendDocumentPortType">
        <soap:binding
             style="document"
             transport="http://schemas.xmlsoap.org/soap/http"/>
        <operation name="opSendDocument">
            <soap:operation/>
            <input name="input1">
                <soap:body use="literal"/>
            </input>
            <output name="output1">
                <soap:body use="literal"/>
            </output>
        </operation>
    </binding>
    <service name="SendDocumentService">
        <port name="SendDocumentPort" binding="tns:SendDocumentBinding">
            <soap:address
location="http://localhost:${HttpDefaultPort}/SendDocumentService/SendDocumentPort"/>
        </port>
    </service>
    <plnk:partnerLinkType name="SendDocument">
        <!-- A partner link type is automatically generated when a new port type is
added. Partner link types are used by BPEL processes.
In a BPEL process, a partner link represents the interaction between the BPEL process
and a partner service. Each partner link is associated with a partner link type.
A partner link type characterizes the conversational relationship between two
services. The partner link type can have one or two roles.-->
        <plnk:role name="SendDocumentPortTypeRole"
portType="tns:SendDocumentPortType"/>
    </plnk:partnerLinkType>
</definitions>
```

The modified WSDL is presented in Listing 3-3

Figure 2-3 Modified WSDL

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions
    name="SendDocument"
    targetNamespace="uri:Sun:Michael:Czapski:WSDL:SendDocument"
    xmlns="http://schemas.xmlsoap.org/wsdl/"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:tns="uri:Sun:Michael:Czapski:WSDL:SendDocument"
    xmlns:ns="uri:Sun:Michael:Czapski:XSD:SendDocument"
    xmlns:plnk="http://docs.oasis-open.org/wsbpel/2.0/plnktype"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/">
    <types>
        <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            targetNamespace="uri:Sun:Michael:Czapski:XSD:SendDocument"
            xmlns:tns="uri:Sun:Michael:Czapski:XSD:SendDocument"
            elementFormDefault="qualified">
            <xsd:element name="DocumentReq">
                <xsd:complexType>
                    <xsd:sequence>
                        <xsd:element name="DocID" type="xsd:int">
                        </xsd:element>
                        <xsd:element name="DocDescription" type="xsd:string">
                        </xsd:element>
                        <xsd:element name="DocDirectoryName" type="xsd:string">
                        </xsd:element>
                        <xsd:element name="DocFileName" type="xsd:string">
                        </xsd:element>
                        <xsd:element name="DocBody"
                                     type="xsd:base64Binary" minOccurs="0">
                        </xsd:element>
                    </xsd:sequence>
                </xsd:complexType>
            </xsd:element>
            <xsd:element name="DocumentRes">
                <xsd:complexType>
                    <xsd:sequence>
                        <xsd:element name="DocID" type="xsd:int">
                        </xsd:element>
                        <xsd:element name="SendStatus" type="xsd:boolean">
                        </xsd:element>
                    </xsd:sequence>
                </xsd:complexType>
            </xsd:element>
        </xsd:schema>
    </types>
    <message name="opSendDocumentRequest">
        <part name="sSendReq" element="ns:DocumentReq"/>
    </message>
    <message name="opSendDocumentResponse">
        <part name="sSendRes" element="ns:DocumentRes"/>
    </message>
    <portType name="SendDocumentPortType">
```

```
        <operation name="opSendDocument">
            <input name="input1" message="tns:opSendDocumentRequest"/>
            <output name="output1" message="tns:opSendDocumentResponse"/>
        </operation>
    </portType>
    <binding name="SendDocumentBinding" type="tns:SendDocumentPortType">
        <soap:binding style="document" 
transport="http://schemas.xmlsoap.org/soap/http"/>
        <operation name="opSendDocument">
            <soap:operation/>
            <input name="input1">
                <soap:body use="literal"/>
            </input>
            <output name="output1">
                <soap:body use="literal"/>
            </output>
        </operation>
    </binding>
    <service name="SendDocumentService">
        <port name="SendDocumentPort" binding="tns:SendDocumentBinding">
            <soap:address 
location="http://localhost:${HttpDefaultPort}/SendDocumentService/SendDocumentPort"/>
        </port>
    </service>
    <plnk:partnerLinkType name="SendDocument">
        <!-- A partner link type is automatically generated when a new port type is 
added. Partner link types are used by BPEL processes. 
In a BPEL process, a partner link represents the interaction between the BPEL process 
and a partner service. Each partner link is associated with a partner link type.
A partner link type characterizes the conversational relationship between two 
services. The partner link type can have one or two roles.-->
        <plnk:role name="SendDocumentPortTypeRole" 
portType="tns:SendDocumentPortType"/>
    </plnk:partnerLinkType>
</definitions>
```

4. Build eInsight / BPEL 1.0-based Web Service

Let's create a Java CAPS Repository-based / Classic project, WSSReceiveDocument. Figures 4-1 and 4-2 illustrate the steps.

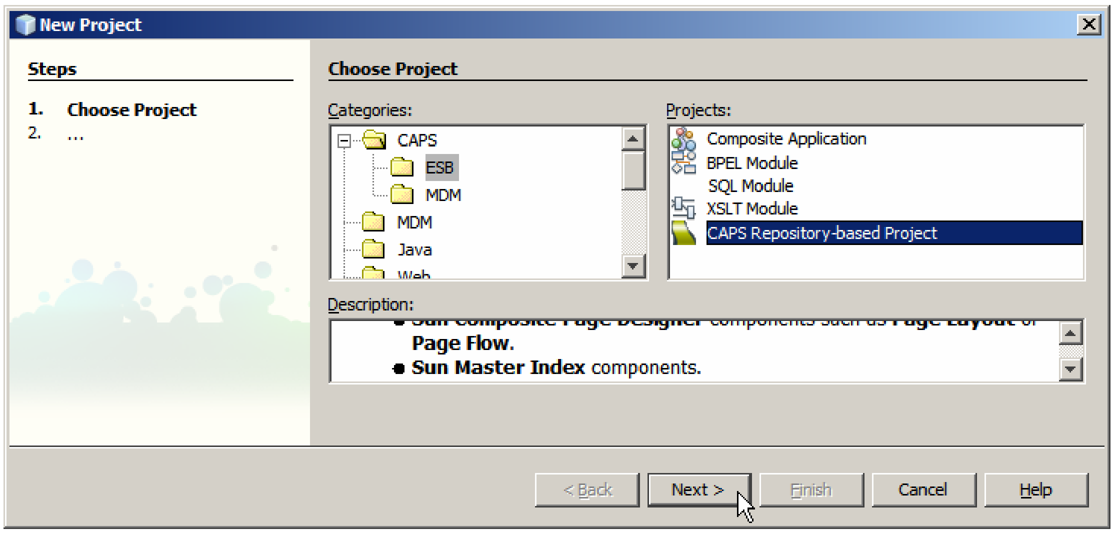
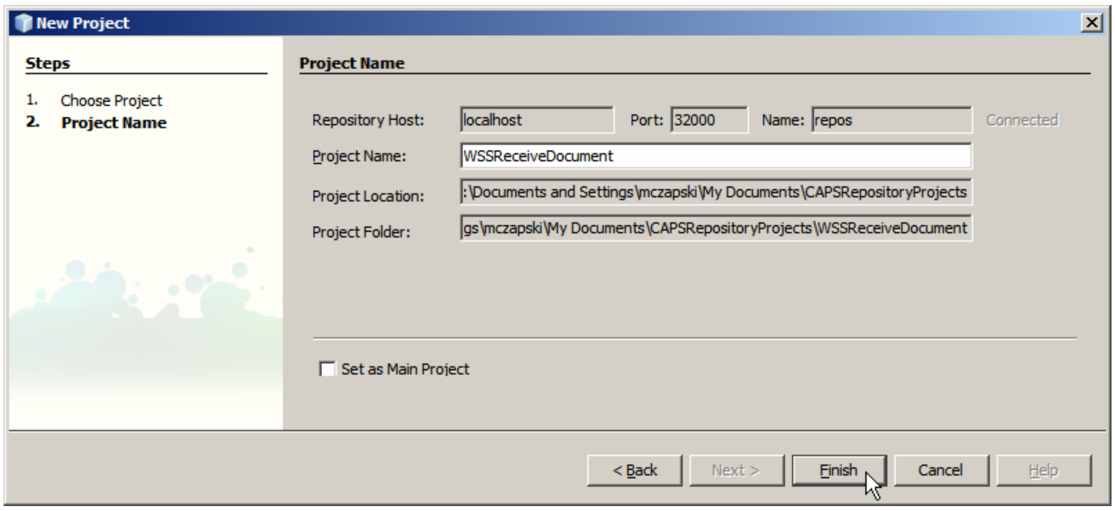


Figure 4-1 Creating Java CAPS Repository-based Project



Repository Host: localhost Port: 32000 Name: repos Connected

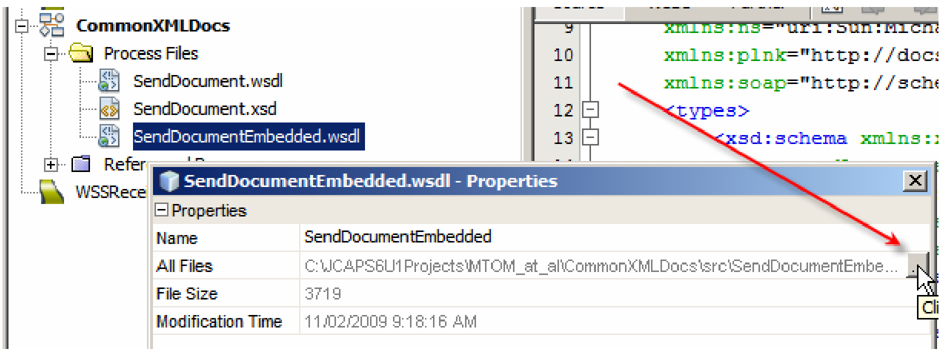
Project Name: WSSReceiveDocument

Project Location: :\Documents and Settings\mczapski\My Documents\CAPSRepositoryProjects

Project Folder: gs\mczapski\My Documents\CAPSRepositoryProjects\WSSReceiveDocument

Figure 4-2 Naming the project

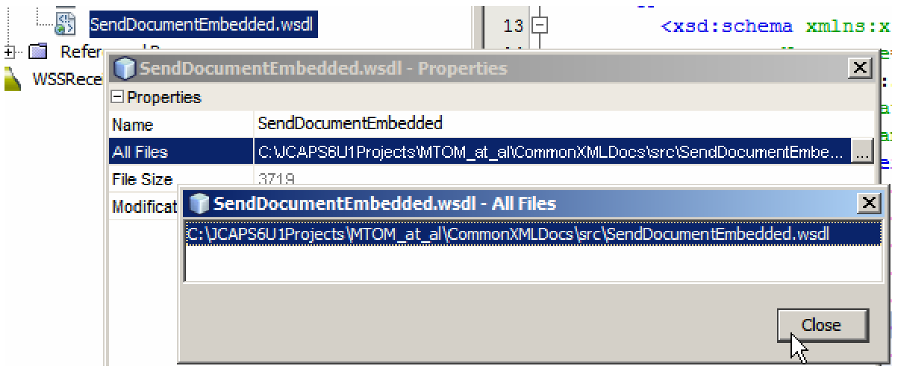
Let's now import the Web Service Definition (WSDL) from the file system, which is where the artefacts in the CommonXMLDocs project reside. Before starting the Wizard let's copy to clipboard the location of the WSDL document. Righ-click the name of the WSDL in CommonXMLDocs/Process Files/SendDocumentEmbedded.wsdl and choose Properties. Click the small button with the ellipsis at the rightmost end of the "All Files" field. See Figure 4-3.



Name: SendDocumentEmbedded
All Files: C:\JCAPS6U1Projects\MTOM_at_al\CommonXMLDocs\src\SendDocumentEmbe...
File Size: 3719
Modification Time: 11/02/2009 9:18:16 AM

Figure 4-3 Open the All Files Property box

Select the WSDL location and copy to clipboard as shown in Figure 4-4.


SendDocumentEmbedded.wsdl - Properties
SendDocumentEmbedded.wsdl - All Files
Close

C:\JCAPS6U1Projects\MTOM_at_al\CommonXMLDocs\src\SendDocumentEmbedded.wsdl

Figure 4-4 Copy the WSDL location to the clipboard

Dismiss both dialogue boxes. Right-click the name of the Classic project and choose Import->Web Service Definition. Figure 4-5 illustrates this.

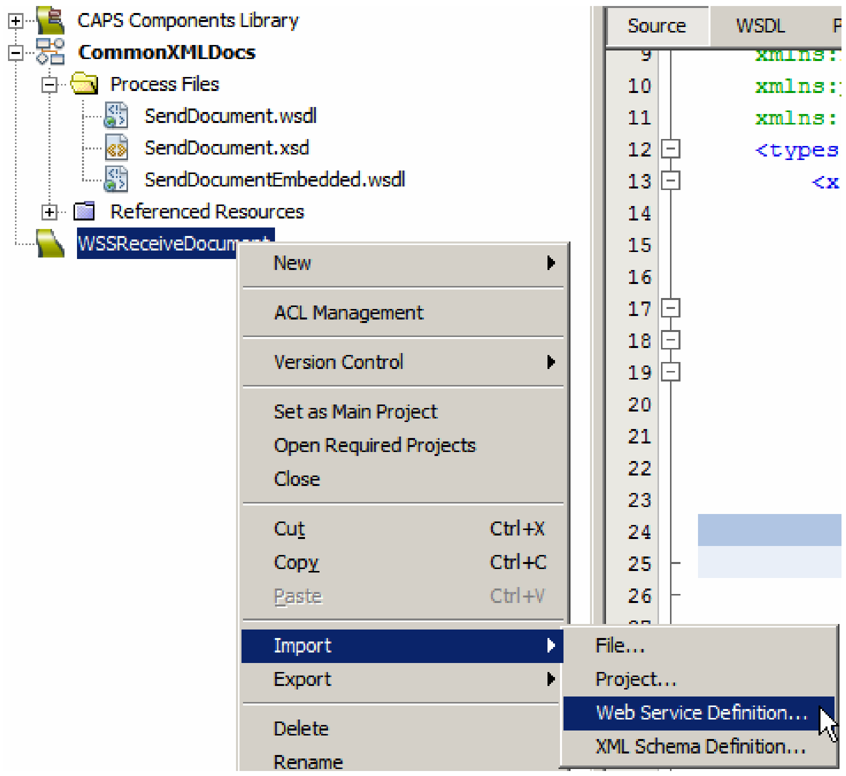
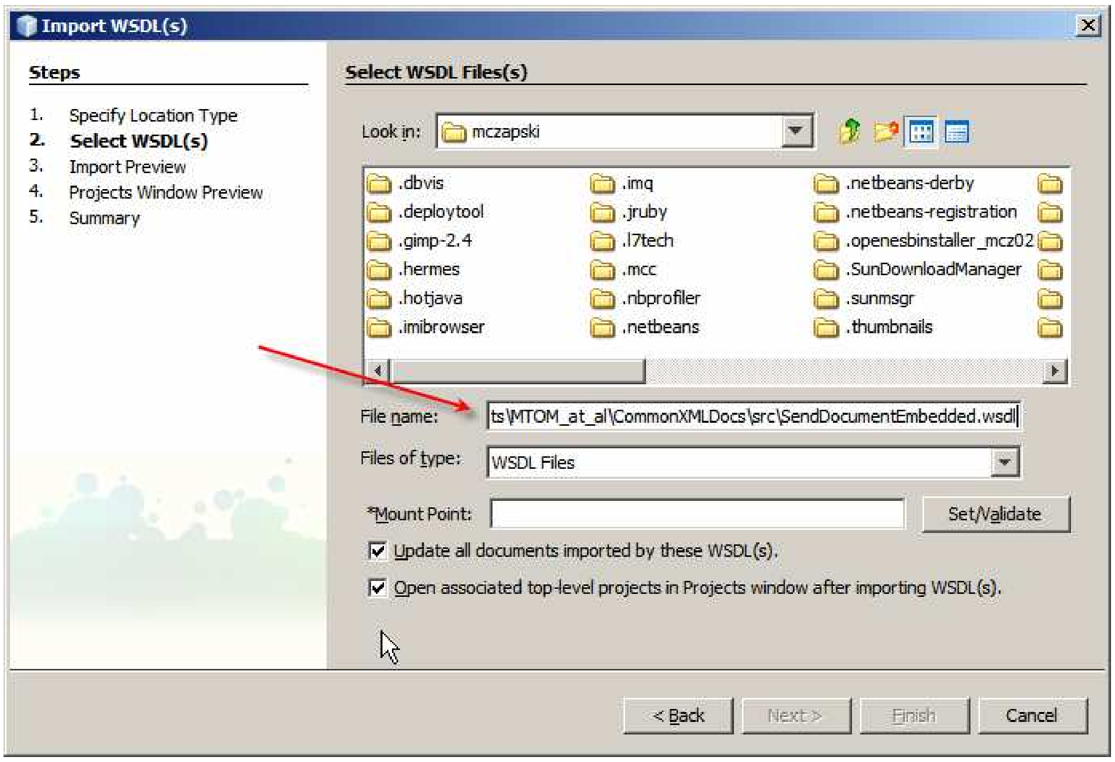


Figure 4-5 Import WSDL Wizard – step 1

Accept default “File System” location type by clicking Next.

Paste the file path into the data entry field and press Enter, see Figure 4-6. Until you do the Next button will be disabled.



