# Oracle® Java CAPS FTP Binding Component Tutorial

Part No: 821–2625
December 2011

120126@25097

# Contents

# Understanding the FTP Binding Component

In this tutorial, you will create a simple BPEL Module project called SendInventory and a Composite Application project called SendInventoryCompAppl. Here, you create two WSDL Documents and design a BPEL Process, and then deploy the project. You will learn the method to transfer the file from the local directory to the FTP Server using the FTP and File Binding Component.

**What You Need to Know**

These topics provide information you need to know before beginning the tutorial.

- "Tutorial Overview" on page 6.
- "Tutorial Requirement" on page 7.
- "Tutorial Plan" on page 7.
- "FTP Binding Component Project in a Nutshell" on page 8.
- "Starting the GlassFish Application Server" on page 9.
- "Working With JBI Runtime Environment" on page 11.
- "FTP Binding Component Runtime Configuration Properties" on page 13.
- "Creating a BPEL Module Project : SendInventory" on page 16.
- "Creating a WSDL Document : Using FTP" on page 19.
- "FTP Binding Component Extensibility Elements" on page 28.
- "Runtime Configuration" on page 29.
- "Creating a WSDL Document : Using FILE" on page 43.
- "Creating a BPEL Process" on page 47.
- "Validating BPEL" on page 62.
- "Design View : Notifications" on page 62.
- "Creating a Composite Application" on page 64.
- "Working With Various Binding Types" on page 72.
- "Exploring the XML Schema" on page 80.
- "Creating the XML Schema" on page 82.

**What You Need to Do**

These topics provide instructions on the following.

- "To Start the GlassFish Application Server" on page 10.
- "To Start JBI Components" on page 12.
- "To Create a BPEL Module Project" on page 16.
- "To Create a WSDL Document : ftpTransfer" on page 19.
- "To Modify ftp:message Properties" on page 24.
- "To Create a WSDL Document : fileTrigger" on page 44.
- "To Create a BPEL Process" on page 47.
- "To Add a Partner Link" on page 49.
- "To Add Web Services and Basic Activities" on page 50.
- "To Edit Web Service : Receive1" on page 53.
- "To Edit the Web Service : Invoke1" on page 56.
- "To Edit the Basic Activities : Assign1" on page 60.
- "To Invoke Explicit Validation" on page 62.
- "To Create a Composite Application" on page 64.
- "To Deploy the Composite Application" on page 69.
- "To Create XML Schema" on page 82.
- "To Add a Complex and a Global Complex Type to the XML Schema" on page 84.
- "To Add Element to the XML Schema" on page 85.
- "To Add Elements to the XML Schema" on page 90.

## Tutorial Overview

The FTP Binding Component (referred as FTP BC) is a binding component implementation in compliance with JBI Specification 1.0. The FTP BC uses the FTP protocol to transport messages. This helps define the services using WSDL and bind them to FTP as the underlying message transport protocol. For example, in a JBI environment, the Services Engine (SE) can orchestrate the services consumption and provision.

The FTP BC implements all required using JBI specification so that it can be deployed and executed in any JBI compliant target environment. This implementation also uses a NetBeans module as the design-time component to facilitate the process of service definition and binding. The NetBeans module makes WSDL authoring and FTP binding convenient in the NetBeans IDE.

This tutorial illustrates the following aspects of the FTP Binding Component.

1. Create an SOA project.
2. Add WSDL documents to your project.
3. Use the Partner view of the WSDL editor to add the messages, partner link type, port type, and operation.
4. Create a Composite Application project.
5. Use the Composite Application (Service Assembly) editor to modify the project configuration.

## Tutorial Requirement

Read the tutorial thoroughly before installing and executing the Binding Component.

### Software Needed for the Tutorial

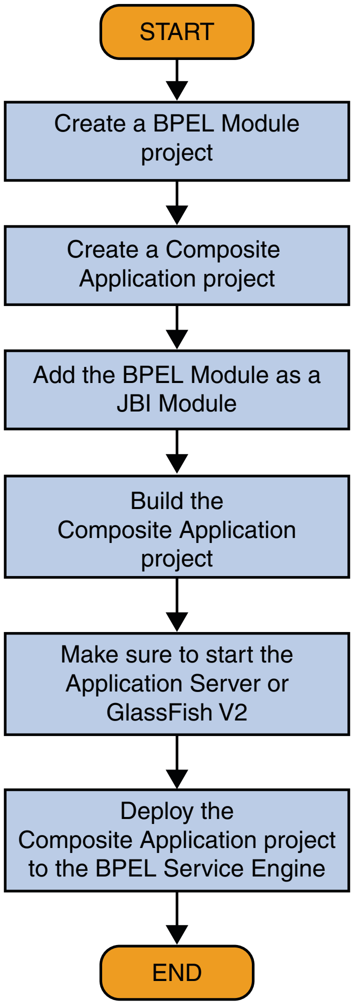
FTP BC assumes that the following are configured on the system:

- Java CAPS and all required software.
- A JCA container, tested with GlassFish Installer.

## Tutorial Plan

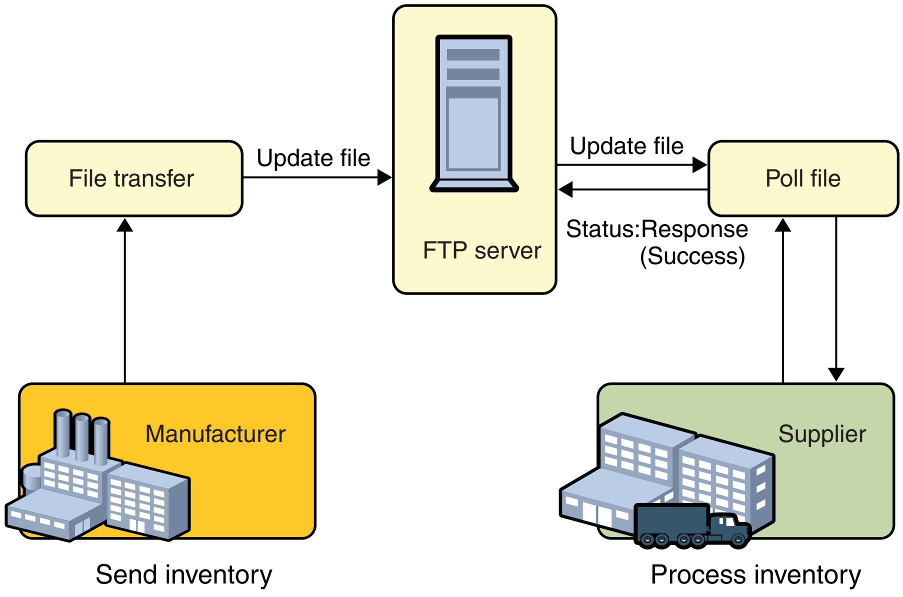
The following steps provide an overview of the tutorial procedures:

1. Start the GlassFish Application Server.
2. Start the JBI Components.
3. Create a BPEL Module project using the New Project wizard.
4. Create the following WSDL Documents for the BPEL Project.
    a. File : WSDL Document
    b. FTP : WSDL Document
5. Create a Composite Application project.
6. Add the BPEL Module project as a JBI Module to the Composite Application project.
7. Build the Composite Application project.
8. Deploy the Composite Application project to the Application Server.
9. Check the Output.


START
Create a BPEL Module project
Create a Composite Application project
Add the BPEL Module as a JBI Module
Build the Composite Application project
Make sure to start the Application Server or GlassFish V2
Deploy the Composite Application project to the BPEL Service Engine
END

## FTP Binding Component Project in a Nutshell

The illustration explains the process of creating a project using FTP Binding Component.



The FTP server is used as a communication hub for a manufacturing supply chain. The Manufacturer sends an XML file with inventory levels of multiple products (using a Business Process that reads a local file with File-BC and places it in the FTP server using FTP-BC). Supplier picks up the XML file and updates inventory levels (using a Business Process that retrieves the file from the FTP server using FTP-BC). The Manufacturer Business Process uses simulated data created inline and the supplier prints inventory levels on ftpout.

## Starting the GlassFish Application Server

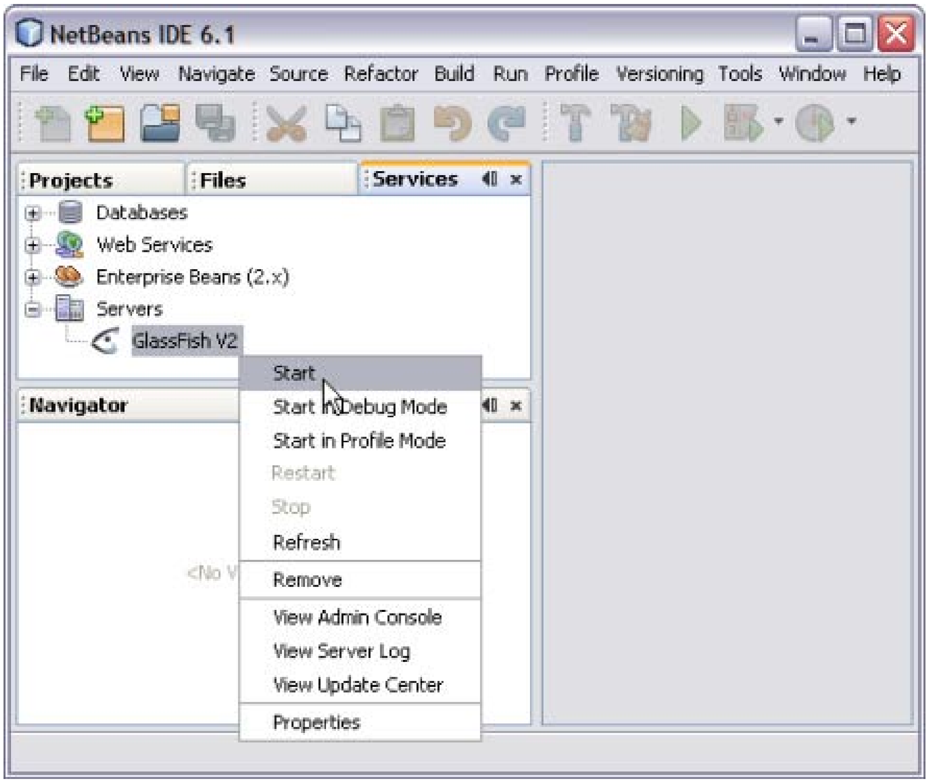
Follow the outlined procedure to start GlassFish Application Server.

## ▼ To Start the GlassFish Application Server

**1 In the NetBeans IDE, click the Services tab and expand the Servers node.**

The Servers node contains a GlassFish subnode.

**2 Right-click the GlassFish node. Select Start.**



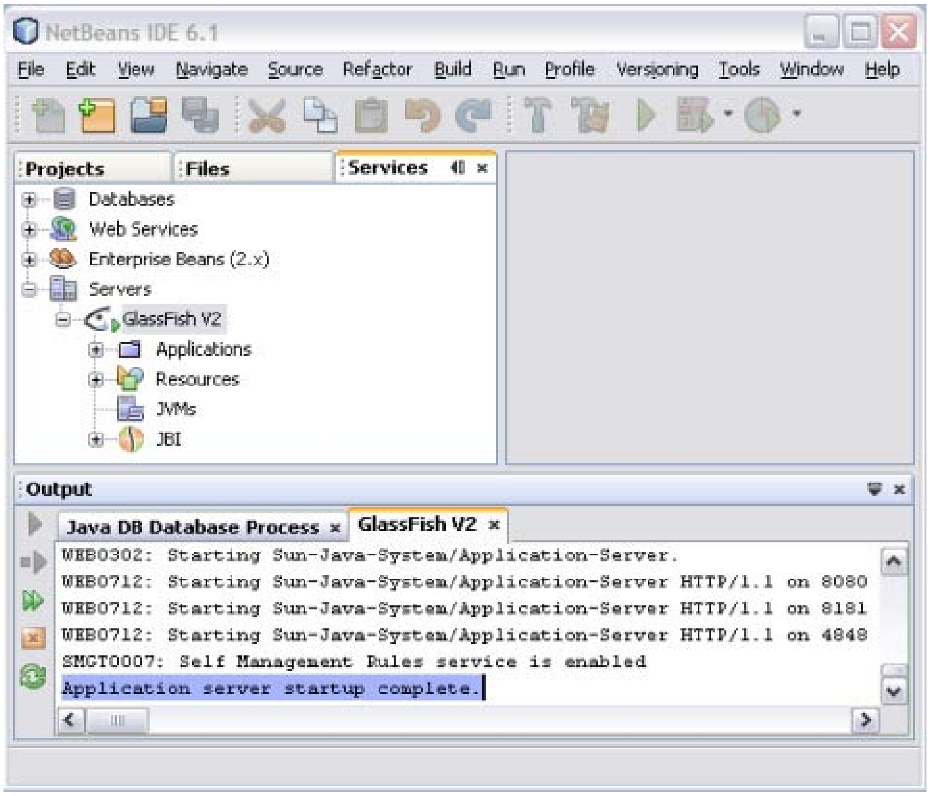
The Output window displays a log generated during application start up. The following message in the Output console indicates that the application server is listening.

```
Application server startup complete
```

---

**Note –** A green arrow badge on the GlassFish Application Server node indicates that the server is listening.

---



```
WEB0302: Starting Sun-Java-System/Application-Server.
WEB0712: Starting Sun-Java-System/Application-Server HTTP/1.1 on 8080
WEB0712: Starting Sun-Java-System/Application-Server HTTP/1.1 on 8181
WEB0712: Starting Sun-Java-System/Application-Server HTTP/1.1 on 4848
SMGT0007: Self Management Rules service is enabled
Application server startup complete.
```

---

**Tip –** Deploying an application to the GlassFish Application Server starts GlassFish automatically. Thus, you do not have to manually start the application server.