File name: ts\MTOM_at_al\CommonXMLDocs\src\SendDocumentEmbedded.wsdl

Files of type: WSDL Files

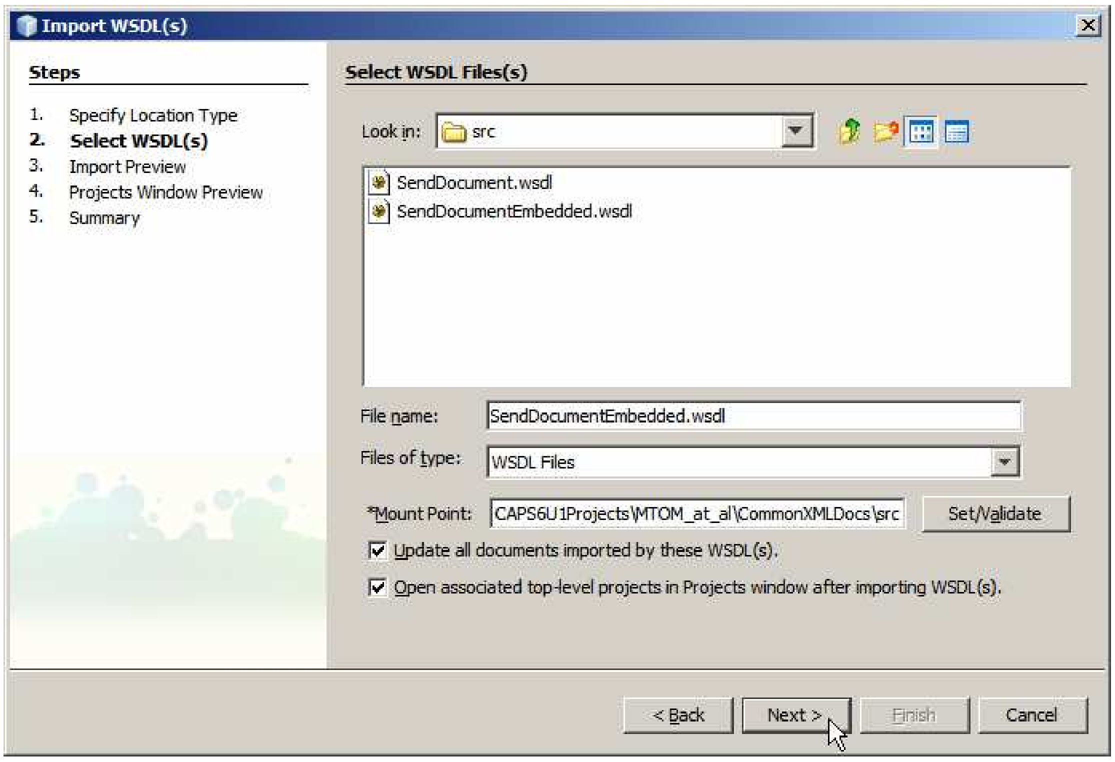
*Mount Point:

☑ Update all documents imported by these WSDL(s).

☑ Open associated top-level projects in Projects window after importing WSDL(s).

Figure 4-6 Paste the WSDL location

Click Next, as shown in Figure 4-7.



File name: SendDocumentEmbedded.wsdl

Files of type: WSDL Files

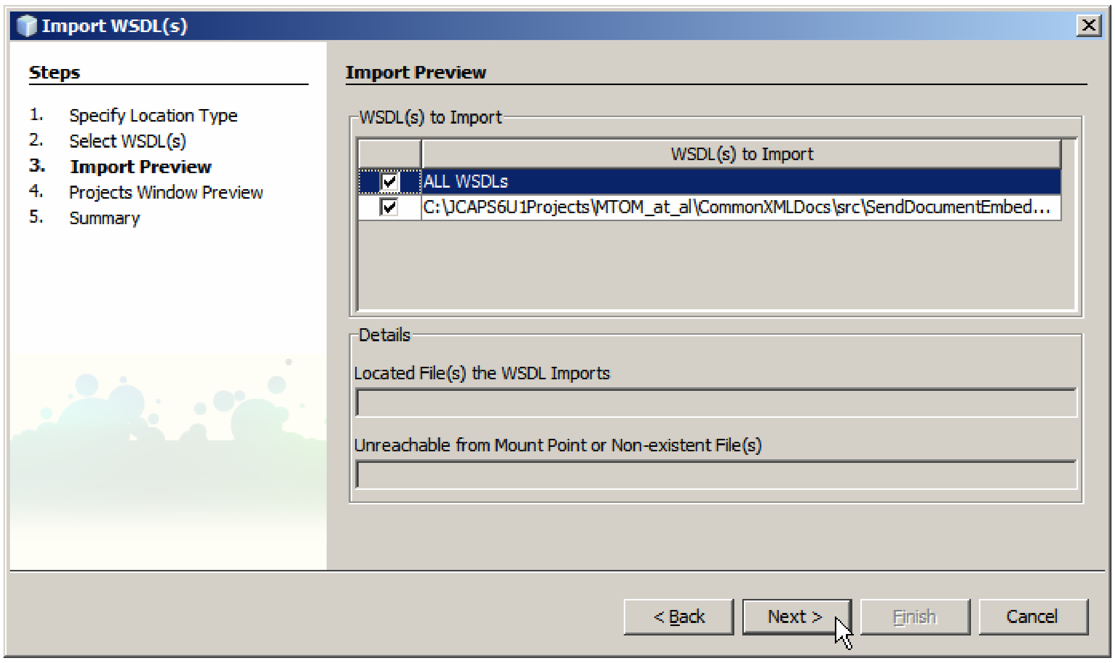
*Mount Point: CAPS6U1Projects\MTOM_at_al\CommonXMLDocs\src

☑ Update all documents imported by these WSDL(s).

☑ Open associated top-level projects in Projects window after importing WSDL(s).

Figure 4-7 Accept selected WSDL

If there are no errors, click Next, as shown in Figure 4-8. If there are errors then sort them out and do this agai.



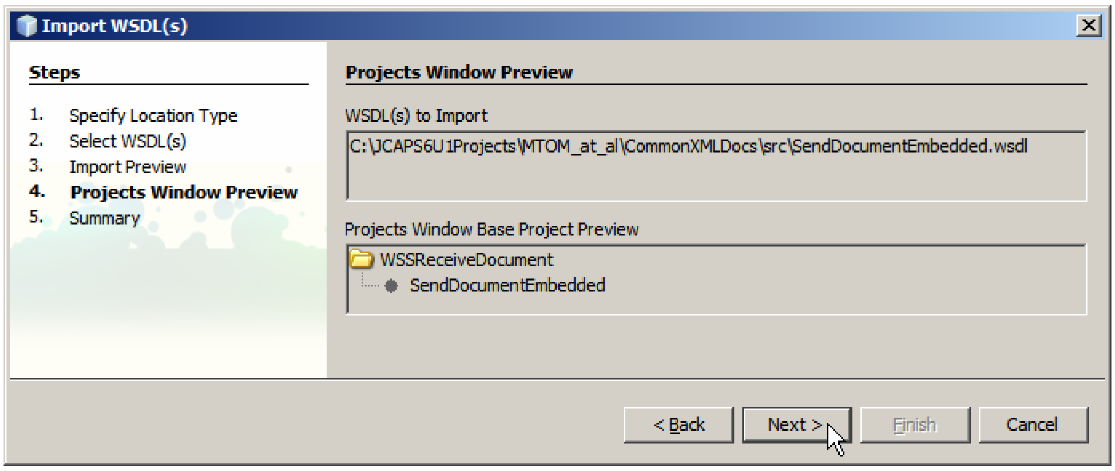
	WSDL(s) to Import
✓	ALL WSDLs
✓	C:\JCAPS6U1Projects\MTOM_at_al\CommonXMLDocs\src\SendDocumentEmbed...

Located File(s) the WSDL Imports

Unreachable from Mount Point or Non-existent File(s)

Figure 4-8 Accept what is offered

Accept what is offered by clicking Next, as illustrated in Figure 4-9.



WSDL(s) to Import

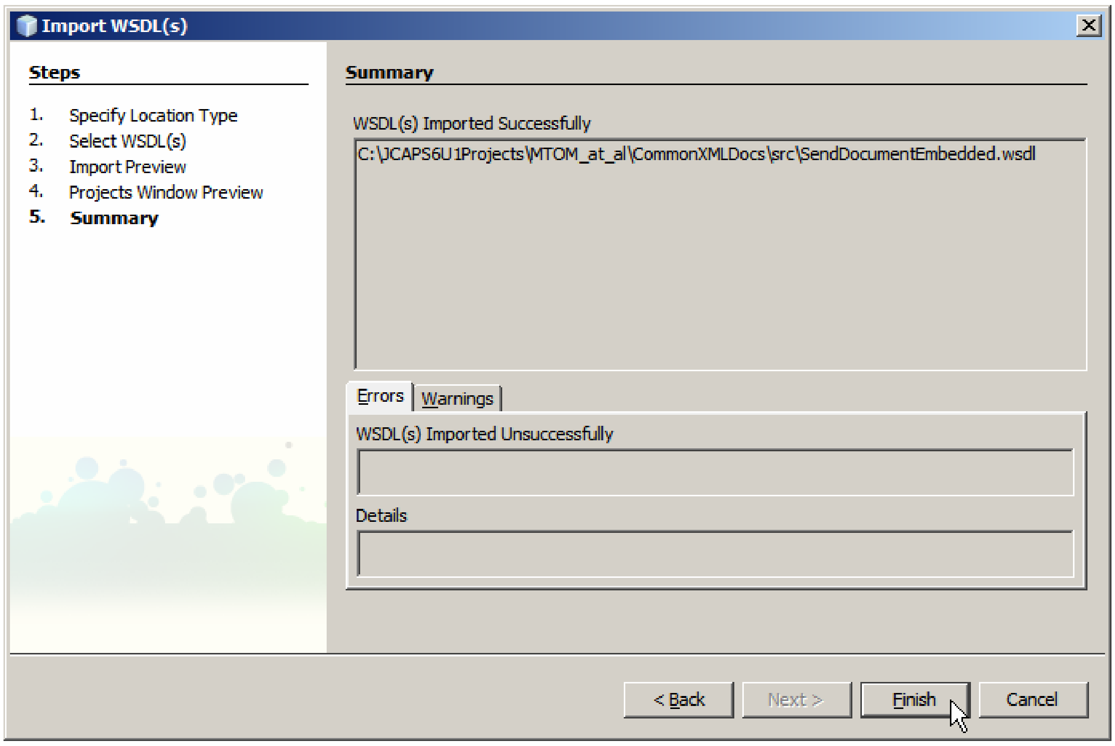
C:\JCAPS6U1Projects\MTOM_at_al\CommonXMLDocs\src\SendDocumentEmbedded.wsdl

Projects Window Base Project Preview

WSSReceiveDocument
SendDocumentEmbedded

Figure 4-9 Accept what is offered

Assuming no errors or warnings are present, click Finish, as illustrated in Figure 4-10.



WSDL(s) Imported Successfully

C:\JCAPS6U1Projects\MTOM_at_al\CommonXMLDocs\src\SendDocumentEmbedded.wsdl

WSDL(s) Imported Unsuccessfully

Details

Figure 4-10 Complete Wizard

The process above produces a Classic WSDL artefact, shown in Figure 4-11.

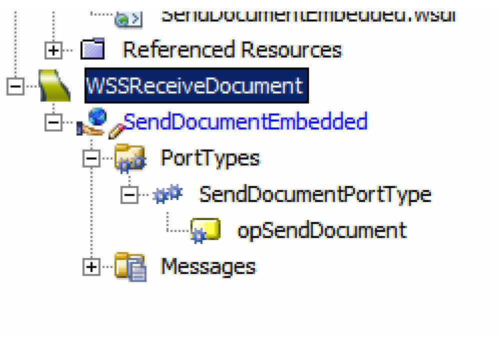


Figure 4-11 Classic WSDL artefact

Create a new Business Process. Figure 4-12 illustrates the menu options involved.

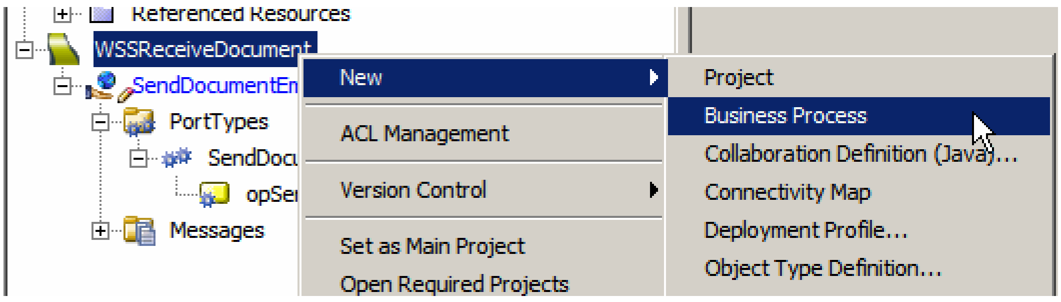


Figure 4-12 Create a new Classic Business Process

To implement the service let’s drag the Web Service operation onto the business process canvas and choose the “Implement” option. Figures 4-13 and 4-14 illustrate the steps.

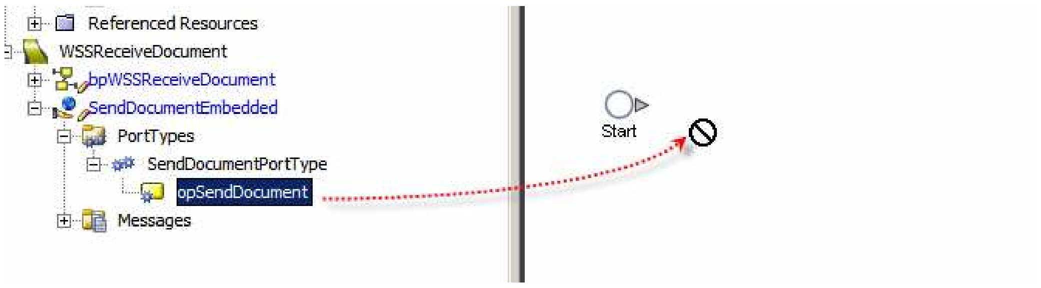
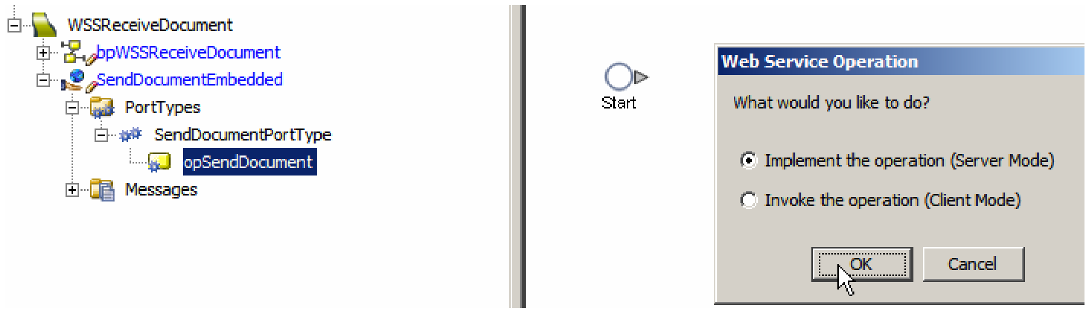


Figure 4-13 Drag the web service operation onto the BP canvas



What would you like to do?

[✓ Implement the operation (Server Mode)]

Invoke the operation (Client Mode)

Figure 4-14 Choose "Implement …"

Connect the activities together, add a Business Rule and map. Figure 4-15 illustrates this.

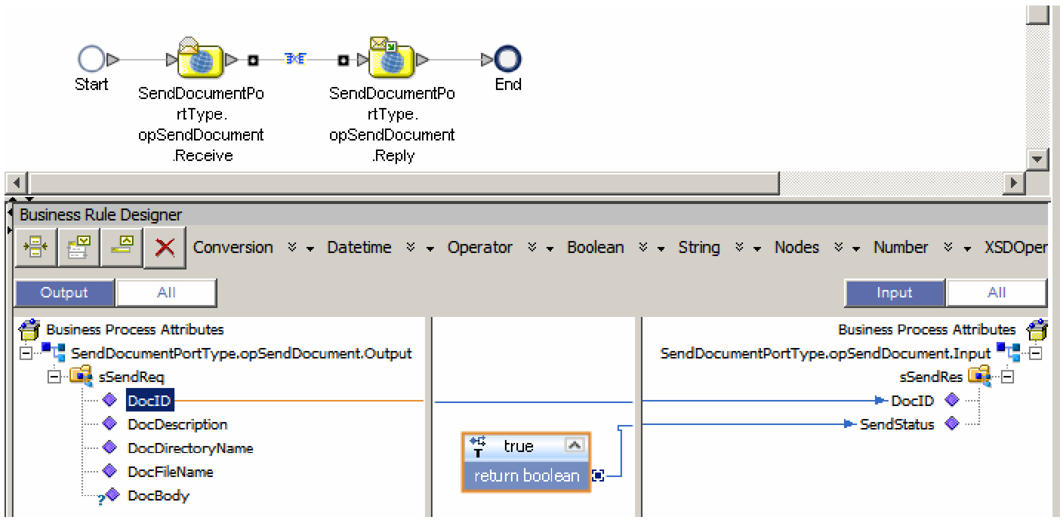

Start
SendDocumentPortType.opSendDocument.Receive
SendDocumentPortType.opSendDocument.Reply
End
Business Rule Designer
Conversion
Datetime
Operator
Boolean
String
Nodes
Number
XSDOper
Output
All
Input
All
Business Process Attributes
SendDocumentPortType.opSendDocument.Output
sSendReq
DocID
DocDescription
DocDirectoryName
DocFileName
DocBody
true
return boolean
Business Process Attributes
SendDocumentPortType.opSendDocument.Input
sSendRes
DocID
SendStatus

Figure 4-15 Map Web Service request data to Web Service Response nodes

The implementation is deliberately simple since the business logic does not really matter in this case.