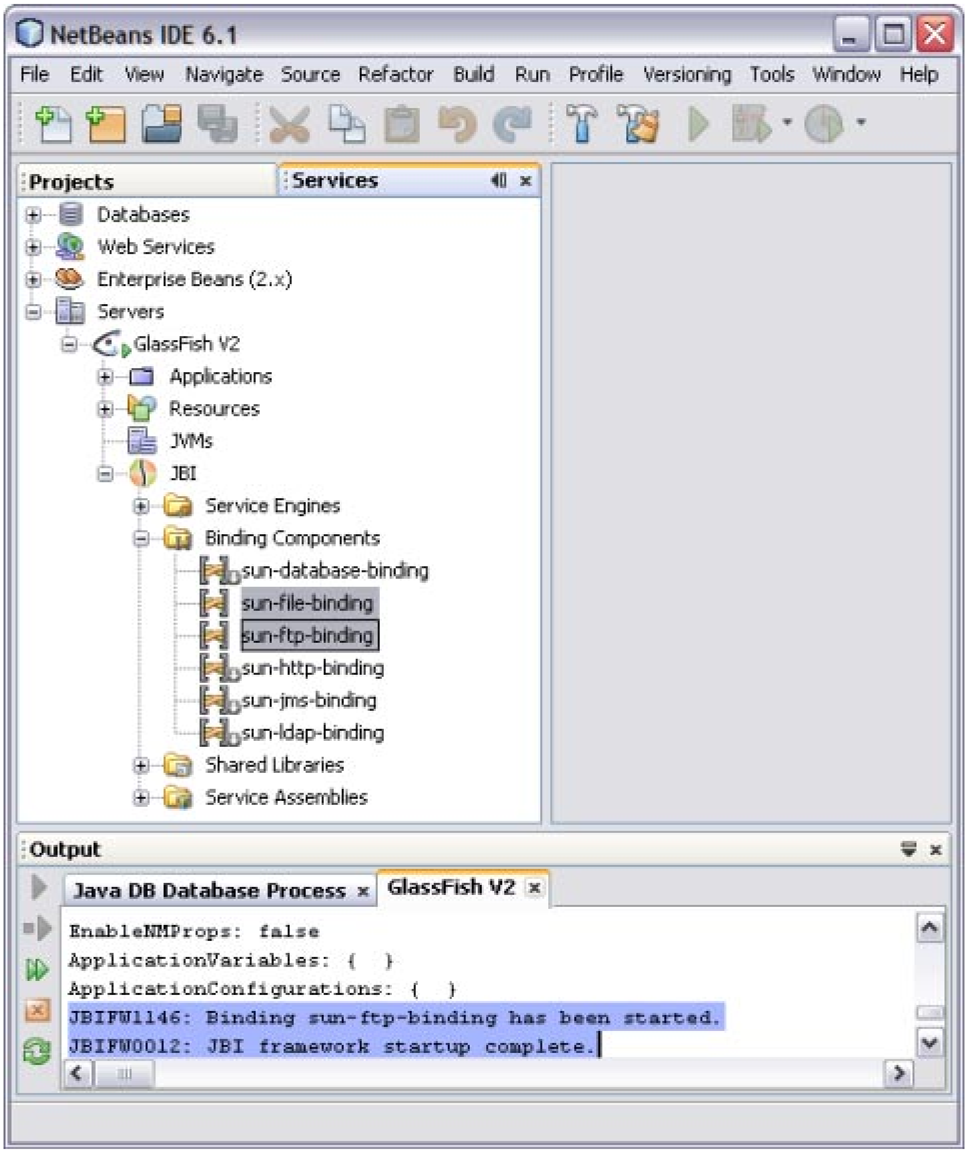
---

## Working With JBI Runtime Environment

The Java Business Integration (JBI) runtime environment provides the runtime capability for SOA tools in the IDE. The JBI runtime environment includes several components that interact using a services model. This model is based on Web Services Description Language (WSDL) 2.0. Components that supply or consume services within the JBI environment are referred to as Service Engines. One of the components is the BPEL Service Engine that provides services for executing business processes. Components that provide access to services that are external to the JBI environment are called Binding Components (BC).

## ▼ To Start JBI Components

1. **Click the Services tab. Expand the GlassFish node. Expand the JBI node.**
2. **Right-click sun-file-binding. Click Start.**
3. **Right-click sun-ftp-binding. Click Start.**

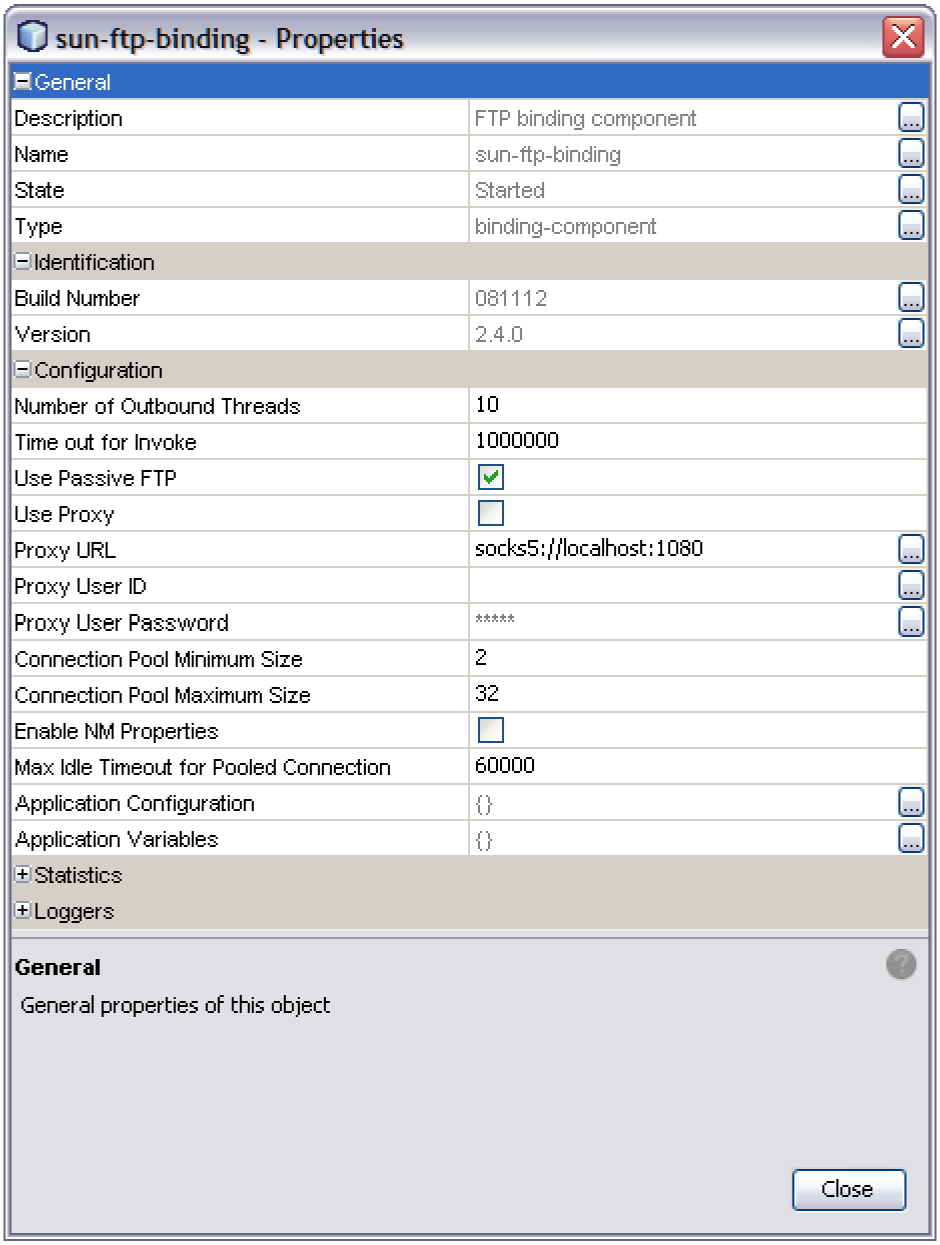


```
EnableNMProps: false
ApplicationVariables: {  }
ApplicationConfigurations: {  }
JBIFW1146: Binding sun-ftp-binding has been started.
JBIFW0012: JBI framework startup complete.
```

The following message in the Console window indicates that the FTP Binding Component has been invoked.

```
Successfully started FTP BC
```

## FTP Binding Component Runtime Configuration Properties



sun-ftp-binding - Properties

| | |
|---|---|
| **General** | |
| Description | FTP binding component |
| Name | sun-ftp-binding |
| State | Started |
| Type | binding-component |
| **Identification** | |
| Build Number | 081112 |
| Version | 2.4.0 |
| **Configuration** | |
| Number of Outbound Threads | 10 |
| Time out for Invoke | 1000000 |
| Use Passive FTP | ✓ |
| Use Proxy | ☐ |
| Proxy URL | socks5://localhost:1080 |
| Proxy User ID | |
| Proxy User Password | ***** |
| Connection Pool Minimum Size | 2 |
| Connection Pool Maximum Size | 32 |
| Enable NM Properties | ☐ |
| Max Idle Timeout for Pooled Connection | 60000 |
| Application Configuration | {} |
| Application Variables | {} |
| **Statistics** | |
| **Loggers** | |

**General**

General properties of this object

The **sun-ftp-binding** JBI Binding Component Runtime Configuration properties are described in the following table.

**TABLE 1** sun-ftp-binding Properties

| Property | Description |
|---|---|
| **General** | |
| Description | Description of the JBI Component. |
| Name | Name of the JBI Component. |
| State | State of the JBI Component. |
| Type | Type of the JBI Component. |
| **Identification** | |
| Build Number | JBI Component build number. |
| Version | JBI component version. |
| **Configuration** | |
| Number of Outbound Threads | The number of outbound threads waiting for messages from the JBI runtime environment. Any integer number between 1 and 2147483647 is allowed.<br><br>Default value: **10** |
| Time out for Invoke | The time out period, in milliseconds, for invoking a synchronous service.<br>■ Minimum – 100 milliseconds<br>■ Maximum – 2145483647 milliseconds<br><br>After a consumer sends a request, the proxy polls the response periodically. If there is no response after the `InvokeTimeout` period has elapsed, the poller thread exits and reports a timeout. The polling interval is specified by the extensible WSDL property `ftp:pollIntervalMillis`.<br><br>The default value in milliseconds is **1,000,000**. |
| Use Passive FTP | Indicates if passive FTP is enabled. |
| Use Proxy | Specifies whether to use a proxy for FTP transfer from within a corporate firewall.<br><br>Default value: **false** |
| Proxy URL | If `UseProxy` is enabled, specifies the URL for the proxy.<br><br>Default value: **socks5://localhost:1080** |
| Proxy User ID | A valid proxy user ID. |
| Proxy User Password | A valid proxy user password. |
| Connection Pool Minimum Size | Minimum number of connections in the pool. |
| Connection Pool Maximum Size | Maximum number of connections in the pool. |

**TABLE 1** sun-ftp-binding Properties *(Continued)*

| Property | Description |
|---|---|
| Enable NM Properties | Indicates if component–specific normalized message properties are enabled. |
| Max Idle Timeout for Pooled Connections | Maximum idle time period for connections in the pool in milliseconds (> 1000). |
| Application Configuration | Application Configuration allows users to define the named group of the configuration parameters specific to a JBI component. It helps an endpoint to reference these parameters and complete the binding.<br><br>Click the ellipses to add an Application Configuration. |
| Application Variables | Application variables are categorized into the following types:<br>■ STRING<br>■ BOOLEAN<br>■ NUMBER<br>■ PASSWORD<br><br>Click the ellipsis to add the Application Variables. |

## Creating a BPEL Module Project : SendInventory

This example demonstrates creating a BPEL Module named SendInventory.

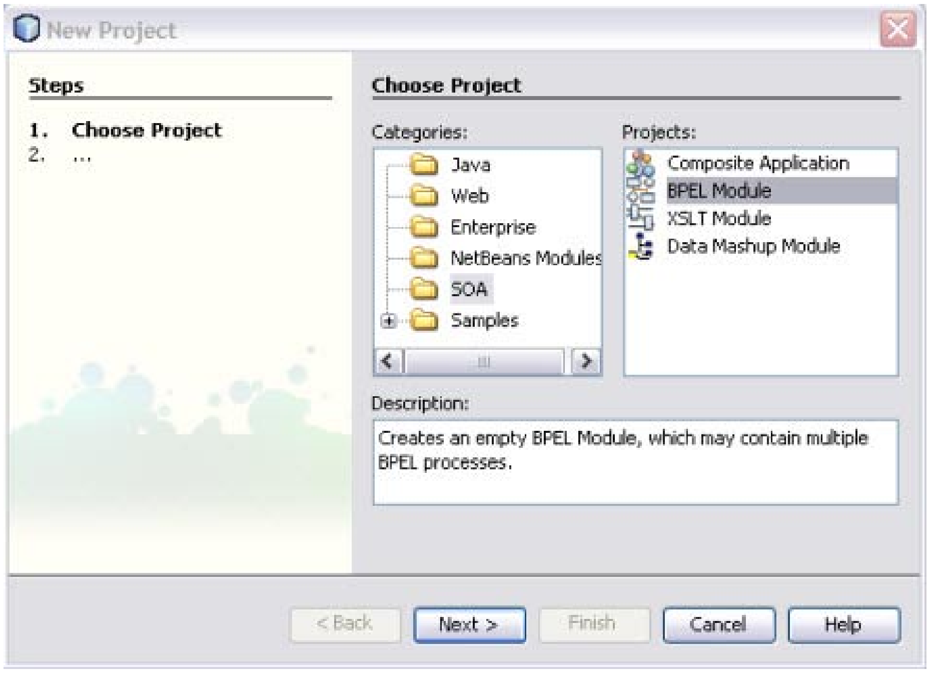
### ▼ To Create a BPEL Module Project

1. **Choose File —> New Project from the main menu.**

   The New Project wizard appears.

2. **Select the SOA node from the Categories list.**

3 **Select the BPEL Module node from the Projects list.**



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

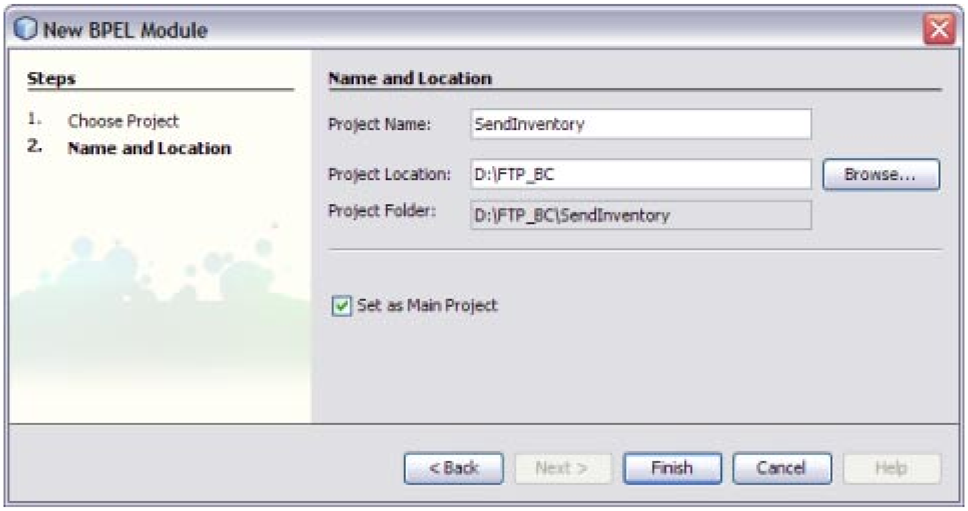
4 **Click Next.**

5 **Type the Project Name in the Project Name field.**

For this tutorial, enter **`SendInventory`**.

6 **Click Browse to navigate to the project location field.**

The IDE stores the project files. This step is optional.

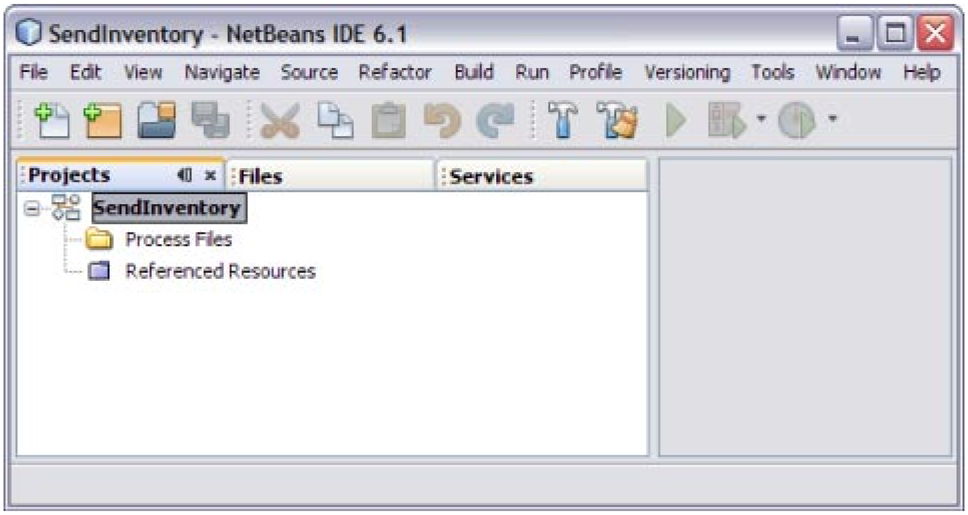


Project Name: SendInventory
Project Location: D:\FTP_BC
Project Folder: D:\FTP_BC\SendInventory
✓ Set as Main Project

7 **Click Finish.**

The Projects window now contains a project node for the SendInventory BPEL Module project.

8 **Click Save All.**



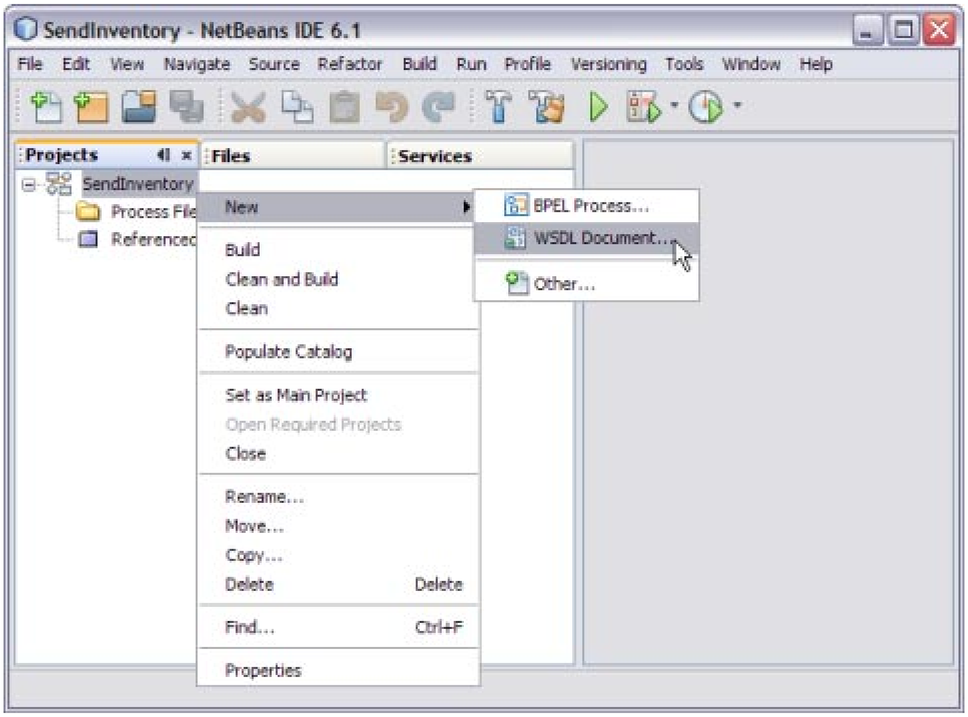
## Creating a WSDL Document : Using FTP

In this section, you will add a WSDL Document. In this example, you will add `ftpTransfer.wsdl` to the BPEL Module. Use the Partner view of the WSDL editor to configure the components of the WSDL document. A WSDL document defines a web service and is bound to the FTP transportation.

### ▼ To Create a WSDL Document : ftpTransfer

1 **Expand the SendInventory BPEL Module project node in the Projects tab.**

2 **Right-click the project node or Process Files node. Point to New and click WSDL Document.**



The New WSDL Document wizard appears.

3 **In the File Name field, enter `ftpTransfer.wsdl`.**

4 **Select Concrete WSDL Document.**

---

**Note –** When creating or editing a WSDL file, you may be prompted to select a binding type and a binding subtype. A binding contains protocol and data format information for the operations and messages of a port type. If you select the FTP binding type, then you must select one of the following binding subtypes. Both subtypes provide basic FTP functionality, such as a directory from which to read or write files, filename pattern matching, and message correlation.

- **FTP Message**. Select this binding subtype if you plan to rely on default values.
- **FTP Transport**. Select this binding subtype if you want to add customization such as specifying custom directories on the FTP server and specifying pre-transfer and post-transfer operations.

---

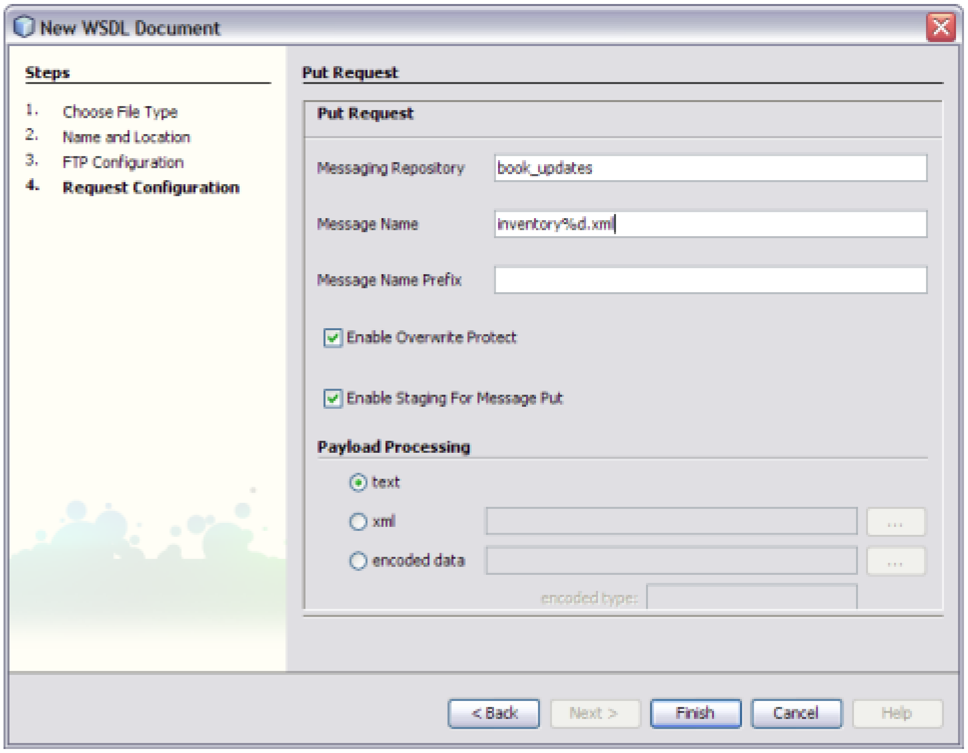
5 **In the Binding field, select FTP.**

6 **Choose any one of the following Types from the drop-down list.**

- **Poll Request Message**: Choose this scenario when the FTP BC polls for request messages from a dedicated subdirectory (inbox). In this example, the subdirectory is under the remote FTP directory (message repository) and invokes a JBI service with the message.
- **Poll Request Message and Put Response**: Choose this scenario when the FTP BC polls for request messages from a dedicated subdirectory (inbox). In this example, the subdirectory is under the remote FTP directory (messaging repository) and invokes a JBI service. It puts the responses back to a dedicated subdirectory (outbox) under a remote FTP directory (messaging repository).
- **Put Request Message**: Choose this scenario when a JBI service invokes FTP BC to put a request message to a dedicated subdirectory (outbox). In this example, the subdirectory is under the remote FTP directory (messaging repository).
- **Put Request Message and Poll Response**: Choose this scenario when a JBI service invokes FTP BC to put request messages to a dedicated subdirectory (inbox). In this example, the subdirectory is under the remote FTP directory (messaging repository). It polls the responses back from a dedicated subdirectory (outbox) under a remote FTP directory (messaging repository).
- **On Demand Get Message**: Outbound Get Messaging.
- **Receive Request**: Choose this scenario when the FTP BC polls (receiving) for request messages from a remote FTP target (source) and invokes a JBI service with the messages. This option is more customized than the Poll Request Message option.
- **Receive Request and Send Response**: Choose this scenario when the FTP BC polls (receiving) for a request messages from a remote FTP target (source). The FTP target invokes a JBI service and puts (sending) the responses back to another remote FTP target (destination). This option is more customized than the Poll Request Message and Put Response option.

- **Send Request**: Choose this scenario when a JBI service invokes FTP BC to put (sending) a message to a remote FTP target (destination). This option is more customized than the Poll Request Message option.
- **Send Request and Receive Response**: Choose this scenario when a JBI service invokes FTP BC to put (sending) request messages to a remote FTP target (destination). It polls (receives) the responses back from another remote FTP target (source). This option is more customized than the Put Request Message and Poll Response option.
- **On Demand Receive Transfer**: Outbound Receiving Transferring.

**7 Select Type — Put Request Message from the drop-down list.**



Put Request

Messaging Repository: book_updates

Message Name: inventory%d.xml

Message Name Prefix:

[✓] Enable Overwrite Protect

[✓] Enable Staging For Message Put

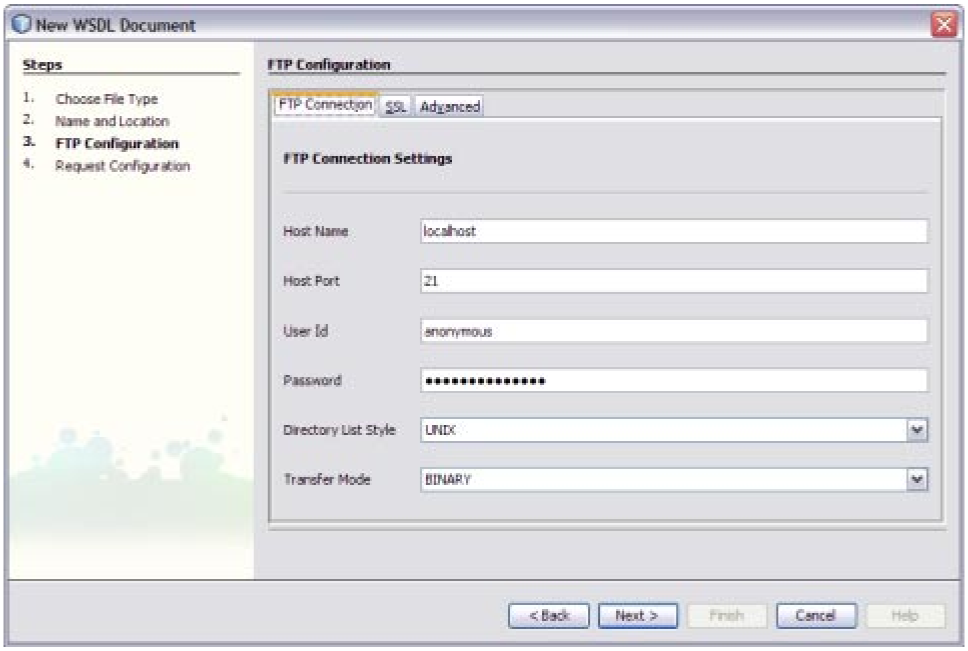
Payload Processing

[● text] ○ xml ○ encoded data

encoded type:

8 **Click Next.**

The New WSDL Document — FTP Configuration wizard appears.



Host Name: localhost
Host Port: 21
User Id: anonymous
Password: •••••••••••••••
Directory List Style: UNIX
Transfer Mode: BINARY

9 **Click the FTP Connection tab.**

10 **Enter the following:**

a. **Host Name: localhost**

b. **Host Port: 21**

c. **User Id: anonymous**

d. **Password: Enter Password**

e. **Directory List Style: UNIX**

f. **Transfer Mode: BINARY**

g. **Message Repository: Enter text in the Message Repository text area. This is optional.**

---

**Note –** The option Correlate Request Response is selected by default.

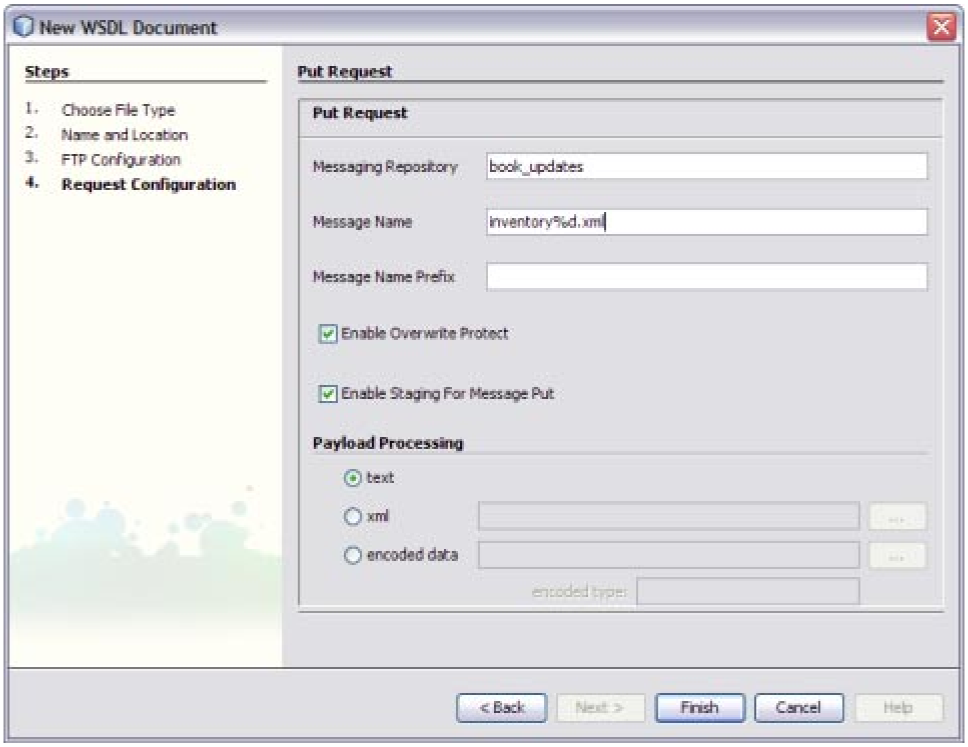
---

**11 Click Next.**

The New WSDL Document — Poll Request wizard appears.

**12 Enter the following fields:**

- **Message Repository**: book_updates
- **Message Name**: inventory%d.xml
- **Message Name Prefix**: This is optional.
- **Enable Overwrite Protect**: This option is selected, by default.
- **Enable Staging When Put Message**: This option is selected, by default.
- **Payload Processing**: The radio button xml is selected by default.



Messaging Repository: book_updates

Message Name: inventory%d.xml

Message Name Prefix:

[✓] Enable Overwrite Protect

[✓] Enable Staging For Message Put

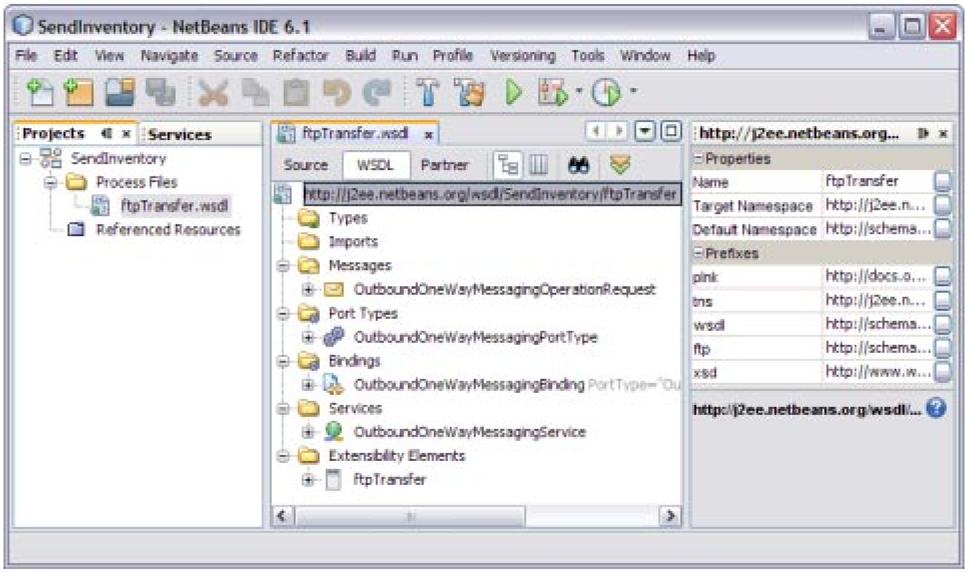
Payload Processing

[✓ text] xml encoded data

encoded type:

**13 Click Finish.**

This action opens the Project tree structure. In the current example, this action displays the WSDL editor for ftpTransfer along with its properties.