Let's create the Connectivity Map, drag the business process onto the CM canvas, add the Web Service Connector, connect and configure it. Figure 4-16 shows the completed connectivity map.

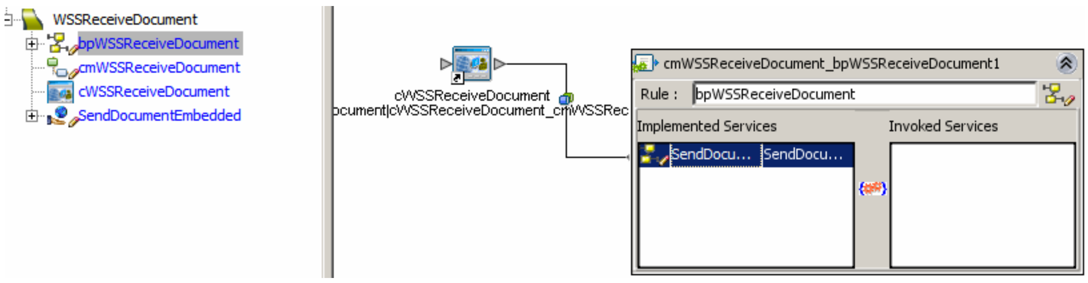


Figure 4-16 cmWSSReceiveDocument Connectivity Map

Assume you have created a Java CAPS Environment as discussed in Section 8, “

Create Java CAPS Environment”.

Add a new SOAP/HTTP Web Service External System container, WSSReceiveDocument, and configure it as appropriate to your environment. Figures 4-17 through 4-19 illustrate this for my environment.

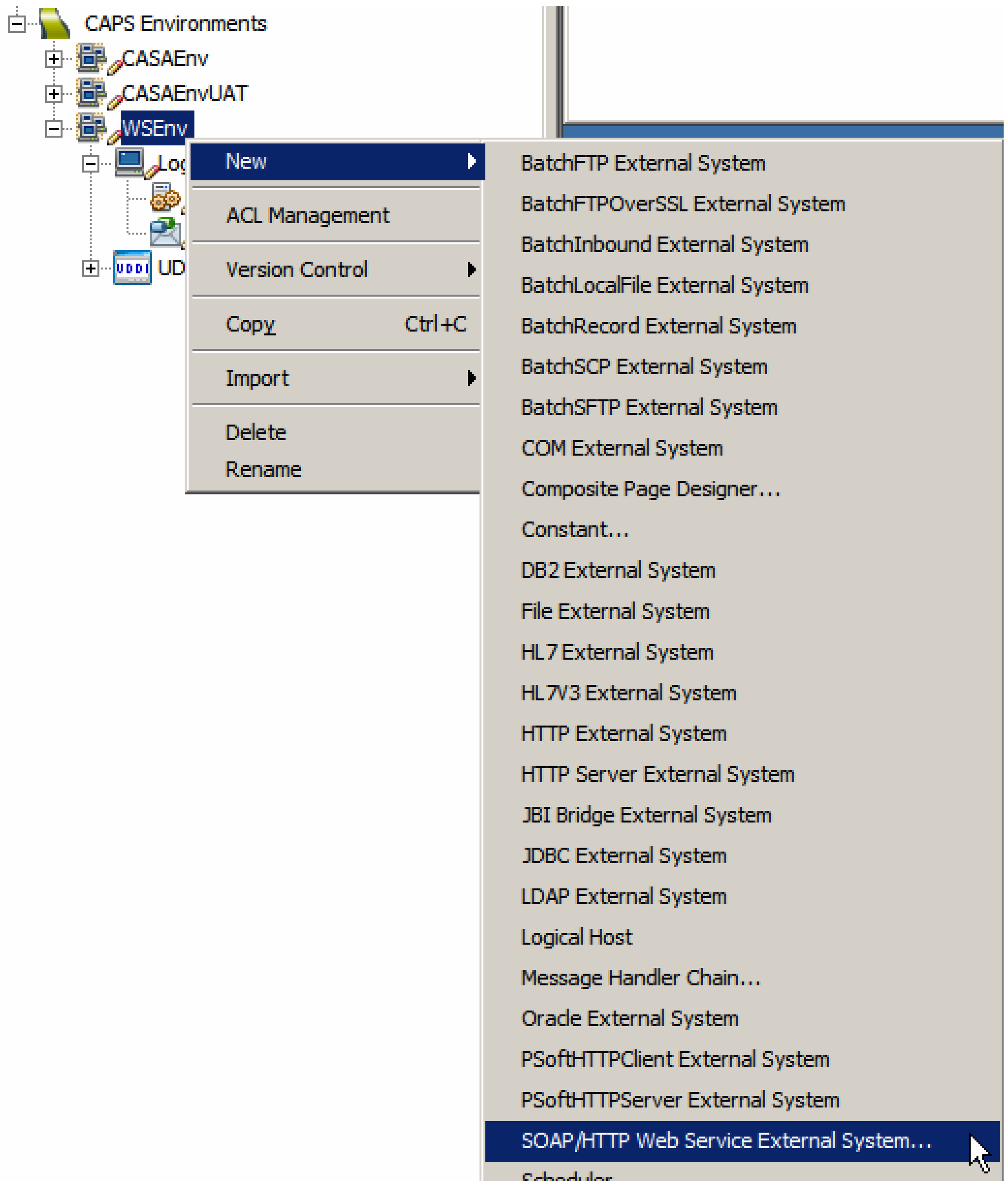
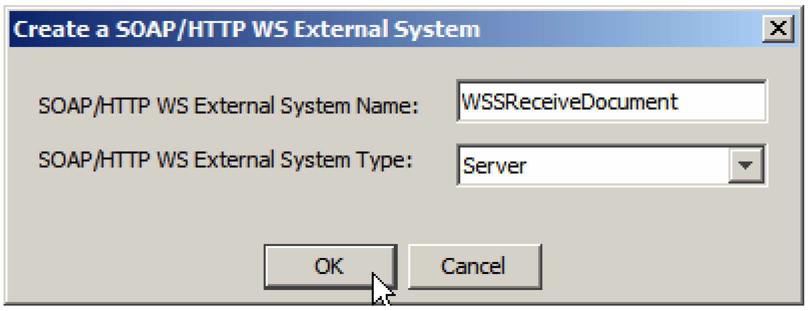


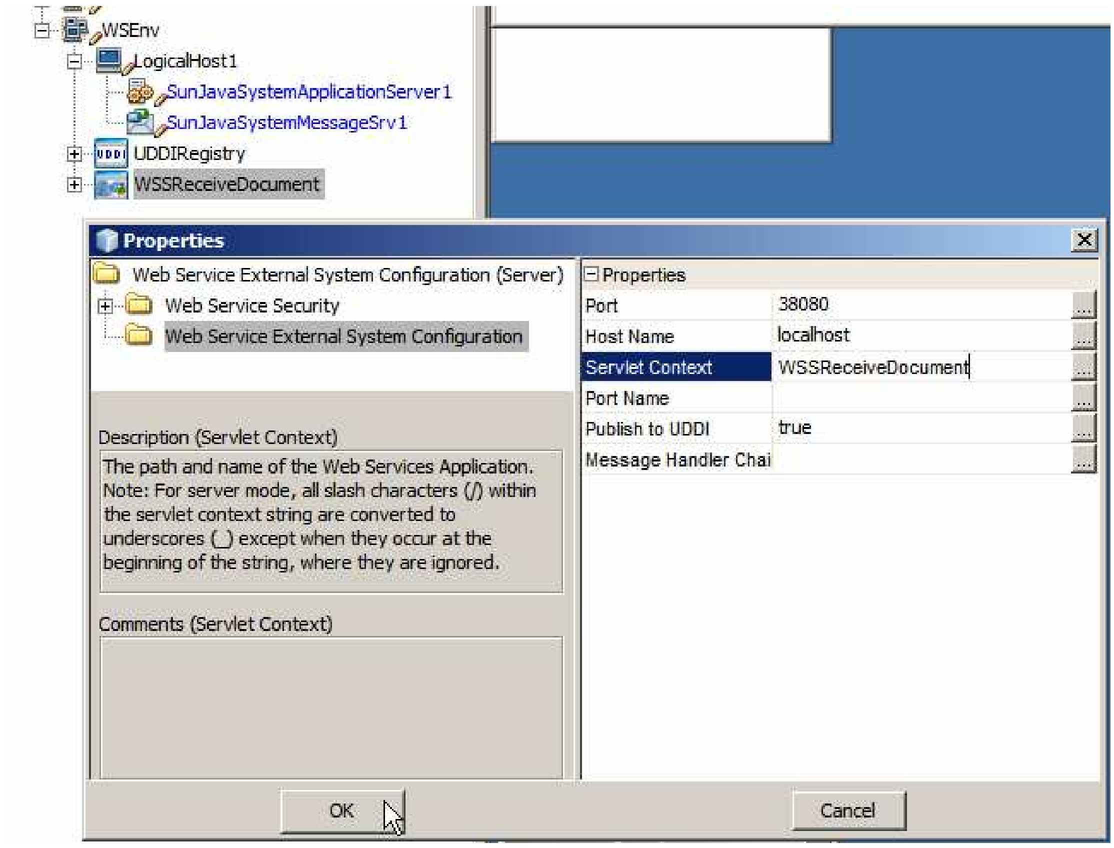
Figure 4-17 Create a SOAP/HTTP Web Service External System …


Create a SOAP/HTTP WS External System

SOAP/HTTP WS External System Name: WSSReceiveDocument

SOAP/HTTP WS External System Type: Server

Figure 4-18 Name the container and choose container type


WSEnv
LogicalHost1
SunJavaSystemApplicationServer1
SunJavaSystemMessageSrv1
UDDIRegistry
WSSReceiveDocument

Properties

Web Service External System Configuration (Server)
Web Service Security
Web Service External System Configuration

Properties	
Port	38080
Host Name	localhost
Servlet Context	WSSReceiveDocument
Port Name	
Publish to UDDI	true
Message Handler Chai	

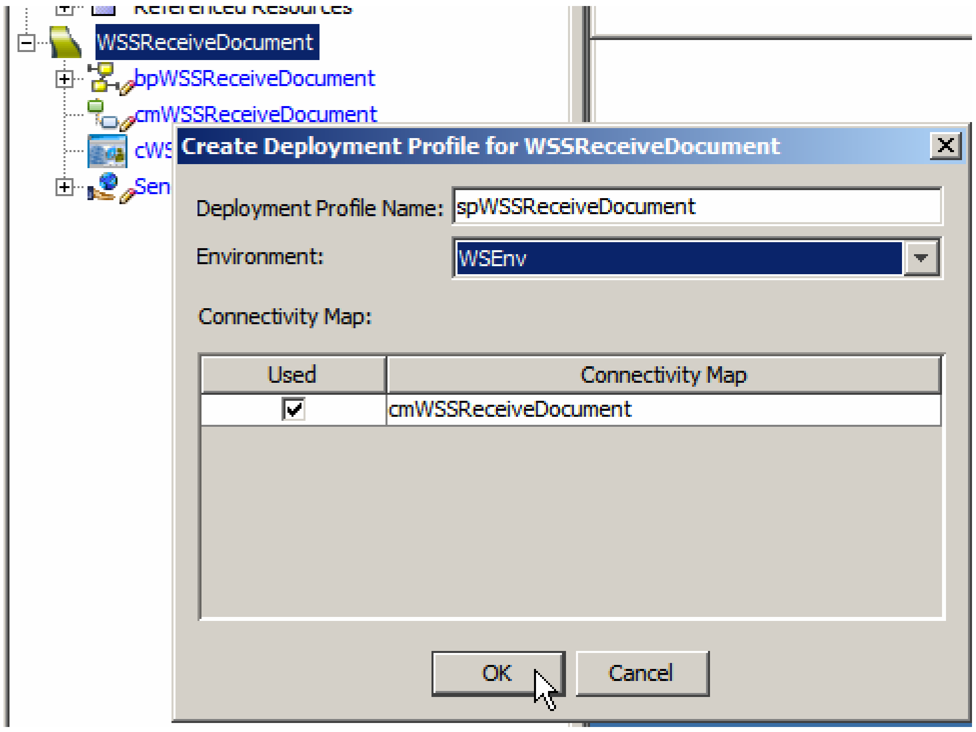
Description (Servlet Context)
The path and name of the Web Services Application. Note: For server mode, all slash characters (/) within the servlet context string are converted to underscores (_) except when they occur at the beginning of the string, where they are ignored.

Comments (Servlet Context)

OK Cancel

Figure 4-19 Configure host, port and servlet context

The Java CAPS Environment now has all the containers necessary to deploy the project we have been working on so far. Let’s create the Deployment Profile, WSSReceiveDocument, build and deploy the project. Figure 4-20 illustrates a step in this process. As the deployment is built choose to publish the WSDL to UDDI.


Referenced Resources

WSSReceiveDocument

bpWSSReceiveDocument

cmWSSReceiveDocument

cWS

Sen

Create Deployment Profile for WSSReceiveDocument

Deployment Profile Name: spWSSReceiveDocument

Environment: WSEnv

Connectivity Map:

Used	Connectivity Map
✓	cmWSSReceiveDocument

Figure 4-20 Choose Deployment Profile name and the environment to which it will belong

The service is deployed. Now let’s exercise it to make sure it works.

5. Exercise eInsight / BPEL 1.0-based Web Service

The modern NetBenas IDE, used in Java CAPS 6, provides a number of ways to test web services. The most straight-forward is to install the SoapUI Plugin, which is doable through the Tools->Plugins-accessible Plugin Manager.

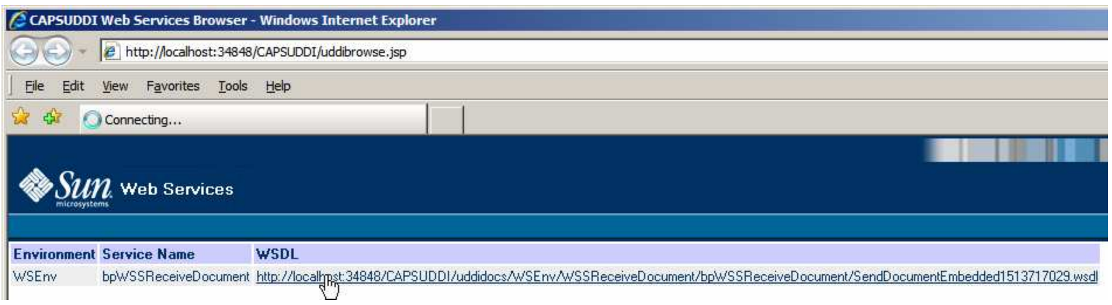
Installation of the SoapUI Plugin is discussed in section 10, “

Install soapUI Plugin”.

If the plugin is installed a new Enterprise Project Category, Web Service Testing Project, becomes available in the “New Project” wizard.

To configure a Web Service Testing Project we will need a URL or a file system location of a WSDL that corresponds to the web service to be tested. This we can obtain through the “UDDI Browser” web-based UI, typically at http://<yourUDDIHost>:<YourUDDIPort>/CAPSUDDI/uddibrowse.jsp.

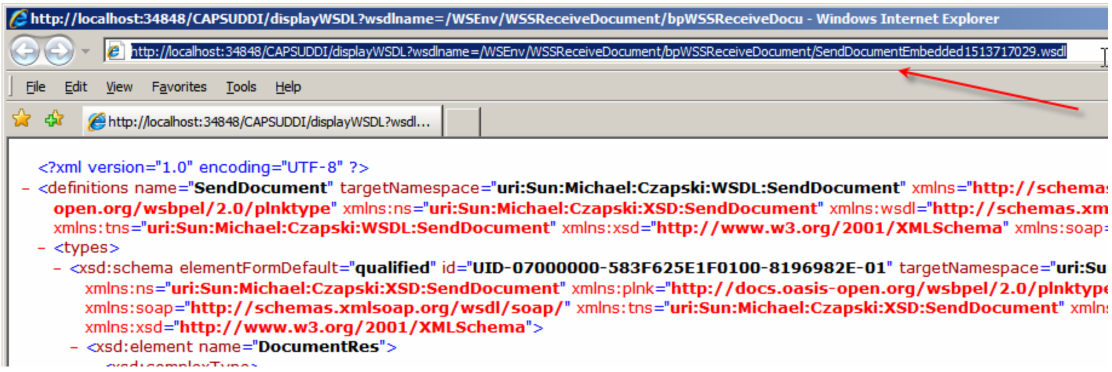
Make sure to select and copy the WSDL URL.