SendInventory - NetBeans IDE 6.1
File Edit View Navigate Source Refactor Build Run Profile Versioning Tools Window Help
Projects
Services
SendInventory
Process Files
ftpTransfer.wsdl
Referenced Resources
ftpTransfer.wsdl
Source WSDL Partner
http://j2ee.netbeans.org/wsdl/SendInventory/ftpTransfer
Types
Imports
Messages
OutboundOneWayMessagingOperationRequest
Port Types
OutboundOneWayMessagingPortType
Bindings
OutboundOneWayMessagingBinding PortType="Ou
Services
OutboundOneWayMessagingService
Extensibility Elements
ftpTransfer
http://j2ee.netbeans.org...
Properties
Name ftpTransfer
Target Namespace http://j2ee.n...
Default Namespace http://schema...
Prefixes
plnk http://docs.o...
tns http://j2ee.n...
wsdl http://schema...
ftp http://schema...
xsd http://www.w...
http://j2ee.netbeans.org/wsdl/...

**14 Click Save All.**

## ▼ To Modify ftp:message Properties

**1 Double-click `ftpTransfer.wsdl`.**

This action opens the WSDL Editor.

**2 Expand Bindings —> OutboundOneWayMessagingBinding —> OutboundOneWayMessagingOperation —> input1 —> ftp:message.**

**3 Double-click ftp:message.**

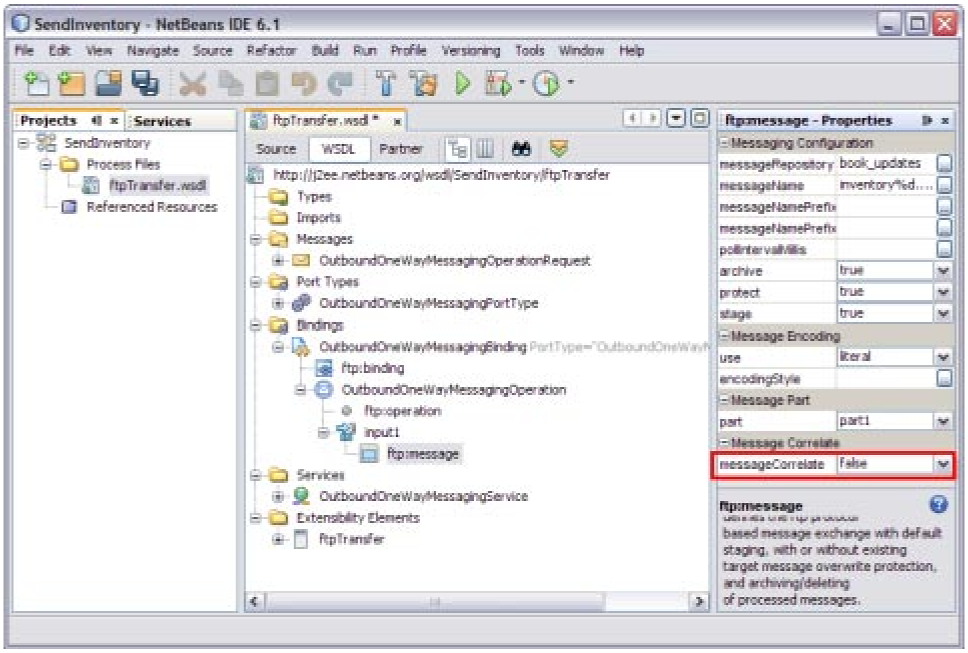
The **ftp:message** Properties window appears.

---

**Note –** Click Window —> Properties if the Properties window is not visible.

---

**4 Click Message Correlate. Choose the Property — false from the drop-down list.**



http://j2ee.netbeans.org/wsdl/SendInventory/ftpTransfer
- Types
- Imports
- Messages
  - OutboundOneWayMessagingOperationRequest
- Port Types
  - OutboundOneWayMessagingPortType
- Bindings
  - OutboundOneWayMessagingBinding PortType="OutboundOneWay
    - ftp:binding
    - OutboundOneWayMessagingOperation
      - ftp:operation
      - input1
        - ftp:message
- Services
  - OutboundOneWayMessagingService
- Extensibility Elements
  - ftpTransfer

| | |
|---|---|
| **Messaging Configuration** | |
| messageRepository | book_updates |
| messageName | inventory%d.... |
| messageNamePrefix | |
| messageNamePrefix | |
| pollIntervalMillis | |
| archive | true |
| protect | true |
| stage | true |
| **Message Encoding** | |
| use | literal |
| encodingStyle | |
| **Message Part** | |
| part | part1 |
| **Message Correlate** | |
| messageCorrelate | false |

**ftp:message**

based message exchange with default staging, with or without existing target message overwrite protection, and archiving/deleting of processed messages.

**5 Click Save All.**

## Poll Request Wizard Properties

The table describes the various FTP Poll Request attributes.

**TABLE 2** Poll Request

| Attribute Name | Description |
|---|---|
| Message Repository | The directory on the remote FTP server where messages will be processed and archived. For more information, see Table 3.<br><br>For example, book_updates |

TABLE 2 Poll Request *(Continued)*

| Attribute Name | Description |
|---|---|
| Message Name | The filename where a message is put into, usually in the form of a name pattern. Pattern is a string containing special characters preceded by a percentage sign. For more information, see Table 4.<br>1. UUID %u is substituted by a UUID value compliant with Java 1.5 UUID.<br>2. In sequence number references, such as %0, %1, %2, %3, %4, %5, %6, %7, %8, %9, the symbol is replaced by the current value of that sequence number, which is an integer count that increments after each reference.<br>For example, `inventory%d.xml`<br>`%d` — This symbol is an integer count that increments after each reference. |
| Message Name Prefix | The prefix for the outbound (OB) message name. |
| Enable Overwrite Protect | Indicates if overwrite protection is required for message send. When this is sent to true, existing messages are moved to a dedicated directory before the current message is put to the target; otherwise, the current message overwrites the existing message. The checkbox is selected by default. |
| Enable Staging When Put Message | Indicates if staging is enabled for message transfer. The check box is selected by default. |
| Payload Processing | 1. **text**: Text Data<br>This radio button is selected by default.<br>2. **xml**: XML Data<br>Click the ellipsis to select an Element or Type.<br>3. **encoded data**: Encoded Data<br>Click the ellipsis to select an Element or Type. |

The following table describes the various FTP message repository directories.

TABLE 3 Message Repository Directories

| Directory | Description |
|---|---|
| /inbox | The default location where the consumer puts the request message and the provider polls for a request message. |
| /instage | The default location where the consumer stages a request message before it is completely uploaded. |
| /inprotect | The default location where consumer moves an existing request message so that it is not overwritten by current request message. |

TABLE 3 Message Repository Directories (Continued)

| Directory | Description |
|---|---|
| /inarchive | The default location where provider archives request message after it is processed. |
| /outbox | The default location where the provider puts a response message and the consumer polls for the response message. |
| /outstage | The default location where the provider stages the response message before it is completely uploaded. |
| /outprotect | The default location where provider moves an existing response message to so that it is not overwritten by current response message. |
| /outarchive | The default location where consumer archives response message after it is processed. |

The following table describes the various Java Timestamp Patterns.

TABLE 4 Java Timestamp Patterns

| Letter | Data or Time Component | Presentation | Examples |
|---|---|---|---|
| G | Era designator | Text | AD |
| y | Year | Year | 1996; 96 |
| M | Month in year | Month | July; Jul; 07 |
| w | Week in year | Number | 27 |
| W | Week in month | Number | 2 |
| D | Day in year | Number | 189 |
| d | Day in month | Number | 10 |
| F | Day of week in month | Number | 2 |
| E | Day in week | Text | Tuesday; Tue |
| a | Am/pm marker | Text | PM |
| H | Hour in day (0-23) | Number | 0 |
| k | Hour in day (1-24) | Number | 24 |
| K | Hour in am/pm (0-11) | Number | 0 |
| h | Hour in am/pm (1-12) | Number | 12 |
| m | Minute in hour | Number | 30 |
| s | Second in minute | Number | 55 |

TABLE 4 Java Timestamp Patterns *(Continued)*

| Letter | Data or Time Component | Presentation | Examples |
|---|---|---|---|
| S | Millisecond | Number | 978 |
| z | Time zone | General time zone | Pacific Standard Time; PST; GMT-08:00 |
| Z | Time zone | RFC 822 time zone | -0800 |

### FTP MessageActivePassive Element (<ftp:messageActivePassive>)

The FTP Message Active/Passive element for the WSDL binding element functions the same as the FTP message element, but contains flags that allow you to set passive FTP messaging for both the consumer and the provider. The following table lists the properties that enable passive FTP messaging for the FTP Message Active/Passive element.

TABLE 5 FTP MessageActivePassive Element

| Attribute Name | Description |
|---|---|
| consumerUsePassive | Specifies whether to use passive FTP on the consumer side.<br><br>Default value: **true** |
| providerUsePassive | Specifies whether to use passive FTP on the provider side.<br><br>Default value: **true** |

## FTP Binding Component Extensibility Elements

The FTP Binding Component is bound to either a service consumer or service provider, and the exposed interfaces are defined by a WSDL document. The FTP Binding Component implements a set of extensibility elements specific to the FTP Binding Component so a service can be defined and bound to a FTP protocol.

The FTP Binding Component supports the following extensibility elements:

1. **Address**: The connectivity element information such as, FTP URL (host, port, login, password), directory listing style, and user defined heuristics for directory listing parsing.
2. **Binding**: A marker element indicating a FTP binding. This element does not have attributes.
3. **Operation**: A marker element indicating a FTP operation. This element does not have attributes.
4. **Transfer**: Specifies a message transfer from a sender and receiver perspective. For example, to specify a message transfer for a service request, there is a sender and a receiver involved and the WSDL document specifies the following:

   a. **The target sender sends to**: Represented by the attribute ftp:sendTo (the target receiver receives from is represented by attribute ftp:receiveFrom). The additional operations performed before a message is sent (PUT) to target or after a message is received (GET) from the target are called Pre/Post operations.
   b. **messageCorrelate**: If enabled, a UUID tagging-based message correlation scheme is used to correlate the request/response of a synchronous service.
5. **Message**: Specifies a message transfer from a service consumer or service provider perspective. The WSDL document can specify the following:
   a. The message repository represented as the **ftp:messageRepository** attribute. A base directory where all the working directories for a message transfer are created, such as,
      i. **inbox**: Used for posting requests (by consumer) and polling request (by provider).
      ii. **instage**: Used for staging requests.
      iii. **outbox**: Used for posting responses (by provider) and polling responses (by consumer).
      iv. **outstage**: Used for staging responses.
   b. **messageCorrelate**: If enabled, a UUID tagging-based message correlation scheme is used to correlate the request/response of a synchronous service.

---

**Note –** Application Variable is a tabular data consisting of one or more name value pairs. On the other hand, WSDL authors are allowed to include references of these 'tokens' in the attributes values of FTP Binding Component extensibility elements in their WSDL, the references are resolved at deployment time.

Application variables are categorized into the following types:

1. STRING
2. BOOLEAN
3. NUMBER
4. PASSWORD

---

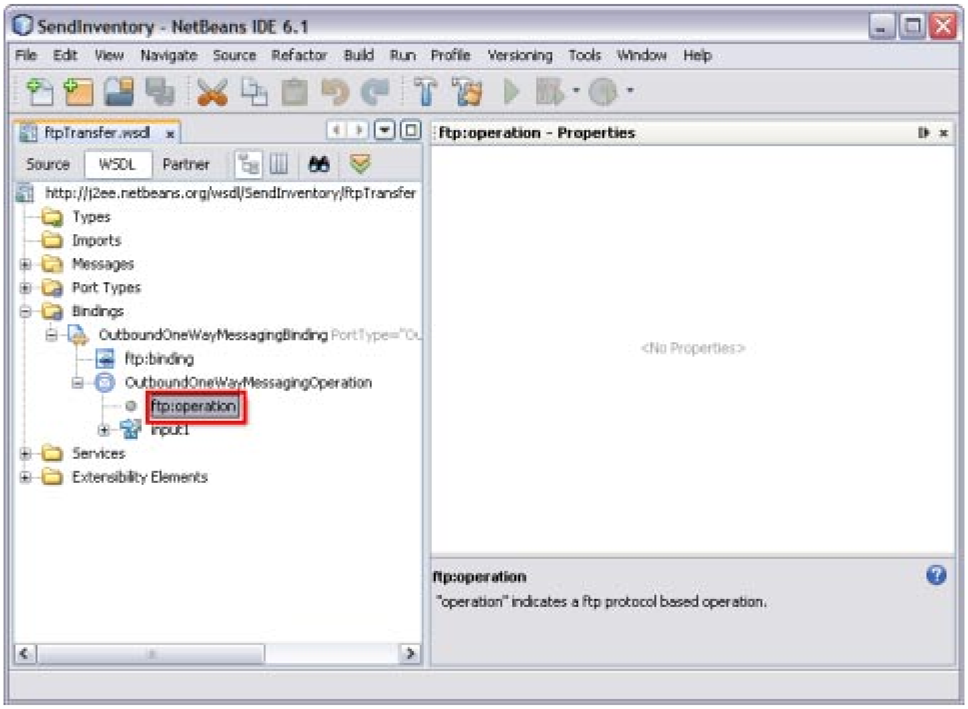
## Runtime Configuration

The runtime configuration for FTP Binding Component includes the following:

1. FTP Operation Element (<ftp:operation>)
2. FTP Binding Element (<ftp:binding>)
3. FTP Transfer Element (<ftp:transfer>)
4. FTP Address Element (<ftp:address>)
5. FTP Message Element (<ftp:message>)

## FTP Operation Element (<ftp:operation>)

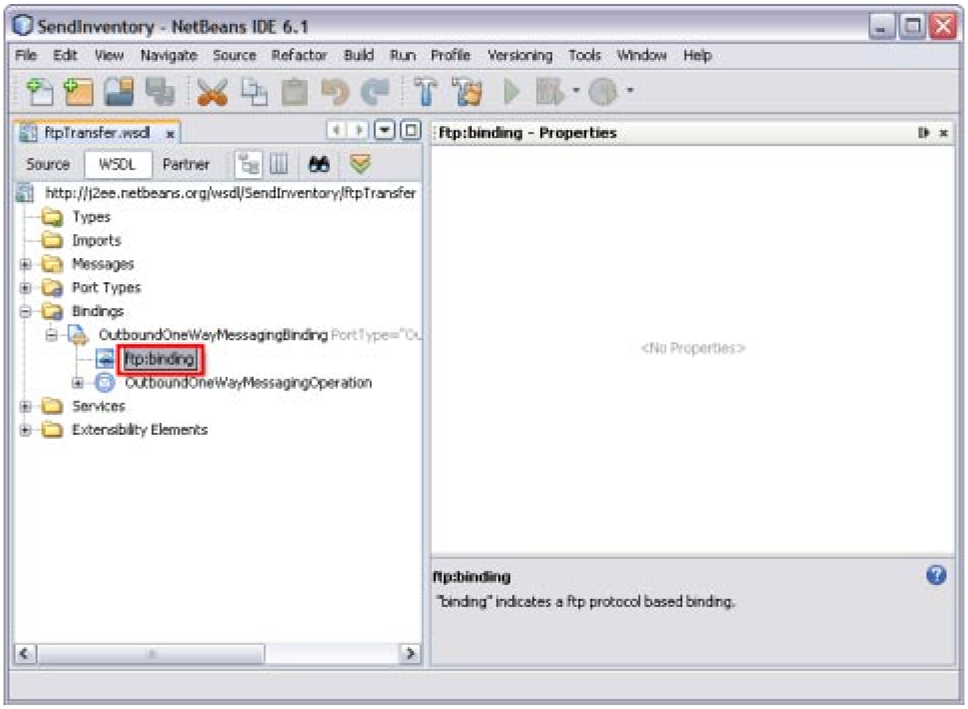
The FTP operation indicates an FTP protocol based operation.


SendInventory - NetBeans IDE 6.1
File Edit View Navigate Source Refactor Build Run Profile Versioning Tools Window Help
ftpTransfer.wsdl
Source WSDL Partner
http://j2ee.netbeans.org/wsdl/SendInventory/ftpTransfer
Types
Imports
Messages
Port Types
Bindings
OutboundOneWayMessagingBinding PortType="Ou
ftp:binding
OutboundOneWayMessagingOperation
ftp:operation
input1
Services
Extensibility Elements
ftp:operation - Properties
<No Properties>

ftp:operation
"operation" indicates a ftp protocol based operation.

## FTP Binding Element (<ftp:binding>)

The FTP binding indicates an FTP protocol based binding.

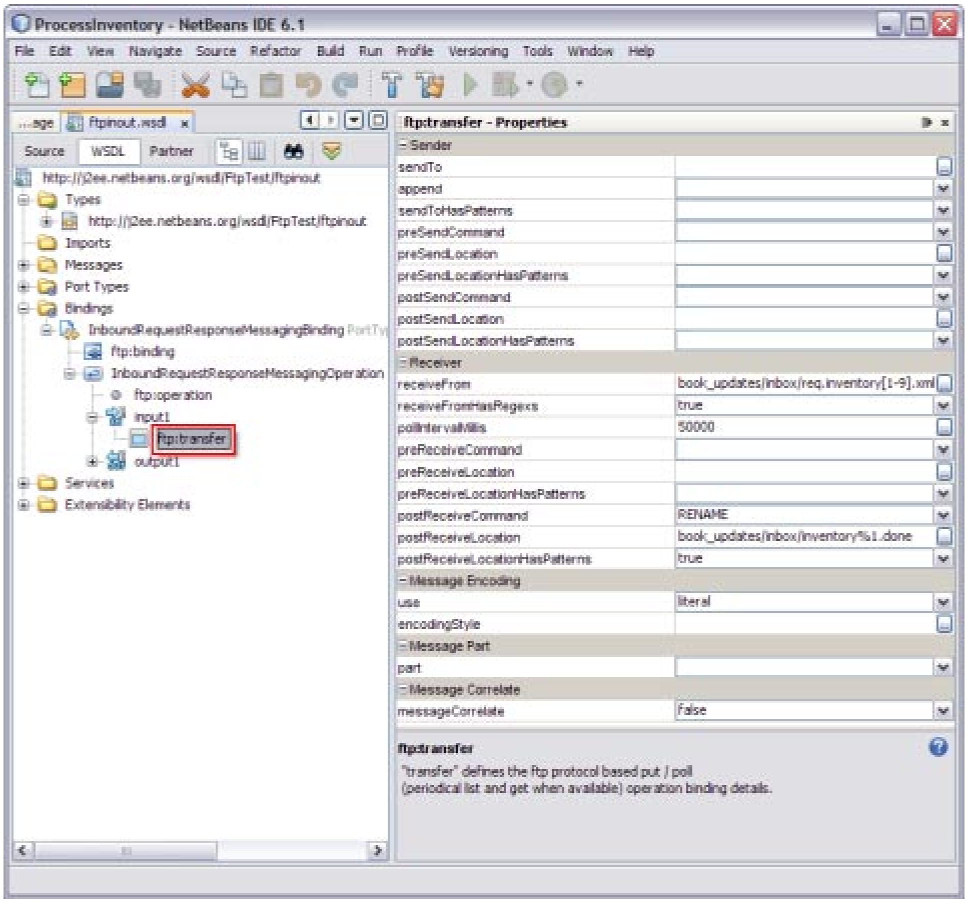

SendInventory - NetBeans IDE 6.1
File Edit View Navigate Source Refactor Build Run Profile Versioning Tools Window Help
ftpTransfer.wsdl
Source WSDL Partner
http://j2ee.netbeans.org/wsdl/SendInventory/ftpTransfer
Types
Imports
Messages
Port Types
Bindings
OutboundOneWayMessagingBinding PortType="Ou
ftp:binding
OutboundOneWayMessagingOperation
Services
Extensibility Elements
ftp:binding - Properties
<No Properties>

ftp:binding
"binding" indicates a ftp protocol based binding.

## FTP Transfer Element (<ftp:transfer>)

The FTP transfer element extends the WSDL binding element to allow you to specify a message transfer from a sender's and a receiver's perspective. Typical reasons to use the FTP transfer element include the following:

- Override the default message repository locations with custom settings
- Use pattern matching when transferring files
- Append the message to the target file


ProcessInventory - NetBeans IDE 6.1
File Edit View Navigate Source Refactor Build Run Profile Versioning Tools Window Help
...age
ftpinout.wsdl
Source
WSDL
Partner
http://j2ee.netbeans.org/wsdl/FtpTest/ftpinout
Types
http://j2ee.netbeans.org/wsdl/FtpTest/ftpinout
Imports
Messages
Port Types
Bindings
InboundRequestResponseMessagingBinding PortTy
ftp:binding
InboundRequestResponseMessagingOperation
ftp:operation
input1
ftp:transfer
output1
Services
Extensibility Elements

ftp:transfer - Properties

| | |
|---|---|
| **Sender** | |
| sendTo | |
| append | |
| sendToHasPatterns | |
| preSendCommand | |
| preSendLocation | |
| preSendLocationHasPatterns | |
| postSendCommand | |
| postSendLocation | |
| postSendLocationHasPatterns | |
| **Receiver** | |
| receiveFrom | book_updates/inbox/req.inventory[1-9].xml |
| receiveFromHasRegexs | true |
| pollIntervalMillis | 50000 |
| preReceiveCommand | |
| preReceiveLocation | |
| preReceiveLocationHasPatterns | |
| postReceiveCommand | RENAME |
| postReceiveLocation | book_updates/inbox/inventory%1.done |
| postReceiveLocationHasPatterns | true |
| **Message Encoding** | |
| use | literal |
| encodingStyle | |
| **Message Part** | |
| part | |
| **Message Correlate** | |
| messageCorrelate | False |

**ftp:transfer**

"transfer" defines the ftp protocol based put / poll
(periodical list and get when available) operation binding details.

**TABLE 6** FTP Transfer Element

| Attribute Name | Description |
|---|---|
| sendTo | The FTP PUT target for the message. Specify the target using either a relative path or absolute path form:<br>■ **relative path**<br>`pattern1/pattern2/. . ./filename`<br>■ **absolute path**<br>`/pattern1/pattern2/. . ./filename`<br>**patterN** and **filename** can be literal or a pattern, depending on the value of the `sendToHasPatterns` property.<br>For more information, see Table 7. |

TABLE 6 FTP Transfer Element *(Continued)*

| Attribute Name | Description |
|---|---|
| append | Specifies whether the message is appended to the target file specified by the `sendTo` property.<br>■ If **append** is true, the message is appended to the target file.<br>■ If **append** is false, then the message overwrites the target file.<br><br>Default value: **false** |
| sendToHasPatterns | Specifies if the target specified by the `sendTo` property contains patterns in its path components and/or file name.<br><br>If `sendToHasPatterns` is **false**, then the target path and filename are literals.<br><br>Default value: **false**<br><br>For more information, see Table 7. |
| receiveFrom | The target to poll for FTP GET messages. Specify the target using either a relative path or absolute path form:<br>■ **relative path**<br>`pattern1/pattern2/. . ./filename`<br>■ **absolute path**<br>`/pattern1/pattern2/. . ./filename`<br>**patterN** and **filename** can be literal or a regular expression, depending on the value of the `receiveFromHasRegexs` property.<br><br>For more information, see Table 7. |
| receiveFromHasRegexs | Specifies if the target specified by the `receiveFrom` property contains regular expressions in its path components and/or file name. If `receiveFromHasRegexs` is **false**, then the target path and filename are literals.<br><br>Default value: **false** |
| pollIntervalMillis | The interval (in milliseconds) for an inbound component (receiver) to poll the target.<br><br>Default value: **5000** |
| preSendCommand | The operation to perform before sending an FTP message. Values can be NONE, COPY, or RENAME<br><br>Default value: NONE |

TABLE 6 FTP Transfer Element *(Continued)*

| Attribute Name | Description |
|---|---|
| preSendLocation | The destination file to be used with the `preSendCommand`. Specify the file using either a relative path or absolute path:<br>■ **relative path**<br>`pattern1/pattern2/. . ./filename`<br>■ **absolute path**<br>`/pattern1/pattern2/. . ./filename`<br>**patterN** and **filename** can be literal or a pattern, depending on the value of the `preSendLocationHasPatterns` property.<br>For more information, see Table 7. |
| preSendLocationHasPatterns | Specifies if the file specified by the `preSendLocation` property contains patterns in its path components or file name.<br>If `preSendLocationHasPatterns` is **false**, then the target path and filename are literals.<br>Default value: **false** |
| postSendCommand | Specifies the operation to perform after sending a message. The value can be either NONE or RENAME.<br>Default value: NONE |
| postSendLocation | The destination file for the `postSendCommand`. Specify the file using either a relative path or absolute path:<br>■ **relative path**<br>`pattern1/pattern2/. . ./filename`<br>■ **absolute path**<br>`/pattern1/pattern2/. . ./filename`<br>**patterN** and **filename** can be literal or a pattern, depending on the value of the `postSendLocationHasPatterns` property.<br>For more information, see Table 7. |
| preReceiveCommand | Specifies the operation to perform before receiving an FTP message. Values can be either NONE, COPY, RENAME<br>Default value: NONE |

TABLE 6 FTP Transfer Element *(Continued)*

| Attribute Name | Description |
|---|---|
| preReceiveLocation | The destination file to be used with the `preReceiveCommand`. Specify the file using either a relative path or absolute path:<br>■ **relative path**<br>`pattern1/pattern2/. . ./filename`<br>■ **absolute path**<br>`/pattern1/pattern2/. . ./filename`<br>**patterN** and **filename** can be literal or a pattern, depending on the value of the `preReceiveLocationHasPatterns` property.<br>For more information, see Table 7. |
| preReceiveLocationHasPatterns | Specifies if the file specified by the `preReceiveLocation` property contains patterns in its path components and/or file name.<br>If `preReceiveLocationHasPatterns` is **false**, then the target path and filename are literals.<br>Default value: **false** |
| postReceiveCommand | Specifies the operation to perform after receiving an FTP message. Values can be either NONE, DELETE, or RENAME.<br>Default value: NONE |
| postReceiveLocation | The destination file to be used with the `preReceiveCommand`. Specify the file using either a relative path or absolute path:<br>■ **relative path**<br>`pattern1/pattern2/. . ./filename`<br>■ **absolute path**<br>`/pattern1/pattern2/. . ./filename`<br>**patterN** and **filename** can be literal or a pattern, depending on the value of the `preReceiveLocationHasPatterns` property.<br>For more information, see Table 7. |
| postReceiveLocationHasPatterns | Specifies if the file specified by the `preReceiveLocation` property contains patterns in its path components and/or file name.<br>If `preReceiveLocationHasPatterns` is **false**, then the target path and filename are literals.<br>Default value: **false** |
| senderUsePassive | Specifies whether to use passive FTP on the sender side.<br>Default value: **true** |

TABLE 6 FTP Transfer Element (Continued)

| Attribute Name | Description |
|---|---|
| ReceiverUsePassive | Specifies whether to use passive FTP on the receiver side.<br><br>Default value: **true** |
| use | Specifies whether a message (or message part) is literal or encoded. If encoded is specified, then you must also specify the encoder using the `encodingStyle` property.<br><br>Default value: **Literal** |
| encodingStyle | Specifies the encoding type associated with the message (or message part). This also defines the encoder type responsible to process the encoded data.<br><br>There is no default value: ud1encoder-1.0 |
| part | References which part of the message described in the abstract WSDL to use for the message. It is **Part1**.<br><br>If the `part` property is not specified, then the first part listed in the abstract WSDL is used. |
| messageCorrelate | Specifies whether UUID tagging-based message correlation is enabled. If true, then the message correlation is enabled. Otherwise, message correlation is not enabled.<br><br>Default value: **false** |

## Pattern Matching

You can use pattern matching to generate filenames for messages and to retrieve messages according to the generated filename patterns. The following message properties make use of pattern matching:

- sendTo
- sendToHasPatterns
- receiveFrom
- receiveFromHasPatterns
- preSendLocation
- preSendLocationHasPatterns
- postSendLocation
- postSendLocationHasPatterns
- preReceiveLocation
- preReceiveLocationHasPatterns
- postReceiveLocation
- postReceiveLocationHasPatterns

The `%` character precedes a character that indicates the pattern to be expanded. For example, %y%y%y%y expands to 2007.

Use an additional `%` as an escape character to print the `%` character as a literal. For example, %%y%%y%%y%%y expands to %y%y%y%y.