CAPSUDDI Web Services Browser - Windows Internet Explorer
http://localhost:34848/CAPSUDDI/uddibrowse.jsp
File Edit View Favorites Tools Help
Connecting...
Sun microsystems Web Services

Environment	Service Name	WSDL
WSEnv	bpWSSReceiveDocument	http://localhost:34848/CAPSUDDI/uddidocs/WSEnv/WSSReceiveDocument/bpWSSReceiveDocument/SendDocumentEmbedded1513717029.wsdl

Figure 5-1 Click on the WSDL URL

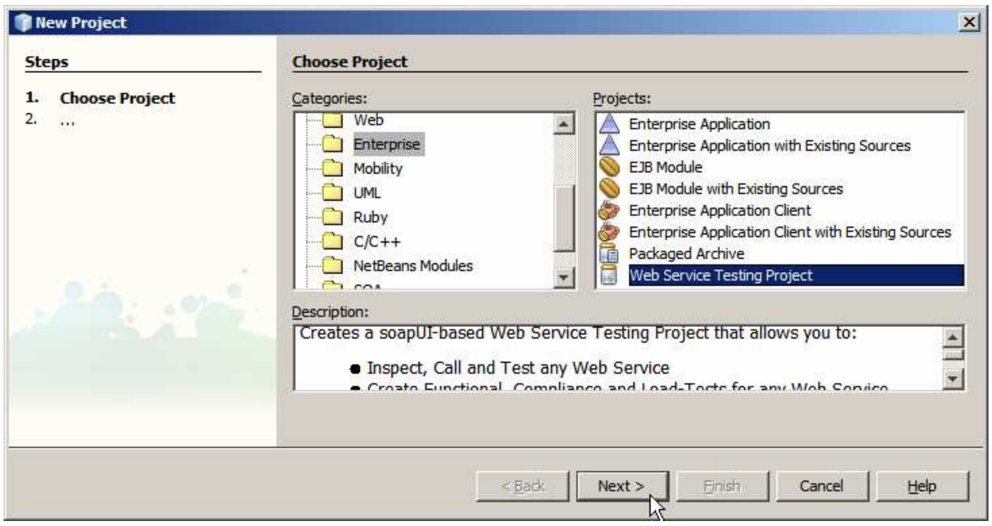
Select the URL.


http://localhost:34848/CAPSUDDI/displayWSDL?wsdlname=/WSEnv/WSSReceiveDocument/bpWSSReceiveDocu - Windows Internet Explorer
http://localhost:34848/CAPSUDDI/displayWSDL?wsdlname=/WSEnv/WSSReceiveDocument/bpWSSReceiveDocument/SendDocumentEmbedded1513717029.wsdl
File Edit View Favorites Tools Help
http://localhost:34848/CAPSUDDI/displayWSDL?wsdl...

```
  <?xml version="1.0" encoding="UTF-8" ?>
- <definitions name="SendDocument" targetNamespace="uri:Sun:Michael:Czapski:WSDL:SendDocument" xmlns="http://schema
    open.org/wsbpel/2.0/plnktype" xmlns:ns="uri:Sun:Michael:Czapski:XSD:SendDocument" xmlns:wsdl="http://schemas.xm
    xmlns:tns="uri:Sun:Michael:Czapski:WSDL:SendDocument" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:soap=
  - <types>
    - <xsd:schema elementFormDefault="qualified" id="UID-07000000-583F625E1F0100-8196982E-01" targetNamespace="uri:Su
        xmlns:ns="uri:Sun:Michael:Czapski:XSD:SendDocument" xmlns:plnk="http://docs.oasis-open.org/wsbpel/2.0/plnktype
        xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:tns="uri:Sun:Michael:Czapski:XSD:SendDocument" xmln
        xmlns:xsd="http://www.w3.org/2001/XMLSchema">
      - <xsd:element name="DocumentRes">
```

Figure 5-2 Select the URL and copy it to the clipboard

Create a new Web Service testing Project, WSSReceiveDocumentWSTP.


New Project

Steps
1. Choose Project
2. ...

Choose Project

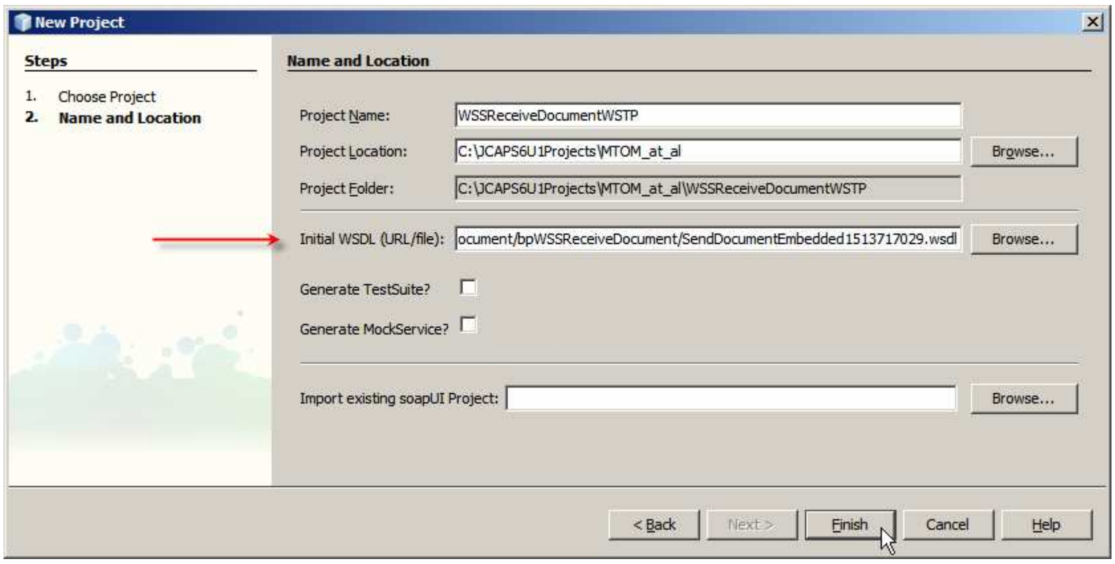
Categories: Web, Enterprise, Mobility, UML, Ruby, C/C++, NetBeans Modules, SOA

Projects: Enterprise Application, Enterprise Application with Existing Sources, EJB Module, EJB Module with Existing Sources, Enterprise Application Client, Enterprise Application Client with Existing Sources, Packaged Archive, Web Service Testing Project

Description:
Creates a soapUI-based Web Service Testing Project that allows you to:
- Inspect, Call and Test any Web Service
- Create Functional, Compliance and Load-Tests for any Web Service

Figure 5-3 New Web Service Testing Project

Name the Project and paste the WSDL URL into the Initial WSDL (URL/File).


New Project

Steps
1. Choose Project
2. Name and Location

Name and Location

Project Name: WSSReceiveDocumentWSTP
Project Location: C:\JCAPS6U1Projects\MTOM_at_al
Project Folder: C:\JCAPS6U1Projects\MTOM_at_al\WSSReceiveDocumentWSTP
Initial WSDL (URL/file): ocument/bpWSSReceiveDocument/SendDocumentEmbedded1513717029.wsdl
Generate TestSuite? ☐
Generate MockService? ☐
Import existing soapUI Project:

Figure 5-4 Paste the URL into the box.

Once the WSDL is parsed and imported, expand the project structure and open “Request 1” for editing.

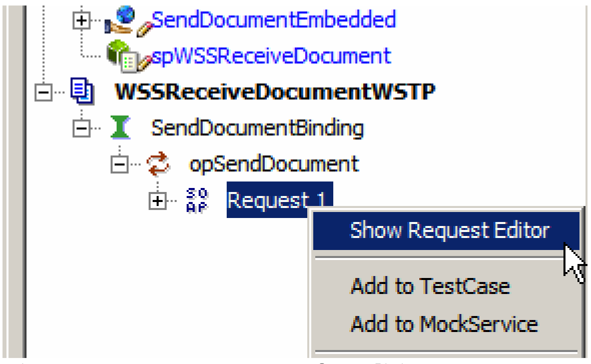
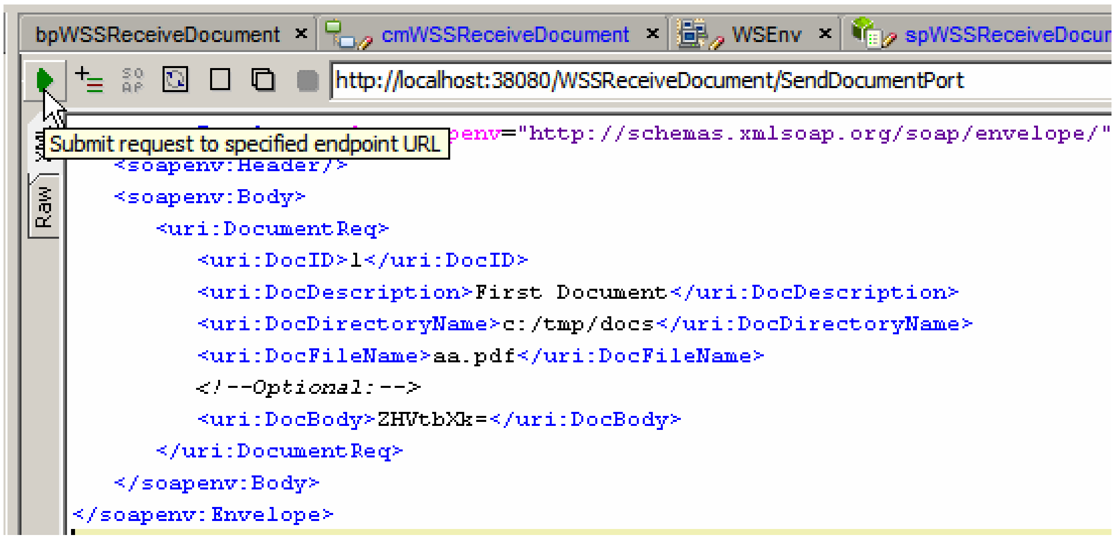


Figure 5-5 Open Request 1 for editing

Complete the request by providing data for the fields. Paste the following string, not including quotes, into the optional DocBody “ZHVtbXk=”. Submit the request.

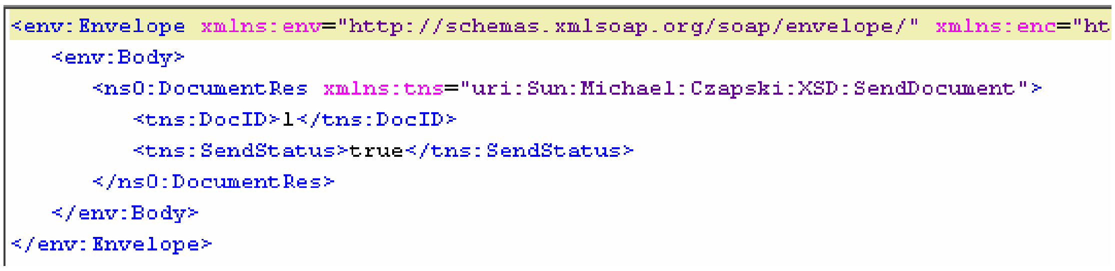


http://localhost:38080/WSSReceiveDocument/SendDocumentPort

```
penv="http://schemas.xmlsoap.org/soap/envelope/"
  <soapenv:Header/>
  <soapenv:Body>
     <uri:DocumentReq>
        <uri:DocID>1</uri:DocID>
        <uri:DocDescription>First Document</uri:DocDescription>
        <uri:DocDirectoryName>c:/tmp/docs</uri:DocDirectoryName>
        <uri:DocFileName>aa.pdf</uri:DocFileName>
        <!--Optional:-->
        <uri:DocBody>ZHVtbXk=</uri:DocBody>
     </uri:DocumentReq>
  </soapenv:Body>
</soapenv:Envelope>
```

Figure 5-6 Configure SOAP Request

The response returned by the service should look like this:

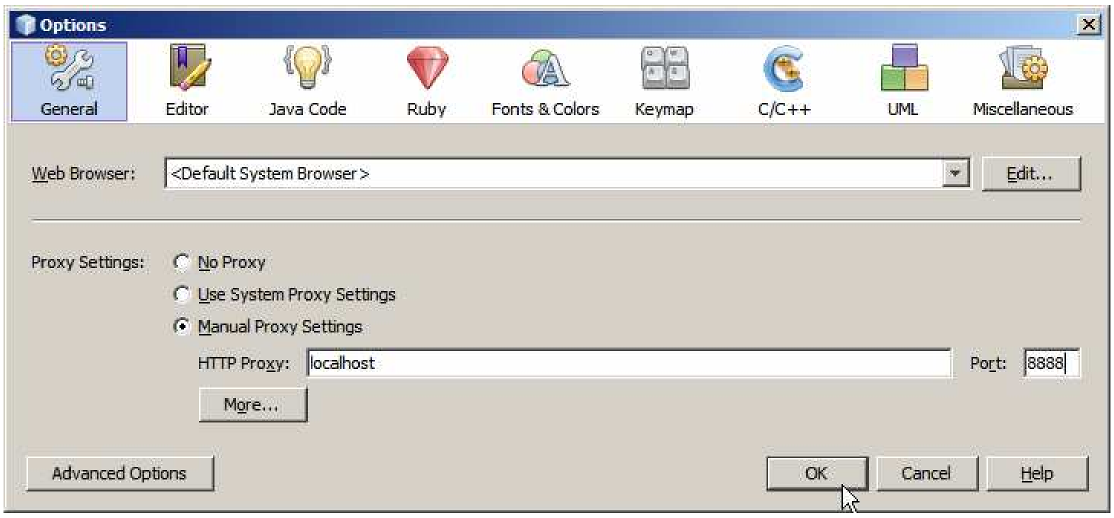


```
<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/" xmlns:enc="ht
   <env:Body>
      <ns0:DocumentRes xmlns:tns="uri:Sun:Michael:Czapski:XSD:SendDocument">
         <tns:DocID>1</tns:DocID>
         <tns:SendStatus>true</tns:SendStatus>
      </ns0:DocumentRes>
   </env:Body>
</env:Envelope>
```

Figure 5-7 SOAP Response

Let’s now start the TCP Mon as a proxy listening on port 8888, see section 9, “

Obtain and use the Apache TCP Mon", and configure NetBeans to use that as the Web Proxy. Tools->Options->General: Manual Proxy Settings.



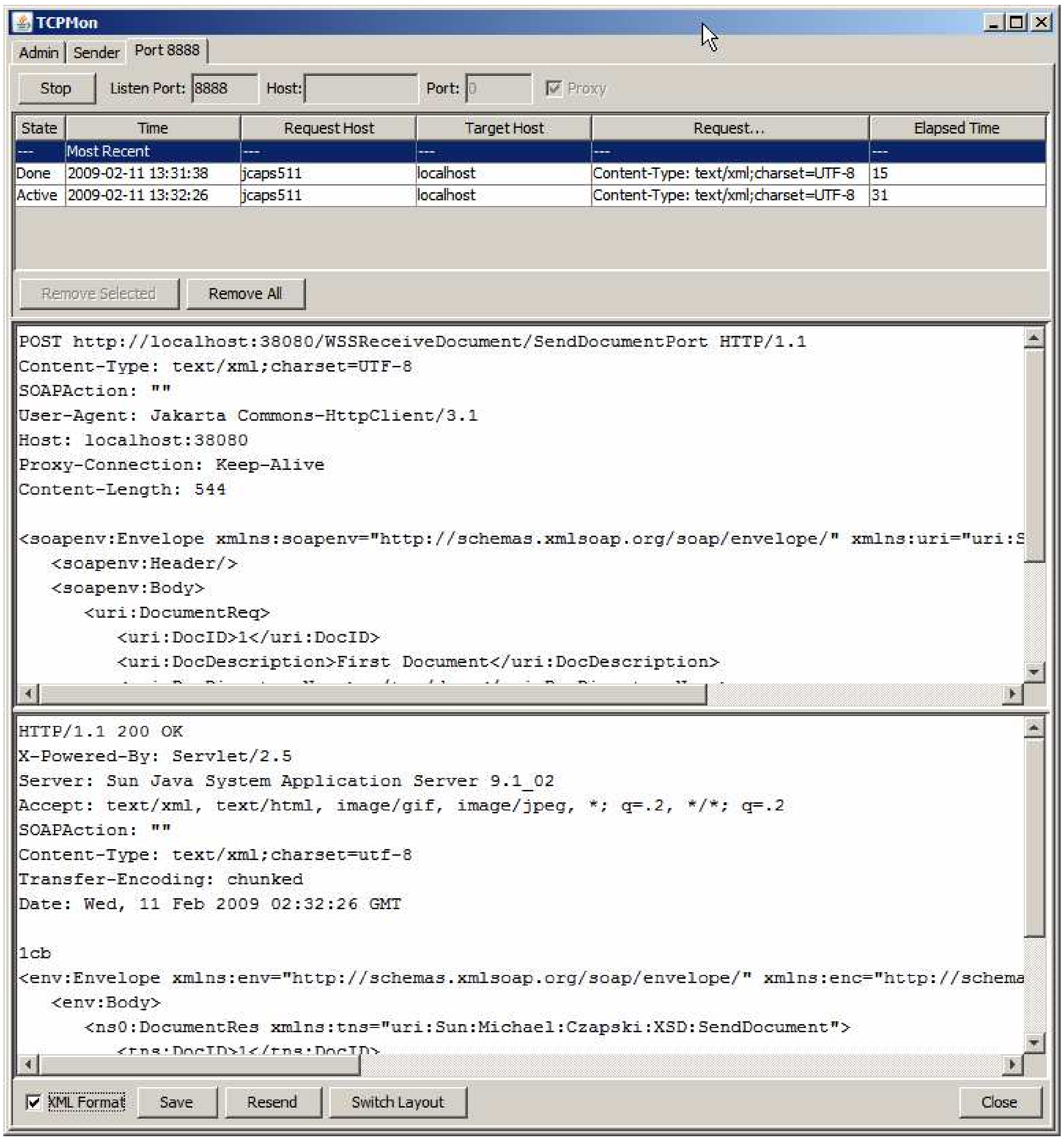
Web Browser: <Default System Browser>

Proxy Settings: ( ) No Proxy ( ) Use System Proxy Settings [✓ Manual Proxy Settings]

HTTP Proxy: localhost Port: 8888

Figure 5-8 Configure Web Proxy

Submit the request again and view the TCP Mon display, noticing the SOA Request and the SOAP Response.