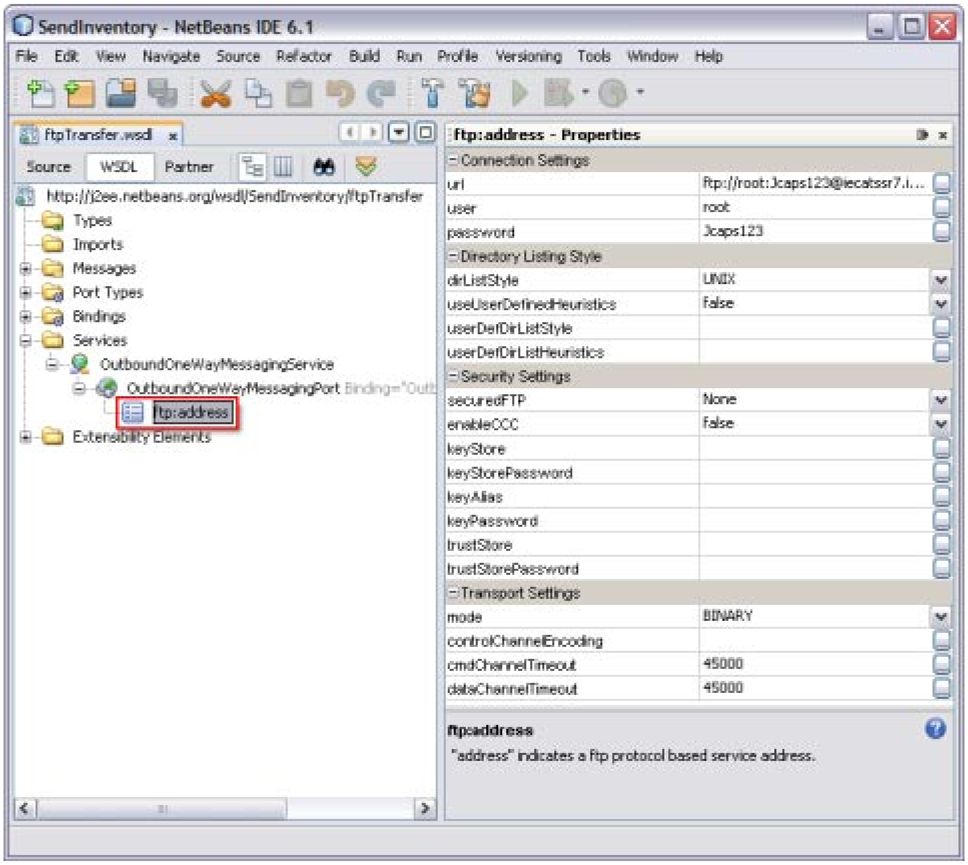
The table describes various Pattern Matching for FTP Binding Component Message Transfer Targets

**TABLE 7** Pattern Matching for FTP Binding Component Message Transfer Targets

| Pattern Type | Description |
|---|---|
| Timestamp | The FTP Binding Component specifies a timestamp using the simple Java date/time formats:<br><br>%[GyMdhHmsSEDFwWakKz]<br><br>For example, abc%y%y%y%y expands to abc2007.<br><br>For more information, see Table 4. |
| Directory and Filename Replacement | `%p/%f`<br><br>Typically used to specify the directory name and filename for pre-transfer and post-transfer operations.<br><br>For example, if `sendTo` specifies `my_in_box/invoice.dat`, then the following pattern:<br><br>`%p_backup/%f.bak`<br><br>Expands to:<br><br>`my_in_box_backup/invoice.dat.bak` |
| UUID | `%u`<br><br>Inserts a UUID value compliant with Java 1.5 UUID. |
| Sequence Numbering | `%0, %1, ... , %9`<br><br>Inserts the current value of a sequence counter that is incremented after each reference. The initial value of a sequence counter is 0. There can be as many as ten sequence counters at runtime, identified as %0 through %9.<br><br>The sequence counters are not persisted and will be reset to 0 after either of the following occurrences:<br>■ The application server restarts.<br>■ The FTP Binding Component is redeployed. |

## FTP Address Element (<ftp:address>)

The FTP address element extends the WSDL service element to allow you to specify the connectivity information to an FTP server. You can specify properties such as the Internet address of the FTP server, style for listing directories, the mode of transfer (for example, binary, ASCII, or EBCDIC), and timeout settings for connecting to the FTP server.



http://j2ee.netbeans.org/wsdl/SendInventory/ftpTransfer
- Types
- Imports
- Messages
- Port Types
- Bindings
- Services
  - OutboundOneWayMessagingService
    - OutboundOneWayMessagingPort Binding="Outb
      - ftp:address
- Extensibility Elements

ftp:address - Properties

| | |
|---|---|
| **Connection Settings** | |
| url | ftp://root:Jcaps123@iecatssr7.i... |
| user | root |
| password | Jcaps123 |
| **Directory Listing Style** | |
| dirListStyle | UNIX |
| useUserDefinedHeuristics | false |
| userDefDirListStyle | |
| userDefDirListHeuristics | |
| **Security Settings** | |
| securedFTP | None |
| enableCCC | false |
| keyStore | |
| keyStorePassword | |
| keyAlias | |
| keyPassword | |
| trustStore | |
| trustStorePassword | |
| **Transport Settings** | |
| mode | BINARY |
| controlChannelEncoding | |
| cmdChannelTimeout | 45000 |
| dataChannelTimeout | 45000 |

**ftp:address**
"address" indicates a ftp protocol based service address.

TABLE 8 FTP Address Element

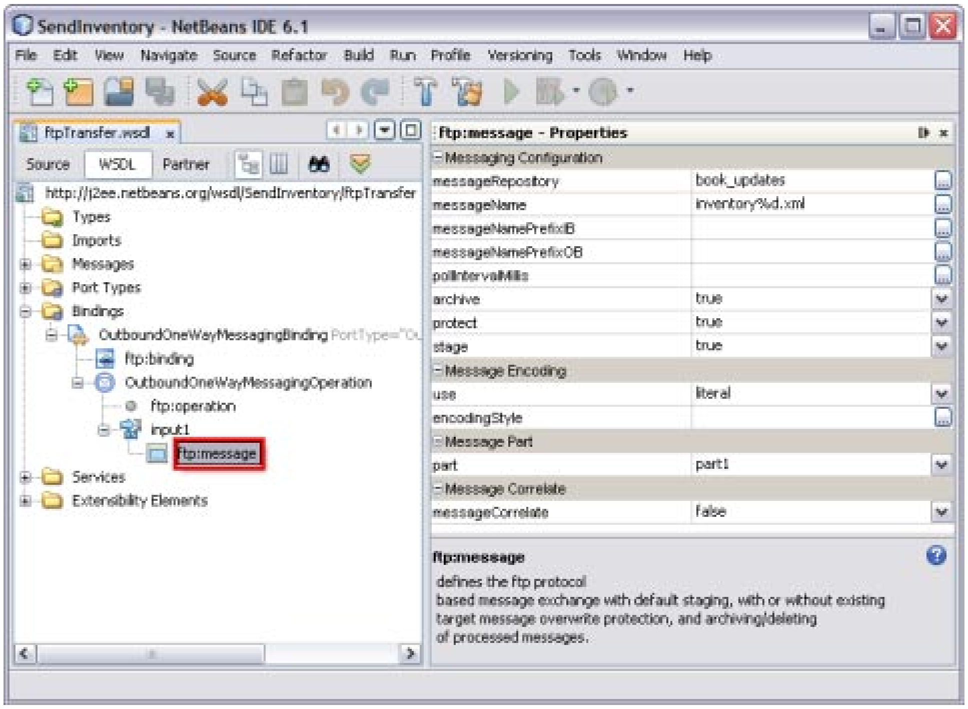
| Attribute Name | Description |
|---|---|
| url | The address of the FTP host. This address must be in the following format:<br><br>`ftp://[user]:[password]@[hostname or IP address]:[port]`<br>■ **user** is the FTP account login<br>■ **password** is the password for the account<br>■ **hostname** is the hostname for where the FTP server is running<br>■ **IP address** is the IP address for the FTP host<br>■ **port** is the FTP port (default is 21)<br><br>Default value: `ftp://anonymous:user\@yahoo.com@localhost:21` |
| user | The FTP login ID.<br><br>Enter the user ID in **ftp:url** format. |
| password | The FTP login password. |
| dirListStyle | Specifies which style to use for listing directories. Choose a style from the list.<br><br>**Note –** The style you select should match the platform for the target FTP host to make sure FTP operations and pattern matching specifications.<br><br>Default value: UNIX |
| useUserDefinedHeuristics: | Indicates whether to use a user-defined style for listing directories. If the property is set to true, then you must specify a style using the `userDefDirListStyle` property.<br><br>Default value: **false** |
| userDefDirListStyle | Names the user-defined directory listing style, which should correspond to the style defined by the property `userDefDirListHeuristics`. |
| userDefDirListHeuristics | The path pointing to a user provided heuristics file. This file should be accessible to the FTP Binding Component runtime in the deployed environment. |
| enabledCCC | Enables Clear Command Channel after handshake. This is only applicable when securedFTP is set to ExplicitSSL. |
| keyStore | The key store location. |
| keyStorePassword | The ey store password. |
| keyAlias | Key alias for client authentication |
| keyPassword | The password that protects the key alias. |
| trustStore | The trust store location. |

TABLE 8 FTP Address Element *(Continued)*

| Attribute Name | Description |
|---|---|
| trustStorePassword | The trust store password. |
| mode | Specifies whether the transfer is binary, ASCII, or EBCDIC.<br><br>Default value: **binary** |
| controlChannelencoding | Encoding (Charset) for FTP control channel IO. Default is ISO-8859-1. When left blank, the default is assumed. |
| cmdChannelTimeout | The time, in milliseconds, to attempt reading the socket for the FTP command channel before a timeout occurs. 0 (zero) indicates no timeout.<br><br>Default value: **45000** |
| dataChannelTimeout | The time, in milliseconds, to attempt reading the socket for the FTP data channel before a timeout occurs. 0 (zero) indicates no timeout.<br><br>Default value: **45000** |

## FTP Message Element (<ftp:message>)

The FTP message element extends the WSDL binding element to allow you to specify message transfer details. These details include the locations on an FTP server on which a message is persisted, staged, retrieved, and archived.


SendInventory - NetBeans IDE 6.1
File Edit View Navigate Source Refactor Build Run Profile Versioning Tools Window Help
ftpTransfer.wsdl
Source WSDL Partner
http://j2ee.netbeans.org/wsdl/SendInventory/ftpTransfer
Types
Imports
Messages
Port Types
Bindings
OutboundOneWayMessagingBinding PortType="O
ftp:binding
OutboundOneWayMessagingOperation
ftp:operation
input1
ftp:message
Services
Extensibility Elements

ftp:message - Properties

| | |
|---|---|
| Messaging Configuration | |
| messageRepository | book_updates |
| messageName | inventory%d.xml |
| messageNamePrefixIB | |
| messageNamePrefixOB | |
| pollIntervalMillis | |
| archive | true |
| protect | true |
| stage | true |
| Message Encoding | |
| use | literal |
| encodingStyle | |
| Message Part | |
| part | part1 |
| Message Correlate | |
| messageCorrelate | false |

**ftp:message**

defines the ftp protocol based message exchange with default staging, with or without existing target message overwrite protection, and archiving/deleting of processed messages.

TABLE 9 FTP Message Element

| Attribute Name | Description |
|---|---|
| Message Repository | The path to a directory on the remote FTP server where messages will be processed and archived.<br><br>For more information, see Table 3. |
| Message Name | The filename where a message is put into, usually in the form of a name pattern. Pattern is a string containing special characters preceded by a percentage sign. The following are all the symbols supported:<br>1. UUID %u is substituted by a UUID value compliant with Java 1.5 UUID.<br>2. In a sequence number reference %0, %1, %2, %3, %4, %5, %6, %7, %8, %9, this symbol is replaced by the current value of the sequence number, which is an integer count that increment after each reference.<br><br>Default value: **%u**<br><br>For more information, see Table 4. |

TABLE 9 FTP Message Element *(Continued)*

| Attribute Name | Description |
|---|---|
| messageNamePrefixIB | A prefix placed before the value of `messageName` to form the actual message name for messages transported in the INBOUND direction (consumer to provider). This property should not contain any pattern symbols or the percentage sign '%'.<br><br>Default value: **req** |
| messageNamePrefixOB | A prefix placed before the value of `messageName` to form the actual message name for messages transported in the OUTBOUND direction (provider to consumer). This property should not contain any pattern symbols or the percentage sign '%'.<br><br>Default value: **resp** |
| pollintervalMillis | The polling interval (in milliseconds) for an inbound component (receiver) to poll the remote target.<br><br>Default value: **5000** |
| archive | Specifies whether a message is archived after it is fetched by a component (either consumer or provider). When set to true, the message is archived (moved to a destination directory such as `inarchive` or `outarchive`. When set to false, the message is deleted.<br><br>Default value: **true** |
| protect | Specifies how a message is moved to a dedicated directly such as `inprotect` or `outprotect` before an incoming message overwrites it.<br>■ If true, the message is moved (protected).<br>■ If false, the message is overwritten.<br><br>Default value: **true** |
| stage | Specifies whether staging is enabled during message transfer.<br><br>Default value: **true** |
| use | Specified if the message is literal or encoded If encoded is specified, you must also specify the encoder using the `encodingStyle` property.<br><br>Default value: **Literal** |
| encodingStyle | Specifies the encoding type associated with the message. This also defines the encoder type responsible to process the encoded data. The value specified here is only applied if the use property is set to encoded.<br><br>Has no default value. ud1encoder-1.0 |

**TABLE 9** FTP Message Element *(Continued)*

| Attribute Name | Description |
|---|---|
| part | References which part of the message described in the abstract WSDL document is to be used for the message. If the part property is not specified, then the first part listed in the abstract WSDL is used.<br><br>Default value: **Part1** |
| messageCorrelate | Specifies whether UUID tagging-based message correlation is enabled. If true, then message correlation is enabled.<br><br>Default value: **true** |

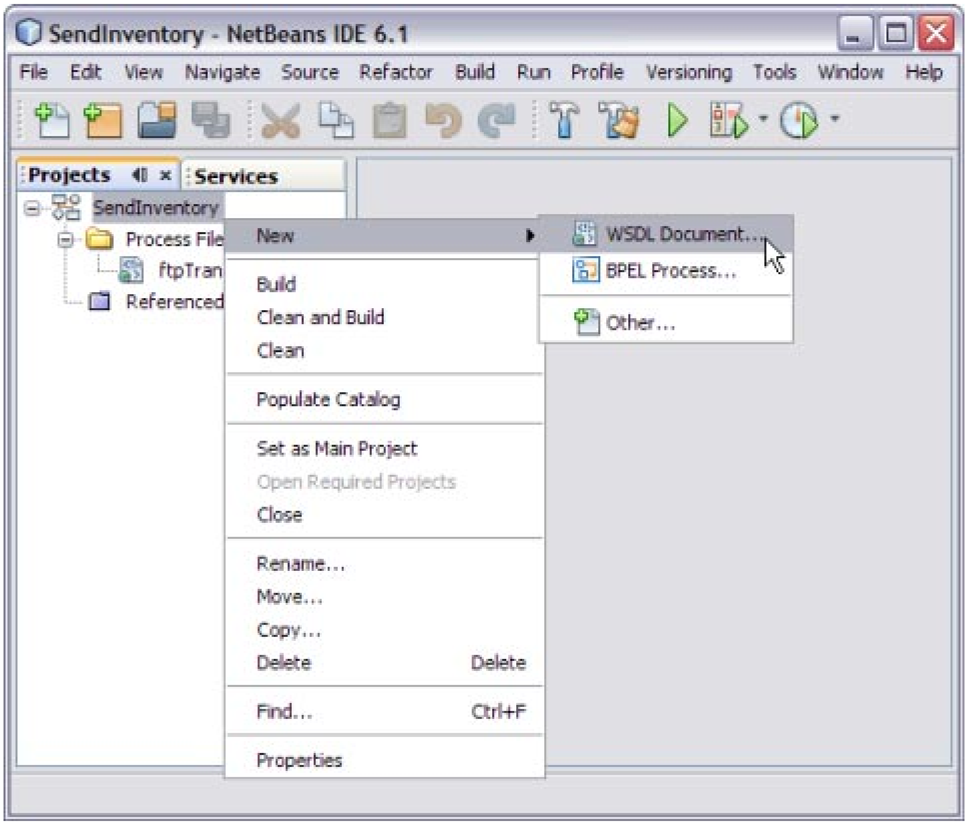
## Creating a WSDL Document : Using FILE

In this section, you add a WSDL document, the `fileTrigger.wsdl`, to the BPEL Module project. After adding the WSDL document, use the partner view of the WSDL editor to configure the components.