Listen Port: 8888 Host: Port: 0 Proxy

State	Time	Request Host	Target Host	Request...	Elapsed Time
---	Most Recent	---	---	---	---
Done	2009-02-11 13:31:38	jcaps511	localhost	Content-Type: text/xml;charset=UTF-8	15
Active	2009-02-11 13:32:26	jcaps511	localhost	Content-Type: text/xml;charset=UTF-8	31

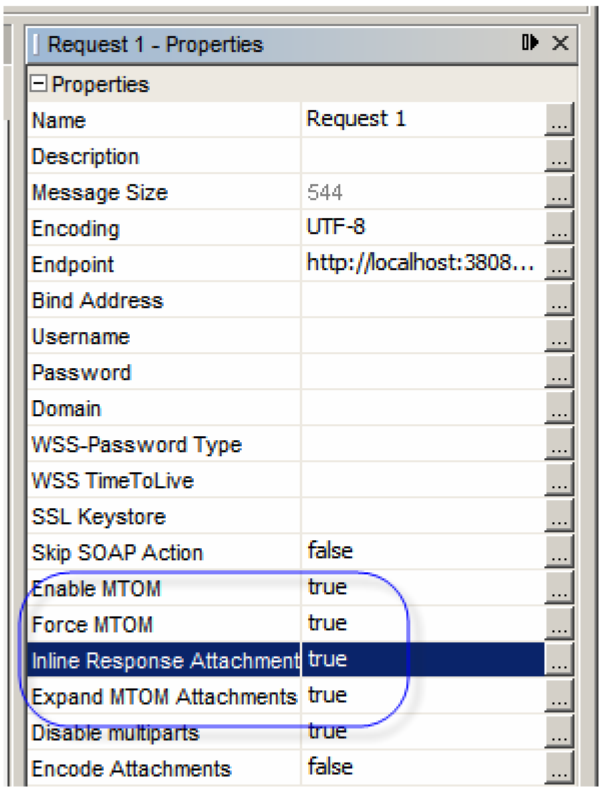
```
POST http://localhost:38080/WSSReceiveDocument/SendDocumentPort HTTP/1.1
Content-Type: text/xml;charset=UTF-8
SOAPAction: ""
User-Agent: Jakarta Commons-HttpClient/3.1
Host: localhost:38080
Proxy-Connection: Keep-Alive
Content-Length: 544

<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:uri="uri:S
   <soapenv:Header/>
   <soapenv:Body>
      <uri:DocumentReq>
         <uri:DocID>1</uri:DocID>
         <uri:DocDescription>First Document</uri:DocDescription>
```

```
HTTP/1.1 200 OK
X-Powered-By: Servlet/2.5
Server: Sun Java System Application Server 9.1_02
Accept: text/xml, text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2
SOAPAction: ""
Content-Type: text/xml;charset=utf-8
Transfer-Encoding: chunked
Date: Wed, 11 Feb 2009 02:32:26 GMT

1cb
<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/" xmlns:enc="http://schema
   <env:Body>
      <ns0:DocumentRes xmlns:tns="uri:Sun:Michael:Czapski:XSD:SendDocument">
```

Figure 5-9 SOAP Request (top) and SOAP Response (bottom)

Let’s now enable MTOM support in the soapUI plugin and enforce it. Select the “Reqeust 1” node in the web service testing project. Pull down the Windows menu and select “Properties”. This will open the Properties window pane. Make sure the four attachment related properties are configured as true, Enable MTOM, Force MTOM, inline Response Attachment and Expand MTOM Attachments.

Submit the request again and observe the SOAP FAULT that is returned.



Request 1 - Properties

Properties

Name	Request 1
Description	
Message Size	544
Encoding	UTF-8
Endpoint	http://localhost:3808...
Bind Address	
Username	
Password	
Domain	
WSS-Password Type	
WSS TimeToLive	
SSL Keystore	
Skip SOAP Action	false
Enable MTOM	true
Force MTOM	true
Inline Response Attachment	true
Expand MTOM Attachments	true
Disable multiparts	true
Encode Attachments	false

Figure 5-10 MTOM Attachment-related properties

The SOAP Fault is shown in Listing 5-1.

Listing 5-1 SOAP Fault – Element not allowed: Include

```
<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:enc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:ns0="uri:Sun:Michael:Czapski:XSD:SendDocument"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <env:Body>
      <env:Fault>
         <faultcode>env:Server</faultcode>
         <faultstring>Internal server runtime exception</faultstring>
         <detail>
            <ans1:SOAPFaultMessage
xmlns="http://seebeyond/com/xsddefined/FaultMessages"
xmlns:ans1="http://seebeyond/com/xsddefined/FaultMessages">
               <ans1:Fault>
                  <faultcode xmlns="">SERVER_ERROR</faultcode>
                  <faultstring xmlns="">root
cause:com.stc.otd.runtime.UnmarshalException: error: Element not allowed:
Include@http://www.w3.org/2004/08/xop/include in element
DocBody@uri:Sun:Michael:Czapski:XSD:SendDocument  and the associated Fault container
is not available</faultstring>
                  <faultactor xmlns="">wsserver</faultactor>
                  <detail xmlns="">root cause:com.stc.otd.runtime.UnmarshalException:
error: Element not allowed: Include@http://www.w3.org/2004/08/xop/include in element
DocBody@uri:Sun:Michael:Czapski:XSD:SendDocument  and the associated Fault container
is not available</detail>
               </ans1:Fault>
            </ans1:SOAPFaultMessage>
         </detail>
      </env:Fault>
   </env:Body>
</env:Envelope>
```

What happened?

The soapUI plugin, which supports MTOM, was told to optimise binary data in the SOAP Request using the Message Transmission Optimisation Mechanism (MTOM) standard. It did so. The base64-encoded content of the DocBody node was decoded, moved to a MIME part and replaced with the “Include ….” reference. The request was transformed into that shown in Listing 5-2.

Listing 5-2 SOAP Request transformed according to MTOM spec.

```
POST http://localhost:38080/WSSReceiveDocument/SendDocumentPort HTTP/1.1
SOAPAction: ""
Content-Type: multipart/related; type="application/xop+xml";
start="<rootpart@soapui.org>"; start-info="text/xml"; boundary="----
=_Part_2_837984.1234319921406"MIME-Version: 1.0User-Agent: Jakarta Commons-
HttpClient/3.1Host: localhost:38080Proxy-Connection: Keep-AliveContent-Length: 1039---
---=_Part_2_837984.1234319921406Content-Type: application/xop+xml; charset=UTF-8;
type="text/xml"Content-Transfer-Encoding: 8bitContent-ID:
   <rootpart@soapui.org>
      <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:uri="uri:Sun:Michael:Czapski:XSD:SendDocument">
         <soapenv:Header/>
         <soapenv:Body>
            <uri:DocumentReq>
               <uri:DocID>1</uri:DocID>
               <uri:DocDescription>First Document</uri:DocDescription>
               <uri:DocDirectoryName>c:/tmp/docs</uri:DocDirectoryName>
               <uri:DocFileName>aa.pdf</uri:DocFileName>
               <!--Optional:-->
                  <uri:DocBody>
                     <inc:Include href="cid:http://www.soapui.org/105621903315644"
xmlns:inc="http://www.w3.org/2004/08/xop/include"/>
                  </uri:DocBody>
               </uri:DocumentReq>
            </soapenv:Body>
         </soapenv:Envelope>------=_Part_2_837984.1234319921406Content-Type:
application/octet-streamContent-Transfer-Encoding: binaryContent-ID:
         <http://www.soapui.org/105621903315644>dummy------
=_Part_2_837984.1234319921406--
```

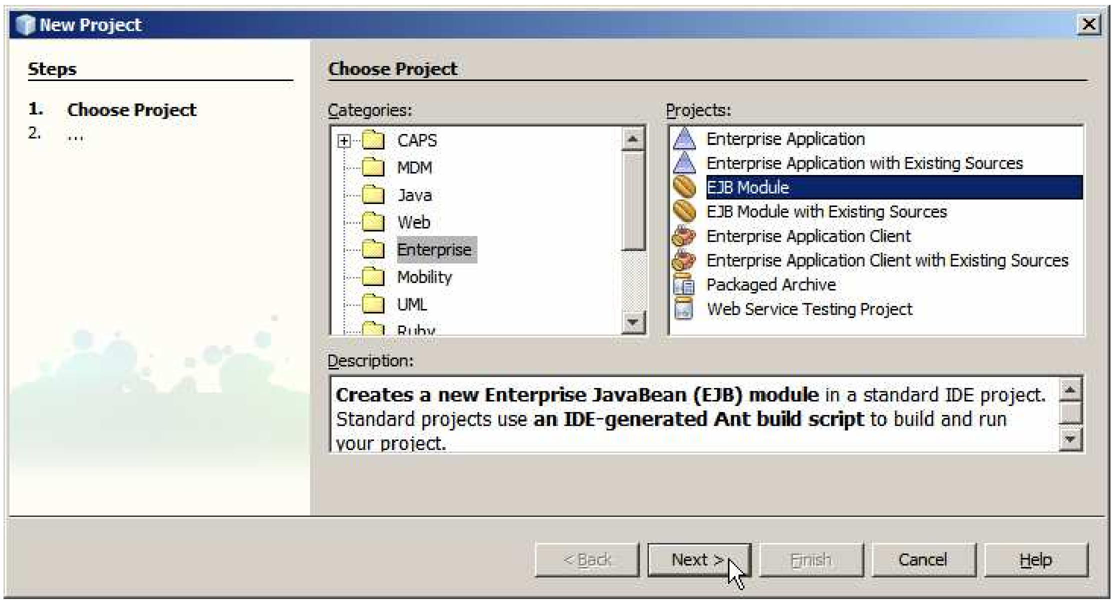
Note the content ID in the part and the content ID in the body of the DocBody node.

Since the service as implemented does not support MTOM the request failed to be processed and caused the SOAP Fault to be returned.

6. Add Service Wrapper to provide MTOM support

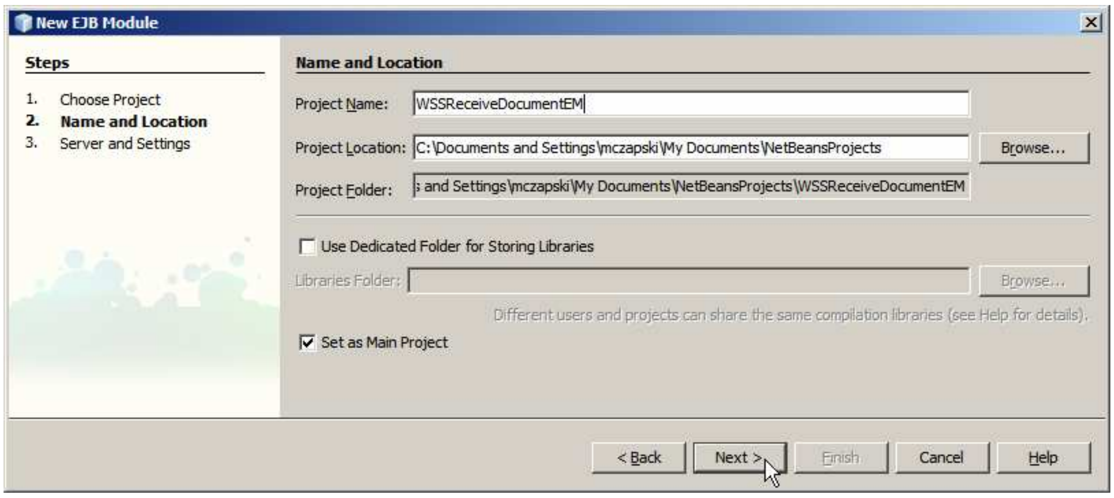
There is no way to configure MTOM support for the Java CAPS Repository-based eInsight-based Web Service directly. There are no configuration properties to do that and there is no code support in the framework. One possibility is to use the SOAP Handler mechanism, available since Java CAPS 5.1.3 Rollup 2, but that requires a deal of low level Java coding and is not for the faint of heart. The simpler way in Java CAPS 6 is to create an EJB-based Web Service as a service wrapper / proxy for the eInsight-based web service. This EJB-based web service would be invoked by the consumer and it, in its turn, would invoke the service proper.

Let's create an EJB Module Project, WSSReceiveDocumentEM.



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 6-1 New EJB Module Project



Project Name: WSSReceiveDocumentEM

Project Location: C:\Documents and Settings\mczapski\My Documents\NetBeansProjects

Project Folder: s and Settings\mczapski\My Documents\NetBeansProjects\WSSReceiveDocumentEM

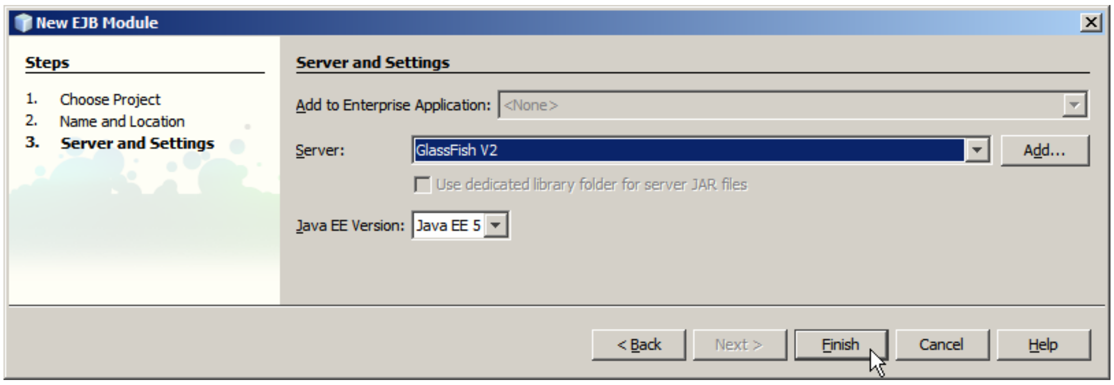
☐ Use Dedicated Folder for Storing Libraries

Libraries Folder:

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

☑ Set as Main Project

Figure 6-2 Name the project and specify location for its artefacts


New EJB Module

Steps
1. Choose Project
2. Name and Location
3. **Server and Settings**

Server and Settings

Add to Enterprise Application: <None>

Server: GlassFish V2

Use dedicated library folder for server JAR files

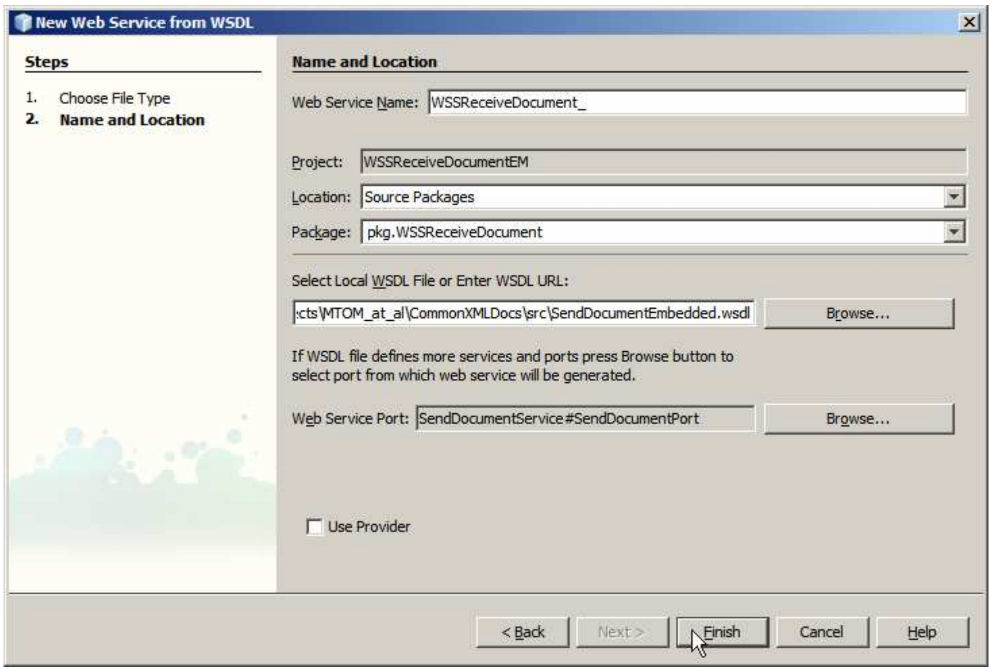
Java EE Version: Java EE 5

Figure 6-3 Finish

Right-click CommonXMLDocs/Process Files/SendDocumentEmbedded.wsdl, choose Properties and copy the "All Files" property value to the clipboard.

Let's now create a New -> Web Service from WSDL … (or New -> Other -> Web Services -> Web Service from WSDL`).

Name the Web Service WSSReceiveDocument, provide a package name, for example pkg.WSSReceiveDocument, and provide the WSDL location.


New Web Service from WSDL

Steps
1. Choose File Type
2. **Name and Location**

Name and Location

Web Service Name: WSSReceiveDocument_

Project: WSSReceiveDocumentEM

Location: Source Packages

Package: pkg.WSSReceiveDocument

Select Local WSDL File or Enter WSDL URL:

:cts\MTOM_at_al\CommonXMLDocs\src\SendDocumentEmbedded.wsdl

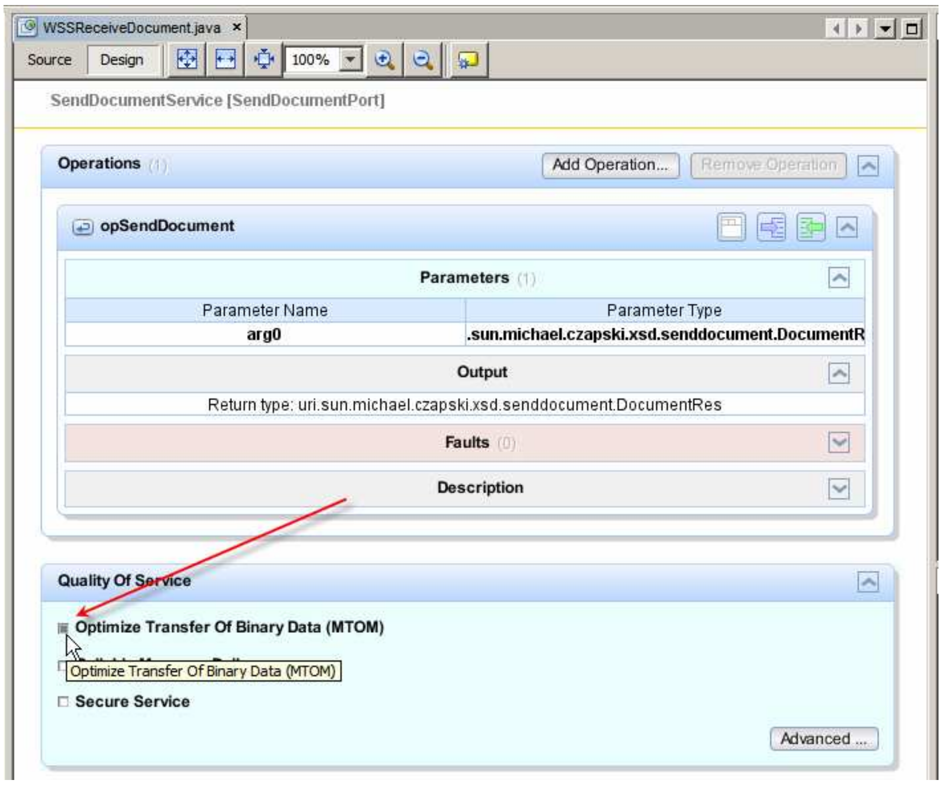
If WSDL file defines more services and ports press Browse button to select port from which web service will be generated.

Web Service Port: SendDocumentService#SendDocumentPort

Use Provider

Figure 6-4 Configure new web service initial properties

The service implementation skeleton opens in Design Mode. Check the Optimize Transfer of Binary Data (MTOM) checkbox to enable MTOM support, as shown in Figure 6-5.


WSSReceiveDocument.java

SendDocumentService [SendDocumentPort]

Operations (1)

opSendDocument

Parameters (1)

Parameter Name	Parameter Type
arg0	.sun.michael.czapski.xsd.senddocument.DocumentR

Output

Return type: uri.sun.michael.czapski.xsd.senddocument.DocumentRes

Faults (0)

Description

Quality Of Service

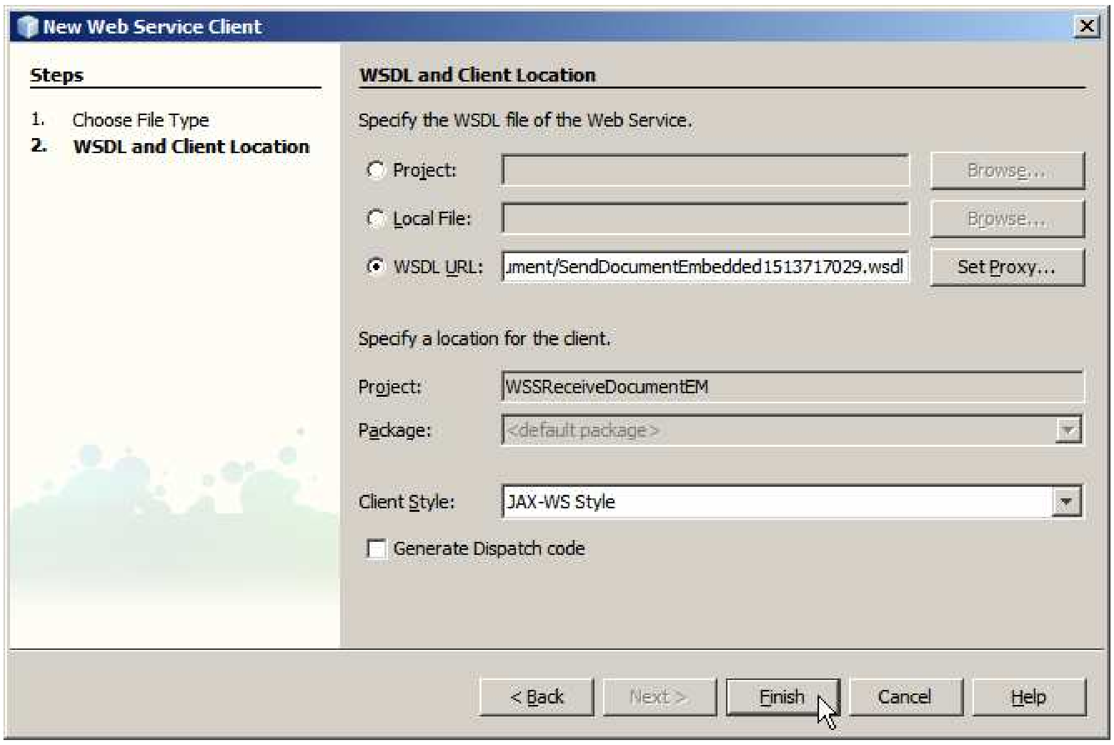
Optimize Transfer Of Binary Data (MTOM)

Secure Service

Figure 6-5 Enable MTOM support

Use the UDDI Browser servlet to get a copy of the URL pointing to the WSDL for the Repository-based Web Service we already deployed, see Figure 5-1 and Figure 5-2.

Right-click on the name of the project, WSSRecevieFileEM, and choose New->Web Service Client … Check the WSDL URL radio button and paste the WSDL URL. Click Finish to complete the wizard. This is illustrated in Figure 6-6.


New Web Service Client

Steps

1. Choose File Type
2. **WSDL and Client Location**

WSDL and Client Location

Specify the WSDL file of the Web Service.

Project:

Local File:

WSDL URL: ument/SendDocumentEmbedded1513717029.wsdl

Specify a location for the client.

Project: WSSReceiveDocumentEM

Package: <default package>

Client Style: JAX-WS Style

Generate Dispatch code

Figure 6-6 Create Web Service Client from the existing service's WSDL

Note a new node, Web Service Reference, appear in the project's node tree, Figure 6-7.

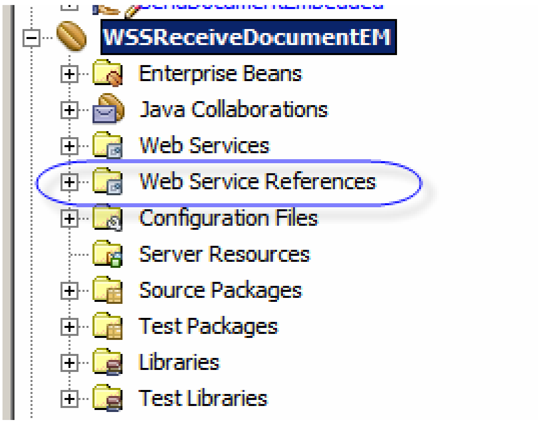
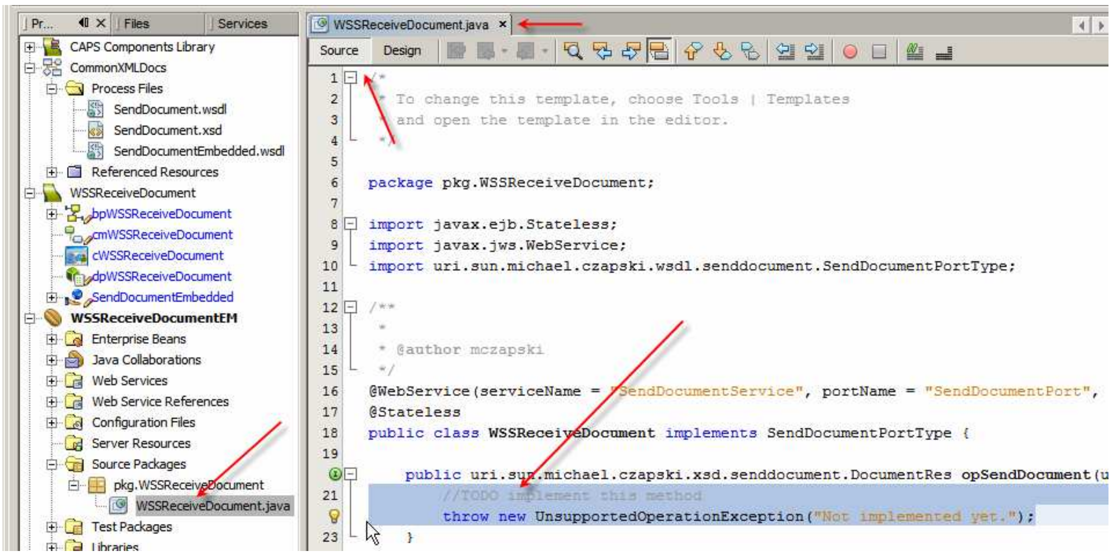


Figure 6-7 Web Service Reference added to the project

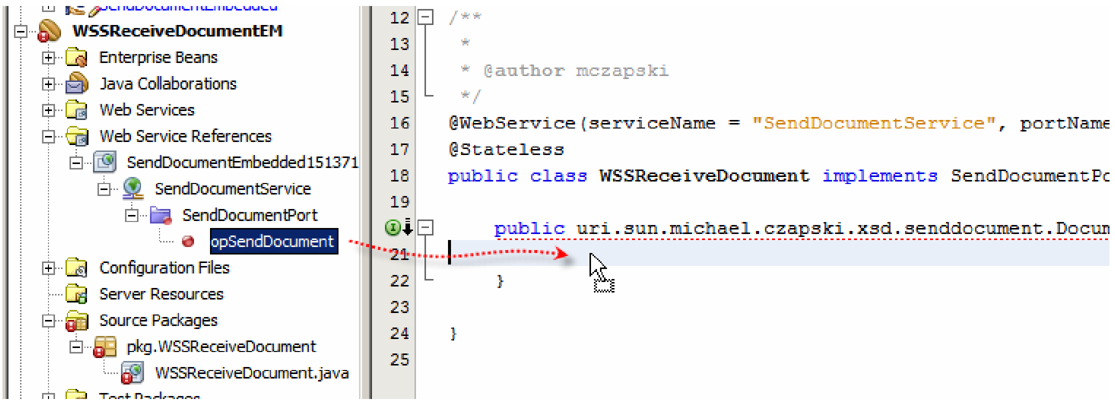
Locate the Java source of the WSSRecevieDocument web service, open it, switch to Source view and select the two lines of code inside the opSendDocument method. Delete the two lines of code by pressing Enter. This is illustrated in Figure 6-8.



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

package pkg.WSSReceiveDocument;

import javax.ejb.Stateless;
import javax.jws.WebService;
import uri.sun.michael.czapski.wsdl.senddocument.SendDocumentPortType;

/**
 *
 * @author mczapski
 */
@WebService(serviceName = "SendDocumentService", portName = "SendDocumentPort",
@Stateless
public class WSSReceiveDocument implements SendDocumentPortType {

    public uri.sun.michael.czapski.xsd.senddocument.DocumentRes opSendDocument(u
        //TODO implement this method
        throw new UnsupportedOperationException("Not implemented yet.");
    }
```

Figure 6-8 Select the code to replace with the real implementation

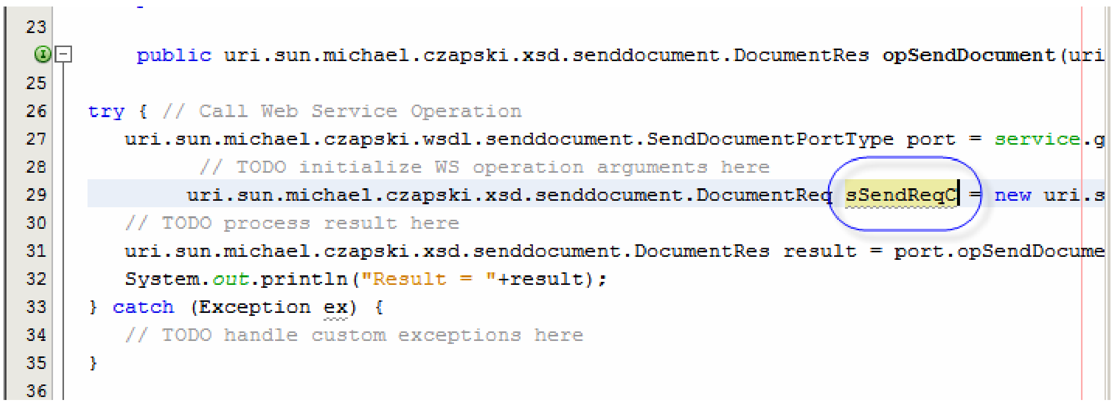
Expand the Web Service Reference node tree all the way to the operation. Drag the web service operation to the Java source canvas as illustrated in Figure 6-9.



```
/**
 *
 * @author mczapski
 */
@WebService(serviceName = "SendDocumentService", portName
@Stateless
public class WSSReceiveDocument implements SendDocumentPo

    public uri.sun.michael.czapski.xsd.senddocument.Docum

    }

}
```

Figure 6-9 Add web service invocation to the Java source

You will get a slab of Java boilerplate code added to the canvas. The service invocation will have an error. Merely rename the variable sSendReq to sSendReqC, or similar, to eliminate the problem. Right-click inside the source code window and choose Format.



```

    public uri.sun.michael.czapski.xsd.senddocument.DocumentRes opSendDocument(uri

try { // Call Web Service Operation
    uri.sun.michael.czapski.wsdl.senddocument.SendDocumentPortType port = service.g
        // TODO initialize WS operation arguments here
        uri.sun.michael.czapski.xsd.senddocument.DocumentReq sSendReqC = new uri.s
    // TODO process result here
    uri.sun.michael.czapski.xsd.senddocument.DocumentRes result = port.opSendDocume
    System.out.println("Result = "+result);
} catch (Exception ex) {
    // TODO handle custom exceptions here
}
```

Figure 6-10 Boilerplate code invoking the web service, with variable name changed

Modify the boilerplate code to read as shown in Listing 6-1.

Listing 6-1 body of the opSendDocument method, simplified