## ▼ To Create a WSDL Document : fileTrigger

1. **Expand the SendInventory BPEL Module project node in the Projects tab.**

2. **Right-click the project node or Process Files node. Point to New and select WSDL Document.**



The ftpTransfer is one of the sub-nodes in the tree structure.

The New WSDL Document wizard appears.

3. **In the File Name field, enter `fileTrigger.wsdl`.**

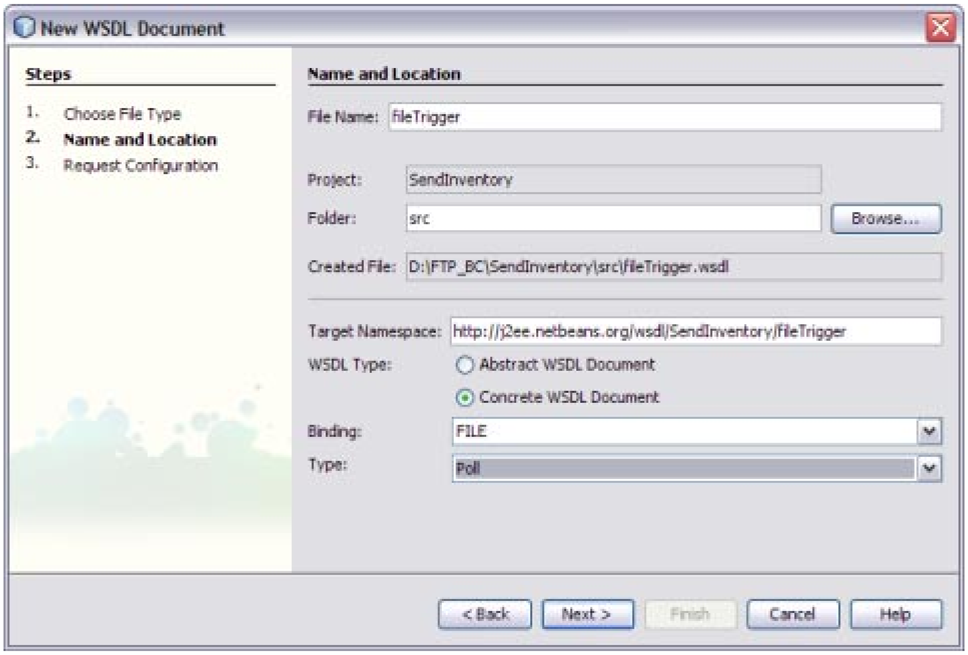
4. **Select Concrete WSDL Document.**

5. **In the Binding field, select FILE from the drop-down list.**

6. **Choose any one of the following Types from the drop-down list.**

- **Poll**: Choose this type for a scenario when the File BC polls for messages from a file directory and invokes a JBI service with the messages.
- **Poll and Write Back Reply**: Choose this type for a scenario when the File BC polls for messages from a file directory, invokes a JBI service, and writes the responses back to the directory.
- **Write**: Choose this type for a scenario when a JBI service invokes File BC to write a message to a file directory.
- **On Demand Read**: Choose this type for a scenario when a JBI service invokes File BC to read a specific message from a file directory.

**7 Select Poll from the Type drop-down list.**


New WSDL Document

Steps
1. Choose File Type
2. **Name and Location**
3. Request Configuration

Name and Location

File Name: fileTrigger

Project: SendInventory

Folder: src

Created File: D:\FTP_BC\SendInventory\src\fileTrigger.wsdl

Target Namespace: http://j2ee.netbeans.org/wsdl/SendInventory/FileTrigger

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

Binding: FILE

Type: Poll

**8 Click Next.**

The New WSDL Document Request Configuration window appears.

**9 Enter the following fields:**

**a. File Name* (pattern): Defines the file name relative to the specified directory.**

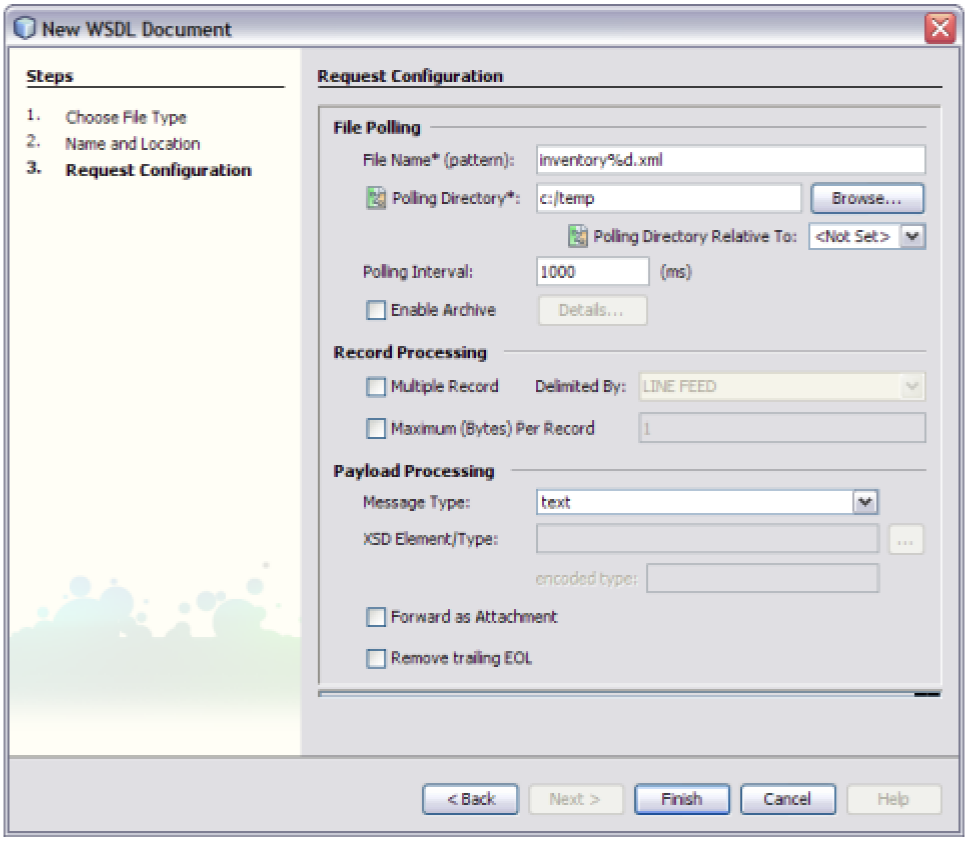
If fileNameIsPattern is not true, this attribute specifies an actual file name. Otherwise, this attribute specifies a pattern marker used for filtering input files from the directory, or a file name format to write to the directory. The supported patterns are:

- **%d**: Denotes an unique number for input and an one-up sequence number for output file names.
- **%u**: Denotes a wild card match for input and an UUID for output file names.
- **%t**: Denotes an unique timestamp for both input and output file names. The expected date format is yyyymmdd-HH-mm-ss-SSS. For input file names, the -HH-mm-ss-SSS part may be omitted to guarantee unique names.
- %{ }: Denotes an integer number in the input file name or a one up sequence number persisted in a sequence file if it is for a output file.

For example, inventory%d.xml

b. **Polling Directory*: Defines the directory name where the WSDL provisioner reads the input files and where the client writes the files.**

For example, `c:/temp`.


New WSDL Document

Steps
1. Choose File Type
2. Name and Location
3. **Request Configuration**

Request Configuration

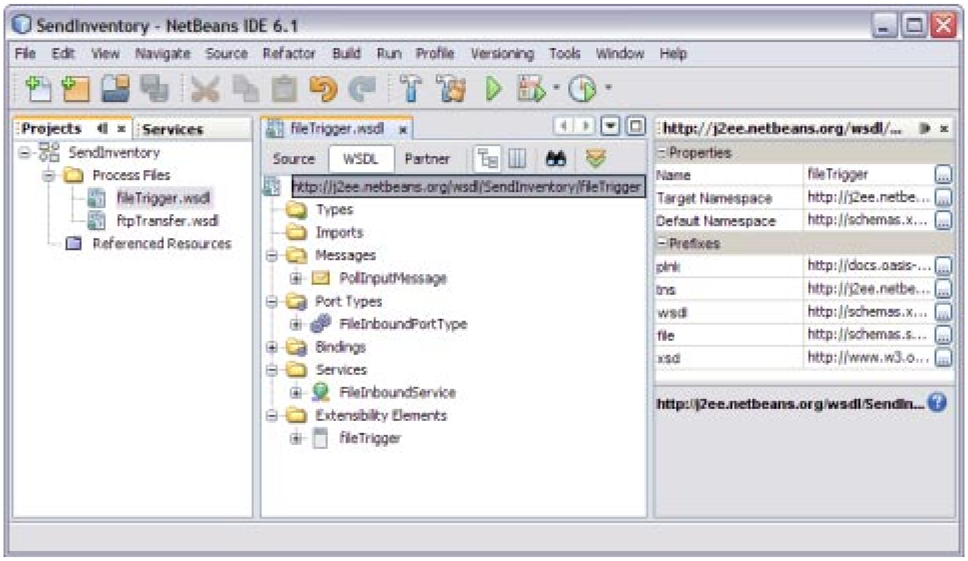
File Polling
File Name* (pattern): inventory%d.xml
Polling Directory*: c:/temp
Polling Directory Relative To: <Not Set>
Polling Interval: 1000 (ms)
Enable Archive

Record Processing
Multiple Record Delimited By: LINE FEED
Maximum (Bytes) Per Record 1

Payload Processing
Message Type: text
XSD Element/Type:
encoded type:
Forward as Attachment
Remove trailing EOL

**10 Click Finish.**

This action opens the Project tree structure. In the current example, the WSDL editor for fileTrigger is displayed along with its properties.



**11 Click Save All.**

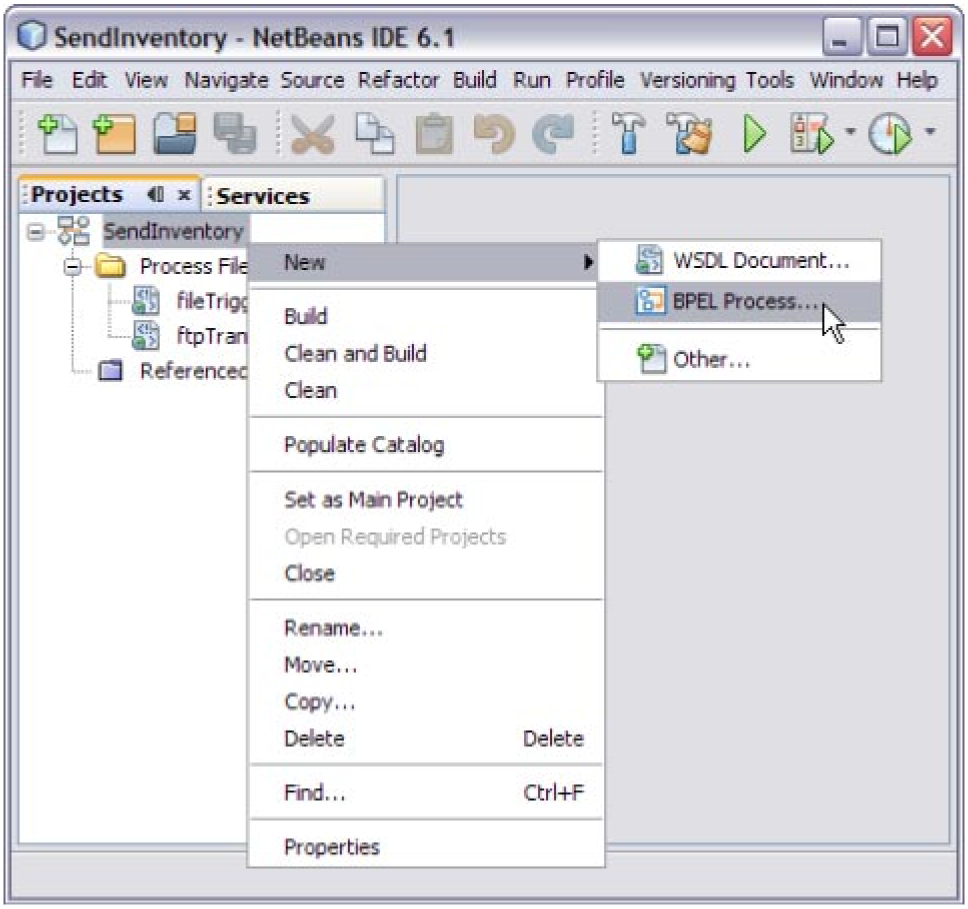
## Creating a BPEL Process

In this section, you add a BPEL process file named `sendInventoryBP.bpel`. This example also illustrates adding a partner link and activities to the BPEL process file.

### ▼ To Create a BPEL Process

**1 Expand the SendInventory BPEL Module project node in the Projects window.**

2 **Right-click the BPEL Module project name or Process Files node. Point to New and select BPEL Process.**



The New BPEL Process wizard appears.

3 **Type `sendInventory` in the File Name field.**

4 **Click Finish.**

**Note –**

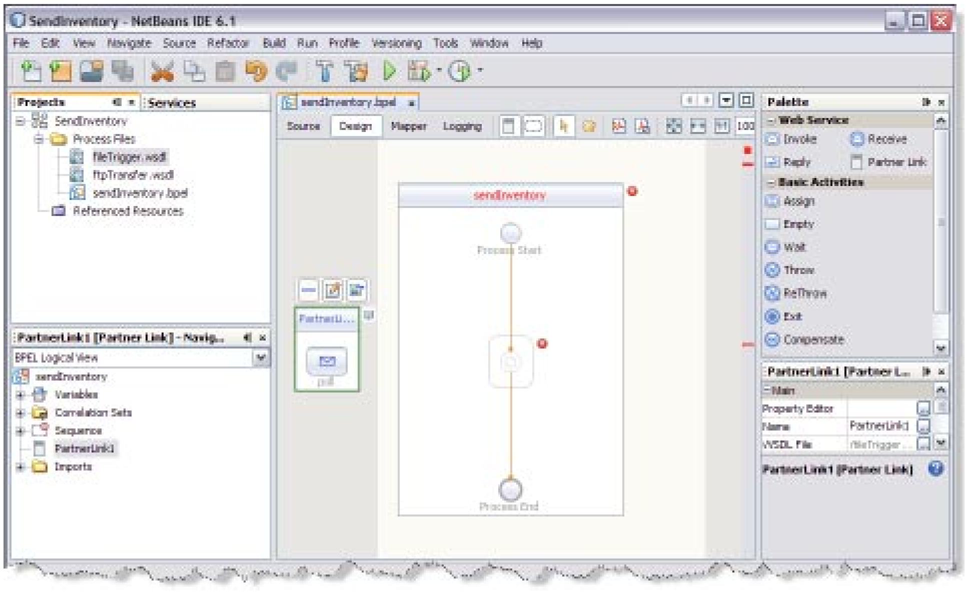
- In the Projects window, the IDE adds a `sendInventory.bpel` node under the Process Files node.
- The `sendInventory.bpel` file appears in the BPEL Designer.
- If the Properties window is not visible, click Window and then click Properties.
- The Navigator window appears showing the BPEL Logical View of the BPEL Process document.

## ▼ To Add a Partner Link

1 **Select the Partner Link from the Projects tab.**

This is the Input WSDL (for example, fileTrigger.wsdl).