```
public uri.sun.michael.czapski.xsd.senddocument.DocumentRes
        opSendDocument
            (uri.sun.michael.czapski.xsd.senddocument.DocumentReq sSendReq) {

    try { // Call Web Service Operation
        uri.sun.michael.czapski.wsdl.senddocument.SendDocumentPortType port
                = service.getSendDocumentPort();
        return port.opSendDocument(sSendReq);
    } catch (Exception ex) {
        uri.sun.michael.czapski.xsd.senddocument.DocumentRes
            result = new uri.sun.michael.czapski.xsd.senddocument.DocumentRes();
        result.setDocID(sSendReq.getDocID());
        result.setSendStatus(false);
        return result;
    }
}
```

Build and deploy this service.

7. Test Wrapped Service

Expand the Web Services node, right-click SendDocumentService[SendDocumentPort] node and choose Create Web Services Tests. Figure 7-1 illustrates this.

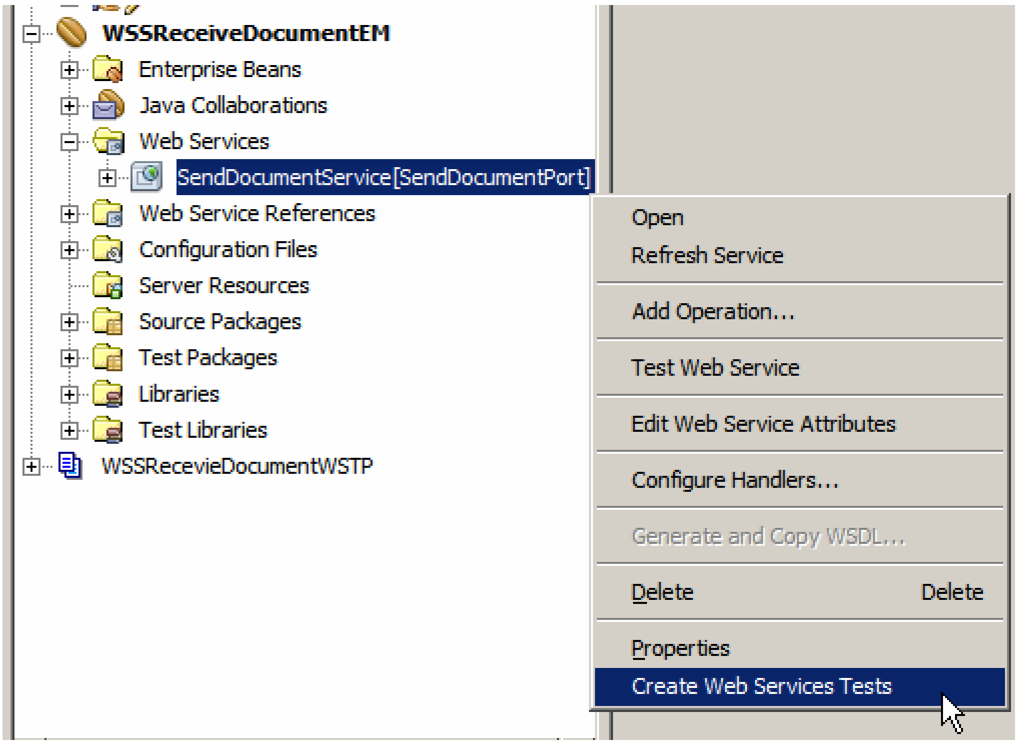
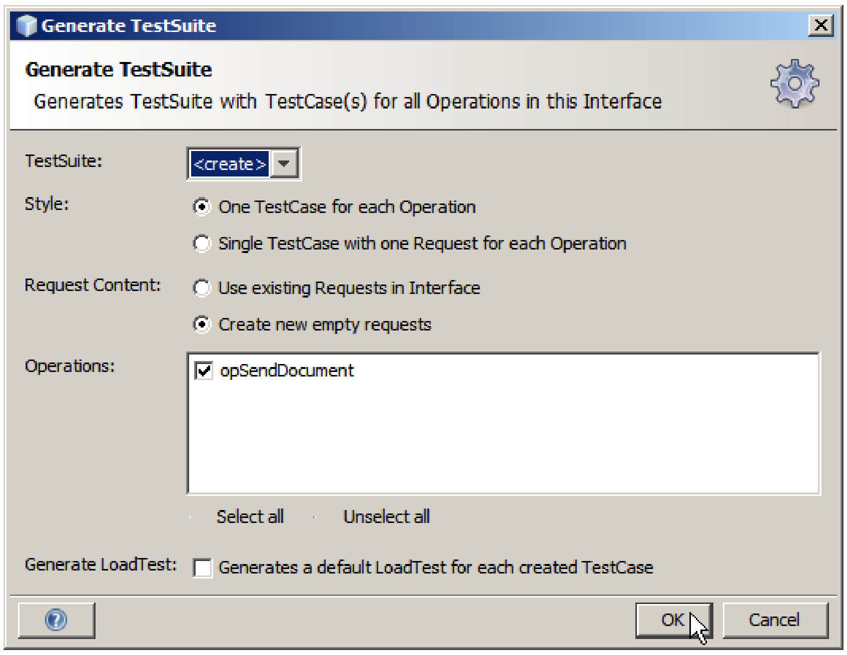


Figure 7-1 Create web services tests for the service

Accept the default options, as shown in Figure 7-2, and accept the default name.


Generate TestSuite

Generate TestSuite

Generates TestSuite with TestCase(s) for all Operations in this Interface

TestSuite: <create>

Style: [✓ One TestCase for each Operation] Single TestCase with one Request for each Operation

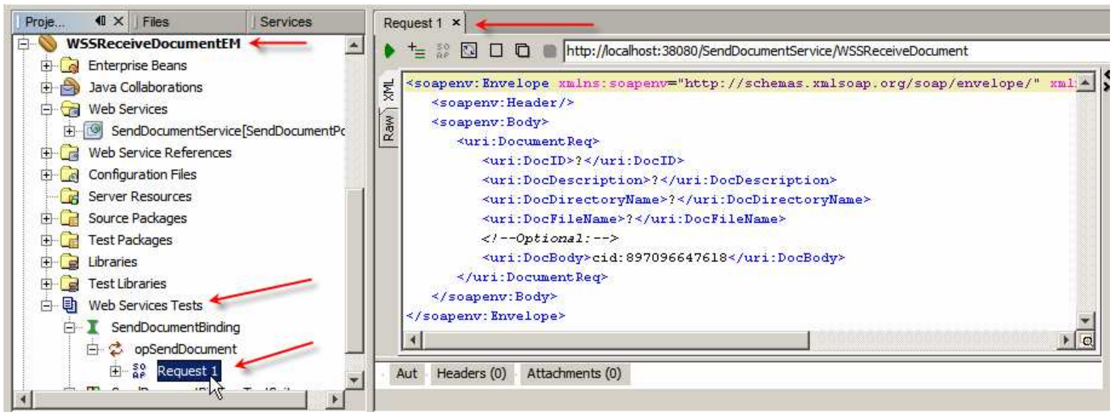
Request Content: Use existing Requests in Interface [✓ Create new empty requests]

Operations: [✓ opSendDocument]

Generate LoadTest: ☐ Generates a default LoadTest for each created TestCase

Figure 7-2 Accept defaults

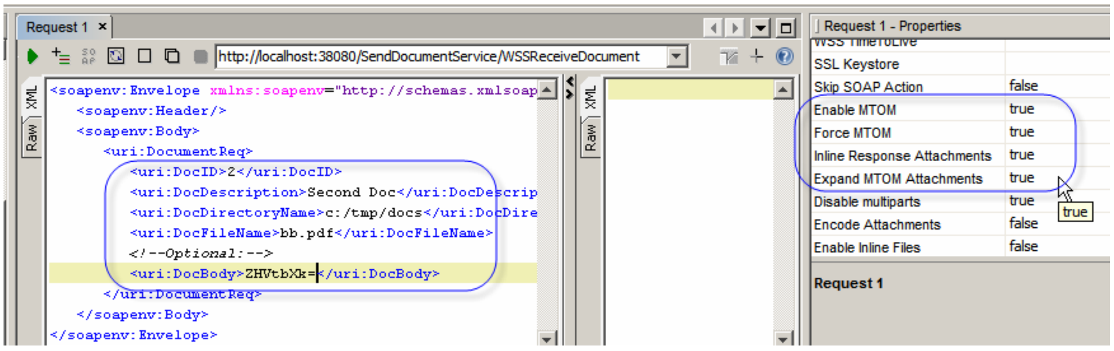
Expand the Web Services Tests node tree all the way to Request 1 and open Request 1 in the Request Editor, as shown in Figure 7-3.



```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xml:
   <soapenv:Header/>
   <soapenv:Body>
      <uri:DocumentReq>
         <uri:DocID>?</uri:DocID>
         <uri:DocDescription>?</uri:DocDescription>
         <uri:DocDirectoryName>?</uri:DocDirectoryName>
         <uri:DocFileName>?</uri:DocFileName>
         <!--Optional:-->
         <uri:DocBody>cid:897096647618</uri:DocBody>
      </uri:DocumentReq>
   </soapenv:Body>
</soapenv:Envelope>
```

Figure 7-3 Open Request 1 in Request Editor

Modify the request body by providing appropriate values for the document elements and modify Request 1 properties to enable MTOM support. Figure 7-4 illustrates both aspects.



Request 1
http://localhost:38080/SendDocumentService/WSSReceiveDocument
XML
Raw
Request 1 - Properties
Request 1

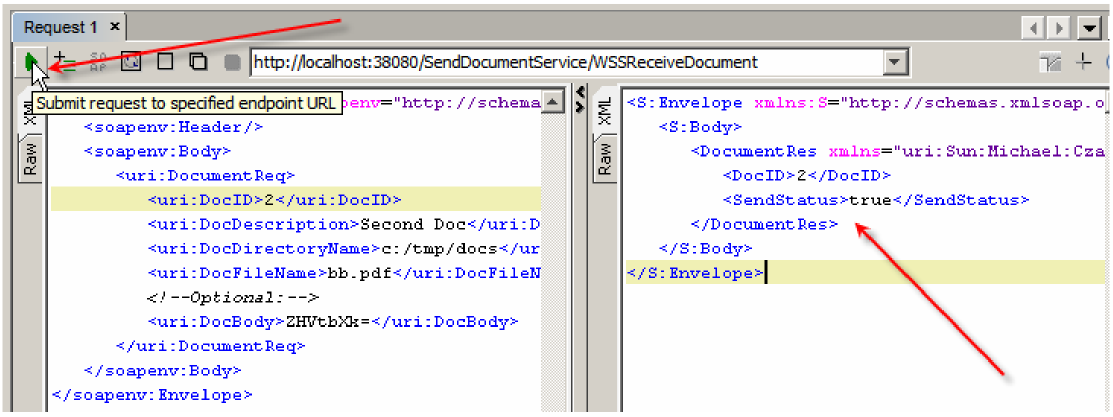
```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap
   <soapenv:Header/>
   <soapenv:Body>
      <uri:DocumentReq>
         <uri:DocID>2</uri:DocID>
         <uri:DocDescription>Second Doc</uri:DocDescrip
         <uri:DocDirectoryName>c:/tmp/docs</uri:DocDire
         <uri:DocFileName>bb.pdf</uri:DocFileName>
         <!--Optional:-->
         <uri:DocBody>ZHVtbXk=</uri:DocBody>
      </uri:DocumentReq>
   </soapenv:Body>
</soapenv:Envelope>
```

WSS TimeToLive	
SSL Keystore	
Skip SOAP Action	false
Enable MTOM	true
Force MTOM	true
Inline Response Attachments	true
Expand MTOM Attachments	true
Disable multiparts	true
Encode Attachments	false
Enable Inline Files	false

Figure 7-4 Modify request body and request properties

You should still have the NetBeans General Manual Proxy Settings configured to use the TCP Mon proxy at port 8888. Switch to TCP Mon and click the Remove All button to clear the old interactions. Switch back to NetBeans and submit the request.

You should receive a response that looks like that shown in Figure 7-5.



Request 1
http://localhost:38080/SendDocumentService/WSSReceiveDocument
Submit request to specified endpoint URL
XML
Raw

```
<soapenv:Envelope xmlns:soapenv="http://schema
   <soapenv:Header/>
   <soapenv:Body>
      <uri:DocumentReq>
         <uri:DocID>2</uri:DocID>
         <uri:DocDescription>Second Doc</uri:D
         <uri:DocDirectoryName>c:/tmp/docs</ur
         <uri:DocFileName>bb.pdf</uri:DocFileN
         <!--Optional:-->
         <uri:DocBody>ZHVtbXk=</uri:DocBody>
      </uri:DocumentReq>
   </soapenv:Body>
</soapenv:Envelope>
```

```
<S:Envelope xmlns:S="http://schemas.xmlsoap.o
   <S:Body>
      <DocumentRes xmlns="uri:Sun:Michael:Cza
         <DocID>2</DocID>
         <SendStatus>true</SendStatus>
      </DocumentRes>
   </S:Body>
</S:Envelope>
```

Figure 7-5 Request and Response

Switch to the TCP Mon and observe the request and the response as they were seen on the wire. Note that the body of the DocBody node has been externalised to a MME part and replaced with the ContentID reference. Note the correspondence between the ContentID in the reference and the contented of the MIME part. Figure 7-8 illustrates the request and the response, encoded as per the MTOM requirements. Note that all the MTOM-related work was performed by the "behind-the-scenes" infrastructure. All we needed to do was to enable MTOM support by checking the checkbox.

To find out what the URL of the MTOM Wrapper Service is right-click on the service name, choose Test Web Service, as shown in Figure 7-6, and copy the WSDL location as shown in Figure 7-7.

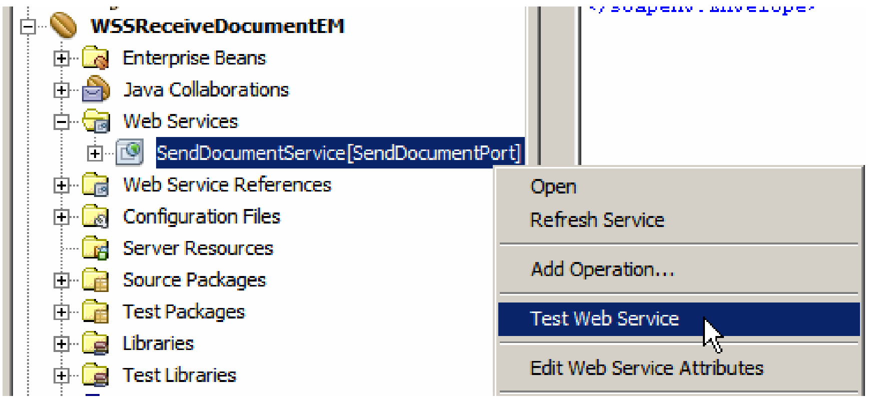
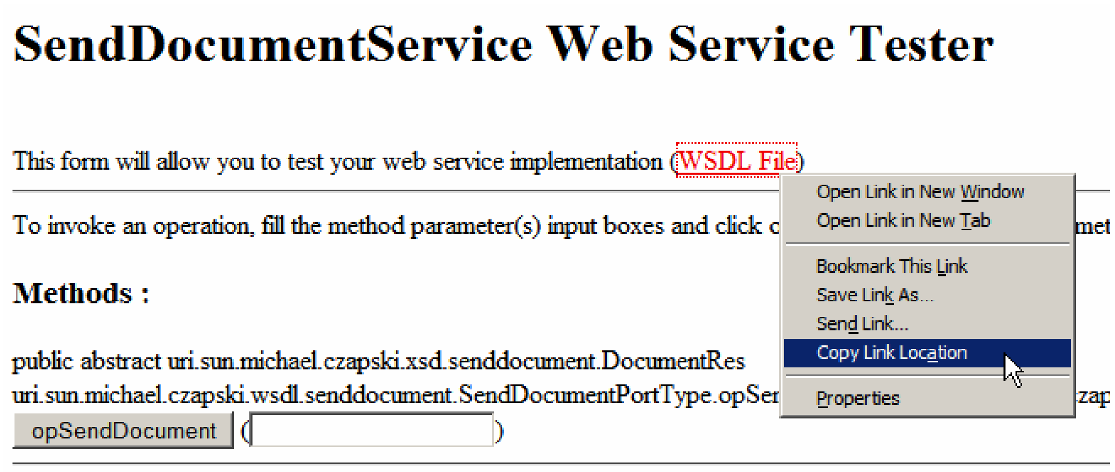


Figure 7-6 Start the Test Web Service functionality


SendDocumentService Web Service Tester

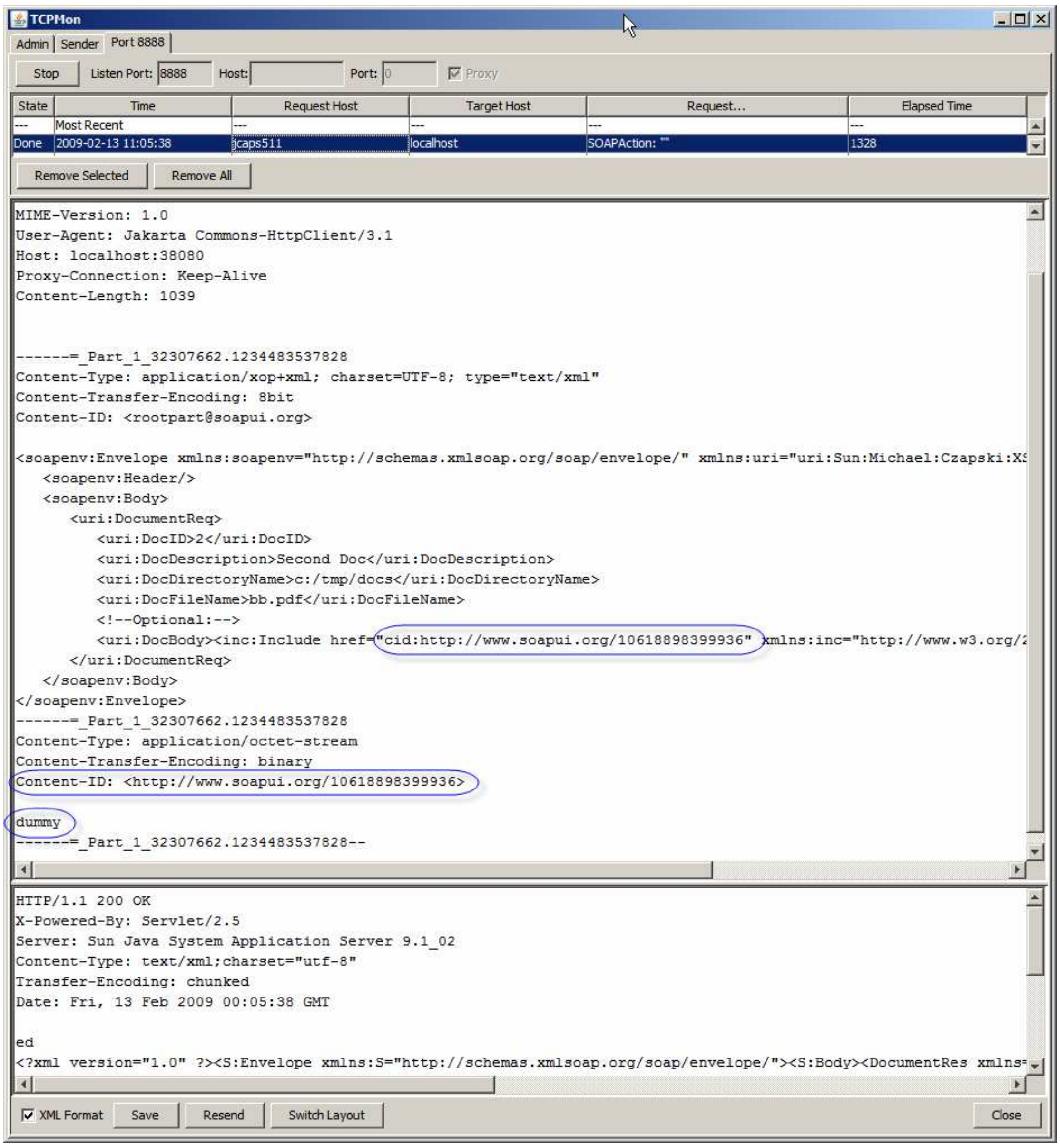
This form will allow you to test your web service implementation (WSDL File)

To invoke an operation, fill the method parameter(s) input boxes and click c met

Methods :

public abstract uri.sun.michael.czapski.xsd.senddocument.DocumentRes
uri.sun.michael.czapski.wsdl.senddocument.SendDocumentPortType.opSer zap

Figure 7-7 Copy WSDL Location



State	Time	Request Host	Target Host	Request...	Elapsed Time
---	Most Recent	---	---	---	---
Done	2009-02-13 11:05:38	jcaps511	localhost	SOAPAction: ""	1328

```
MIME-Version: 1.0
User-Agent: Jakarta Commons-HttpClient/3.1
Host: localhost:38080
Proxy-Connection: Keep-Alive
Content-Length: 1039


------=_Part_1_32307662.1234483537828
Content-Type: application/xop+xml; charset=UTF-8; type="text/xml"
Content-Transfer-Encoding: 8bit
Content-ID: <rootpart@soapui.org>

<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:uri="uri:Sun:Michael:Czapski:X
   <soapenv:Header/>
   <soapenv:Body>
      <uri:DocumentReq>
         <uri:DocID>2</uri:DocID>
         <uri:DocDescription>Second Doc</uri:DocDescription>
         <uri:DocDirectoryName>c:/tmp/docs</uri:DocDirectoryName>
         <uri:DocFileName>bb.pdf</uri:DocFileName>
         <!--Optional:-->
         <uri:DocBody><inc:Include href="cid:http://www.soapui.org/10618898399936" xmlns:inc="http://www.w3.org/2
      </uri:DocumentReq>
   </soapenv:Body>
</soapenv:Envelope>
------=_Part_1_32307662.1234483537828
Content-Type: application/octet-stream
Content-Transfer-Encoding: binary
Content-ID: <http://www.soapui.org/10618898399936>

dummy
------=_Part_1_32307662.1234483537828--
```

```
HTTP/1.1 200 OK
X-Powered-By: Servlet/2.5
Server: Sun Java System Application Server 9.1_02
Content-Type: text/xml;charset="utf-8"
Transfer-Encoding: chunked
Date: Fri, 13 Feb 2009 00:05:38 GMT

ed
<?xml version="1.0" ?><S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"><S:Body><DocumentRes xmlns=
```

Figure 7-8 MTOM-encoded request and response on the wire

The WSDL location for the MTOM Wrapper service looks, for me, like:

`http://localhost:38080/SendDocumentService/WSSReceiveDocument?WSDL`

The original Repository-based project’s WSDL location looks, for me, like:

`http://localhost:38080/WSSReceiveDocument/SendDocumentPort?WSDL`

Even though both services are built from the same WSDL the servlet contexts are different so there is no clash.

8. Create Java CAPS Environment

Create a Java CAPS Environment, WSEnv, as illustrated in Figures Figure 8-1 through Figure 8-3.

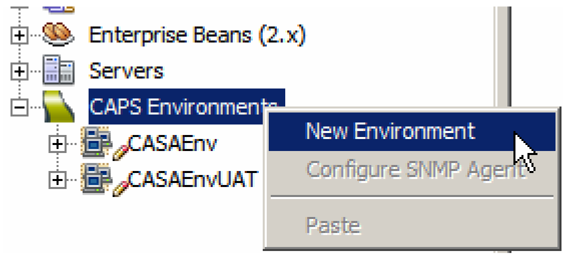


Figure 8-1 Create Java CAPS Environment

Add a Logical Host. Add a Sun Java System Application Server and a Sun Java System Message Queue. Set the properties for both to provide authentication credentials, host names and port numbers that reflect your environment.

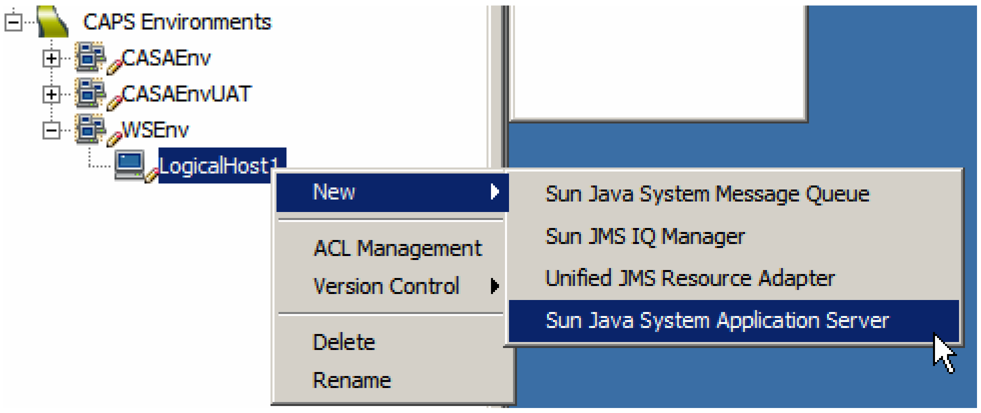
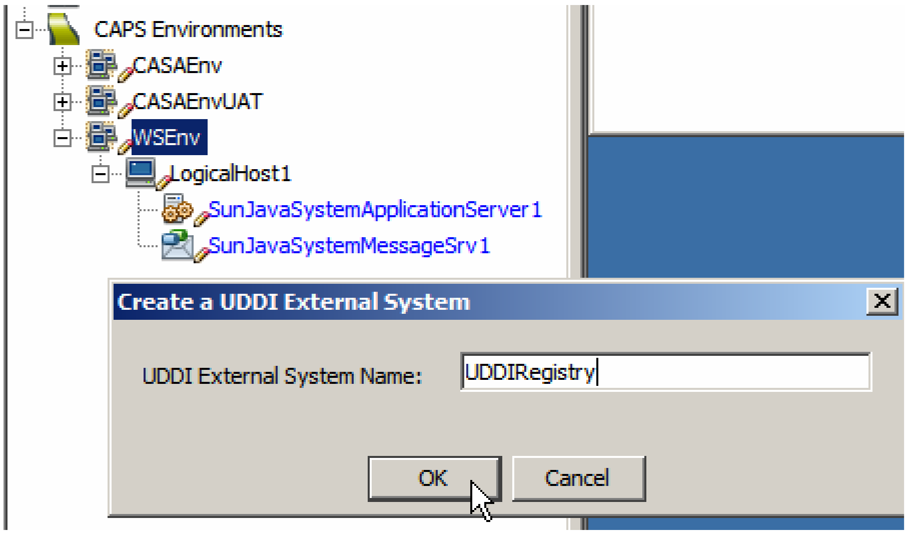


Figure 8-2 Add an Application Server to the Environment's Logical Host.

Add a new UDDI External System container and modify its properties to reflect your environment.



UDDI External System Name: UDDIRegistry

Figure 8-3 Add new UDDI External System

We will add and configure Web Services Client and Server External System containers as we go along in solution development.

9. Obtain and use the Apache TCP Mon

One of the issues with SOAP decoration, which is what MTOM and others do, is that the on-the-wire message looks different from what the sending and the receiving applications see. It is very hard to make the application server and the client log what they are sending and receiving and even then t here is a good chance that what is logged differs from what is sent / received. To see what is really exchanged a wire snooper of some sort is required.

Apache TCP Mon, see http://ws.apache.org/commons/tcpmon/index.html, can be used as a convenient proxy to view the on-the-wire messages exchanged between web services invokers and providers. Download the TCP Mon from the site. Tutorial at http://ws.apache.org/commons/tcpmon/tcpmontutorial.html has a nice explanation of the usage modes.

To start the TCP Mon from the command line in a direct intermediary mode with specific host and port configuration one could say (on Windows):

```
C:> cd C:\tools\tcpmon-1.0-bin\build
C:> tcpmon.bat 38081 localhost 38080
```

This will start the TCP Mon with the listening port 38081, relaying messages to port 38080 on localhost.

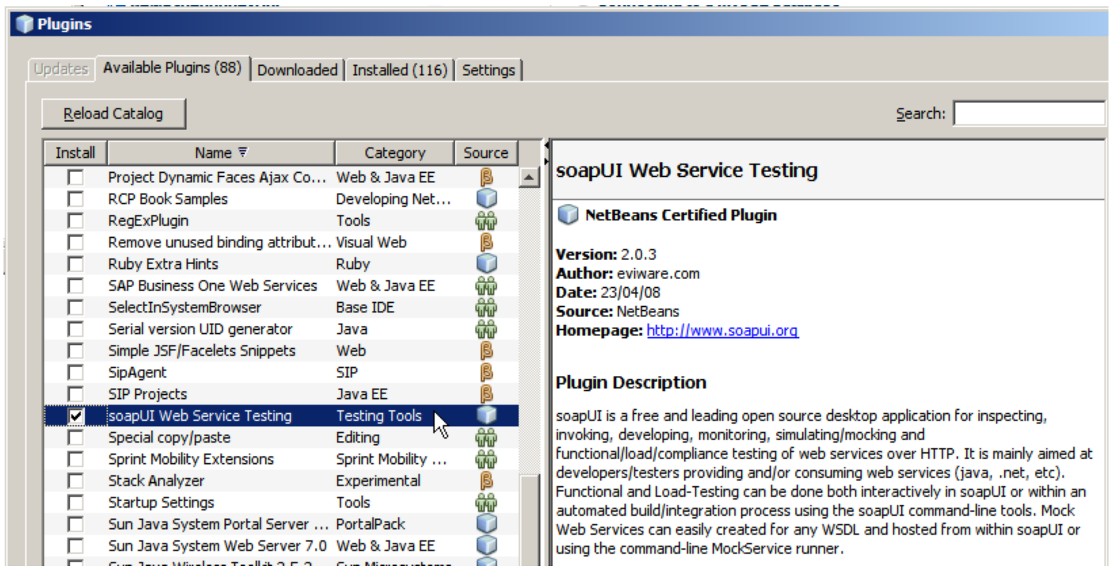
```
C:> cd C:\tools\tcpmon-1.0-bin\build
C:> tcpmon.bat 8888
```

This will start the TCP Mon as a proxy listening on port 8888. One needs to configure one’s client to use the proxy.

10. Install soapUI Plugin

To test EJB-based web services with complex messages SoapUI plugin needs be installed.

Tools->Plugins->Available Plugins



Install	Name	Category	Source
☐	Project Dynamic Faces Ajax Co…	Web & Java EE	β
☐	RCP Book Samples	Developing Net…	⬡
☐	RegExPlugin	Tools	👥
☐	Remove unused binding attribut…	Visual Web	β
☐	Ruby Extra Hints	Ruby	⬡
☐	SAP Business One Web Services	Web & Java EE	👥
☐	SelectInSystemBrowser	Base IDE	👥
☐	Serial version UID generator	Java	👥
☐	Simple JSF/Facelets Snippets	Web	β
☐	SipAgent	SIP	β
☐	SIP Projects	Java EE	β
☑	soapUI Web Service Testing	Testing Tools	⬡
☐	Special copy/paste	Editing	👥
☐	Sprint Mobility Extensions	Sprint Mobility …	👥
☐	Stack Analyzer	Experimental	β
☐	Startup Settings	Tools	👥
☐	Sun Java System Portal Server …	PortalPack	⬡
☐	Sun Java System Web Server 7.0	Web & Java EE	⬡

soapUI Web Service Testing

NetBeans Certified Plugin

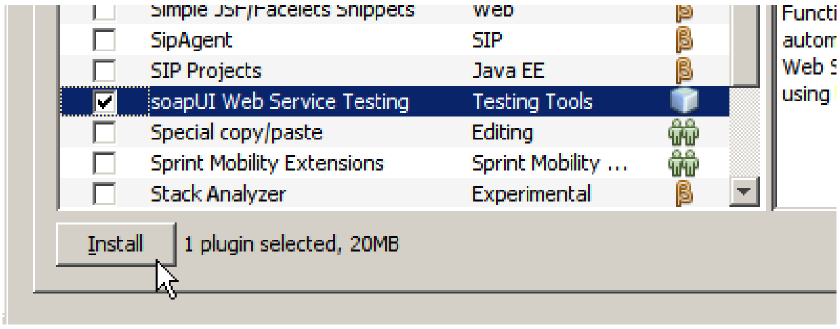
Version: 2.0.3
Author: eviware.com
Date: 23/04/08
Source: NetBeans
Homepage: http://www.soapui.org

Plugin Description

soapUI is a free and leading open source desktop application for inspecting, invoking, developing, monitoring, simulating/mocking and functional/load/compliance testing of web services over HTTP. It is mainly aimed at developers/testers providing and/or consuming web services (java, .net, etc). Functional and Load-Testing can be done both interactively in soapUI or within an automated build/integration process using the soapUI command-line tools. Mock Web Services can easily created for any WSDL and hosted from within soapUI or using the command-line MockService runner.

Figure 10-1 Locate soapUI in the list of AvailablePlugins

Click the checkbox next to plugin name and click Install. Follow the prompts.



☐	Simple JSF/Facelets Snippets	Web	β
☐	SipAgent	SIP	β
☐	SIP Projects	Java EE	β
☑	soapUI Web Service Testing	Testing Tools	⬡
☐	Special copy/paste	Editing	👥
☐	Sprint Mobility Extensions	Sprint Mobility …	👥
☐	Stack Analyzer	Experimental	β

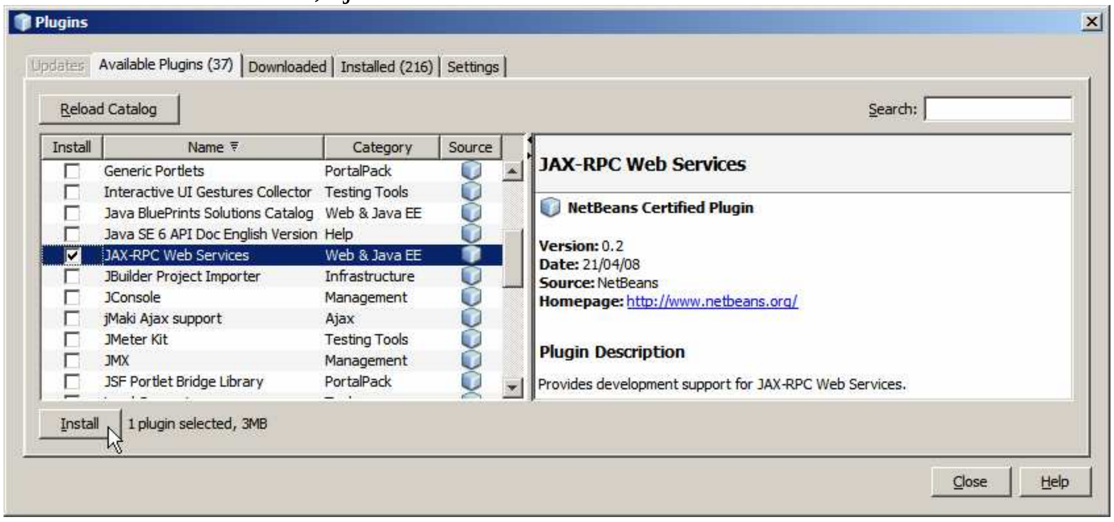
Functi
autom
Web S
using

Figure 10-2 Choosing to install the plugin

You will need to have access to the Internet as the tooling will try to download the plugin.

11. Install JAX-RPC Plugin

Java CAPS Repository Web Services ae JAX-RPC services. To integrate them with non-Repository-based services we need to install JAX-RPC Plugin. Do this through the Tools -> Plugins Plugin Manager's Available Plugins Tab, where the plugin is listed. Locate JAX-RPC, cject the Install checkbox and click Install.



Install	Name	Category
☐	Generic Portlets	PortalPack
☐	Interactive UI Gestures Collector	Testing Tools
☐	Java BluePrints Solutions Catalog	Web & Java EE
☐	Java SE 6 API Doc English Version	Help
✓	JAX-RPC Web Services	Web & Java EE
☐	JBuilder Project Importer	Infrastructure
☐	JConsole	Management
☐	jMaki Ajax support	Ajax
☐	JMeter Kit	Testing Tools
☐	JMX	Management
☐	JSF Portlet Bridge Library	PortalPack

JAX-RPC Web Services

NetBeans Certified Plugin

Version: 0.2
Date: 21/04/08
Source: NetBeans
Homepage: http://www.netbeans.org/

Plugin Description

Provides development support for JAX-RPC Web Services.

1 plugin selected, 3MB

Figure 11-1 Choose to install JAX-RPC plugin