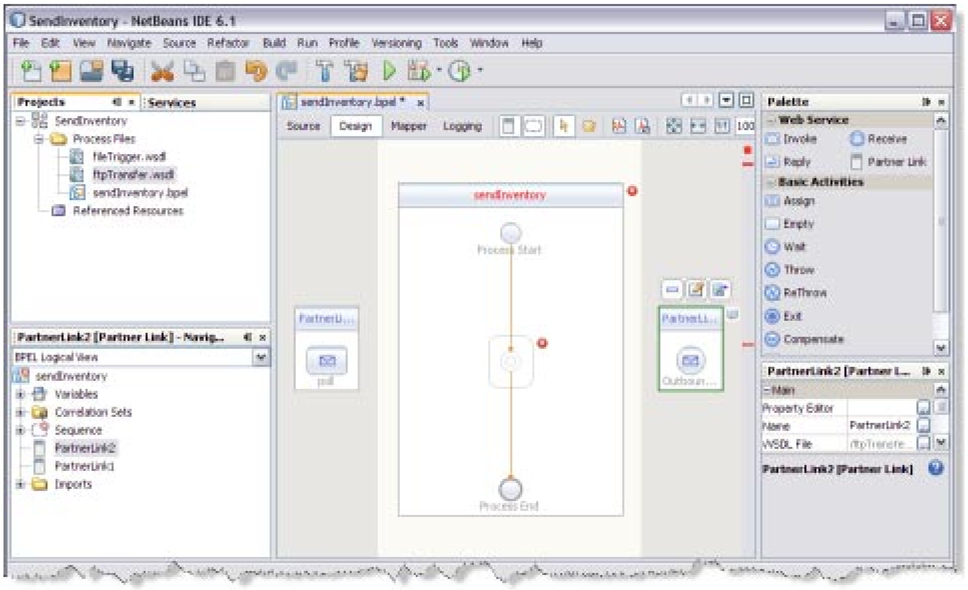
2 **Drag and drop the File WSDL document (fileTrigger.wsdl) to the left panel of the design area.**



3 **Select the ftpTransfer.wsdl Partner Link from the Projects tab.**

This is the Output WSDL.

**4 Drag and drop the ftpTransfer.wsdl FTP WSDL Document to the right panel of the design area.**



## ▼ To Add Web Services and Basic Activities

Drag and drop the following web services:

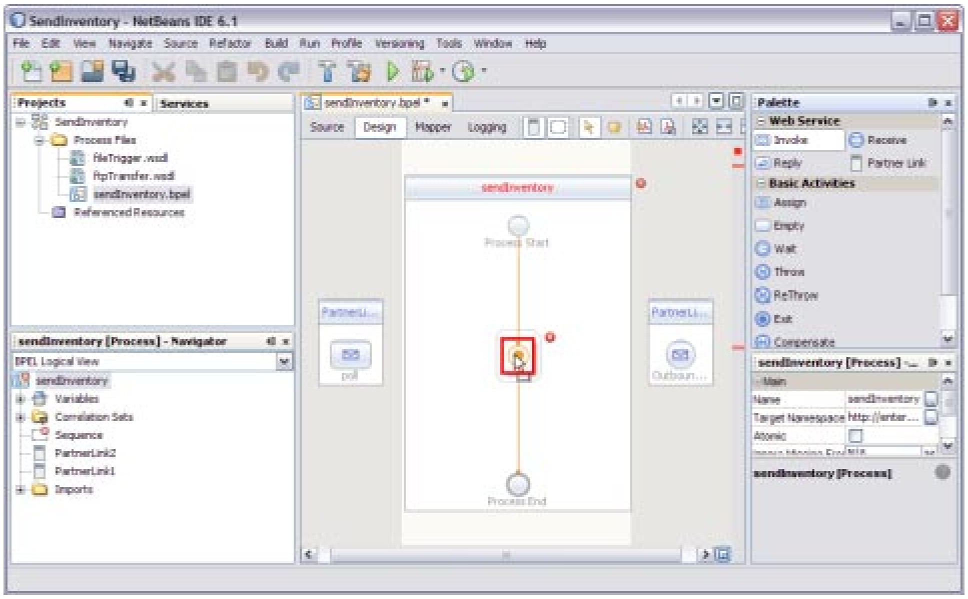
- Receive
- Invoke

Drag and drop the basic activities : Assign.

**1 Select the Web Service : Receive in the Web Service section of the Palette.**

2 **Drag the selection to the sendInventory Process box in the design area between the Process Start and the Process End activities.**

The IDE provides the visual clues to show an appropriate location to drop the selection.



This action places a Web Service : Receive1 in the Design view.

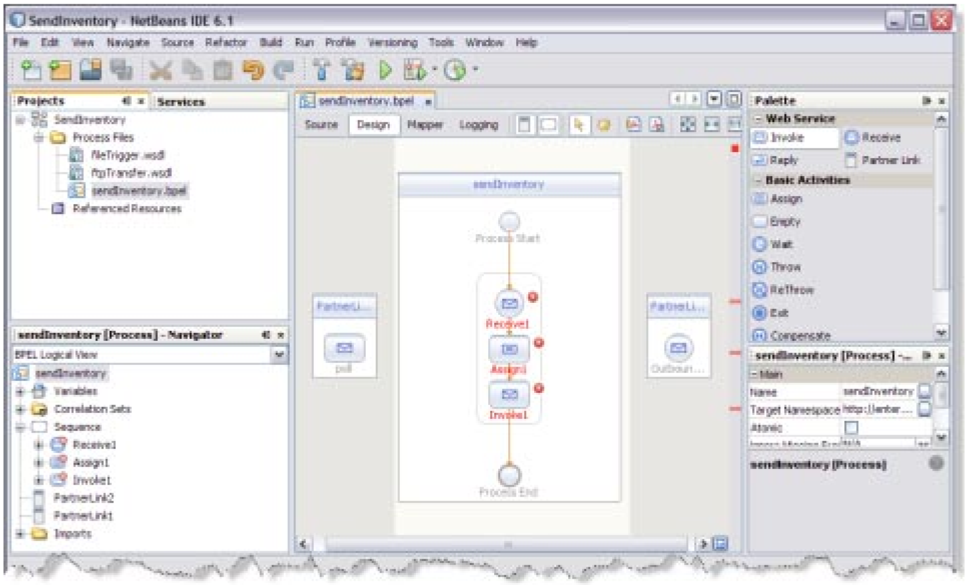
3 **Select the Basic Activities : Assign in the Basic Activities section of the Palette.**

This action places a Assign activity called Assign1 in the Design view.

4 **Select the Web Service : Invoke in the Web Service section of the Palette.**

This action places a Assign activity called Invoke1 in the Design view.

**5 Click Save All.**

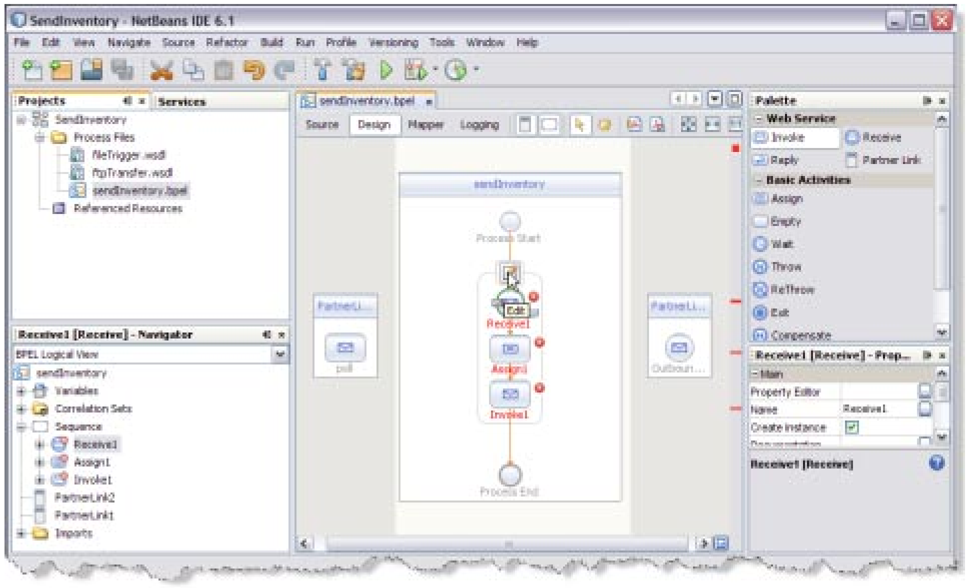


---

**Note –** In the diagram, a red cross next to an element means that the element has not passed validation and the output contains errors. Edit each Sequence to pass validation.

---

The icon symbolizes the Web Services that can be edited.



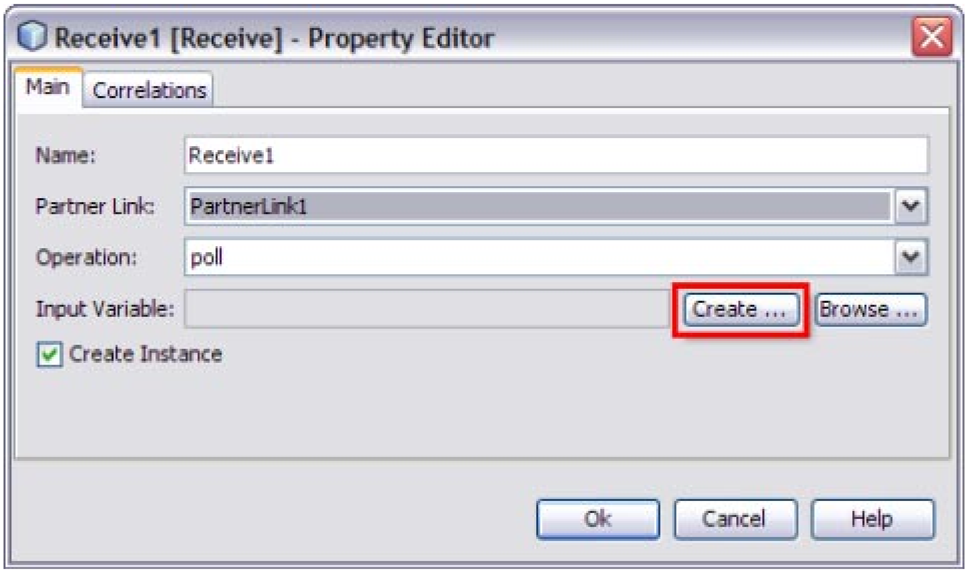
## ▼ To Edit Web Service : Receive1

1 **Click Web Service — Receive1 and click Edit.**

This opens the Receive1 [Receive] - Property Editor.

**2 Select the properties from the Main tab. In this example, select PartnerLink1 from the drop-down list.**

The IDE populates poll against the Operation field.



Receive1 [Receive] - Property Editor

Main | Correlations

Name: Receive1

Partner Link: PartnerLink1

Operation: poll

Input Variable:

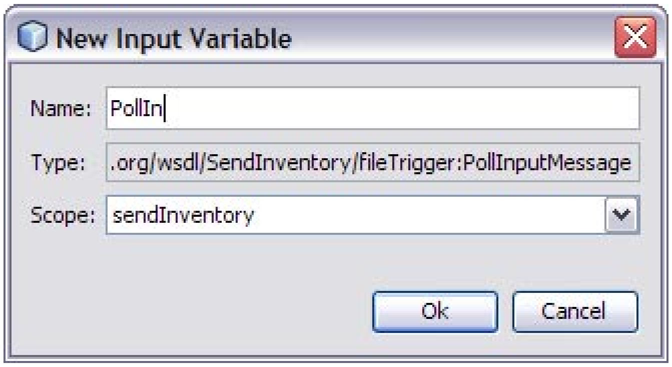
[✓] Create Instance

**3 Do the following to create a New Input Variable:**

- Click the Create button next to the Input Variable field.

  The New Input Variable dialog box appears.
- The default values assigned in the Name, Type, and Scope fields are populated for the variable.

  The value in the Name field can be changed.
- Click OK.



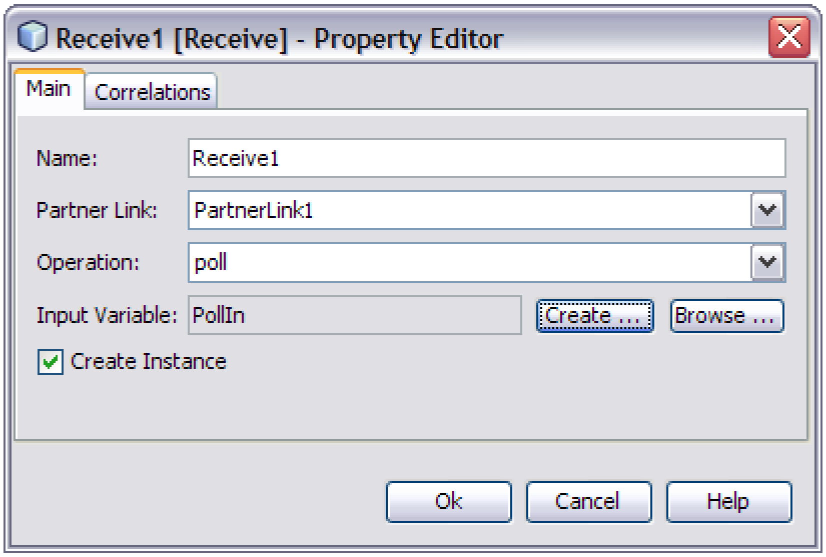
New Input Variable

Name: PollIn

Type: .org/wsdl/SendInventory/fileTrigger:PollInputMessage

Scope: sendInventory

Input Variable — PollIn



Receive1 [Receive] - Property Editor

Name: Receive1

Partner Link: PartnerLink1

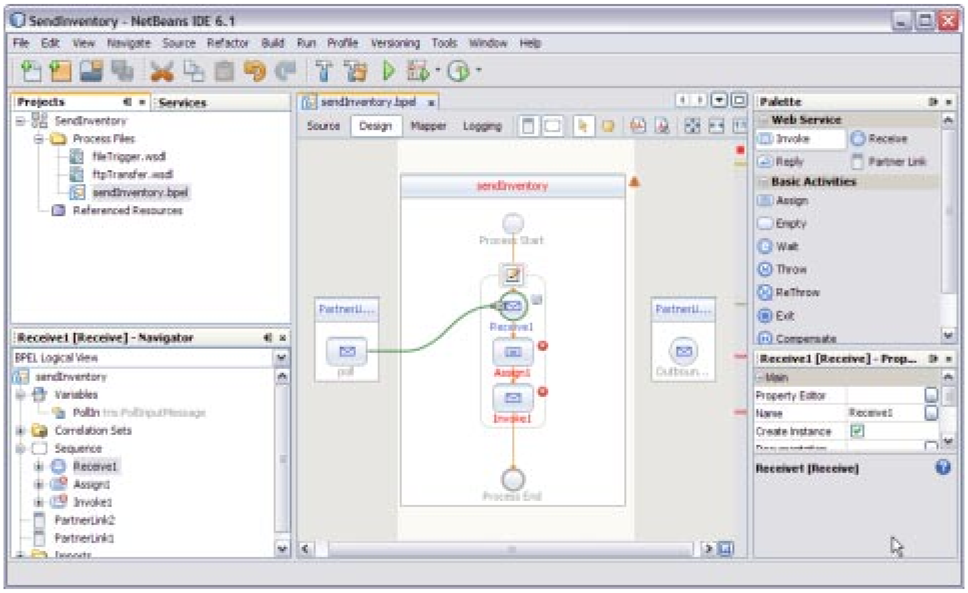
Operation: poll

Input Variable: PollIn

✓ Create Instance

4 **Click OK to close the Receive1 [Receive] - Property Editor.**

**5 Click Save All.**



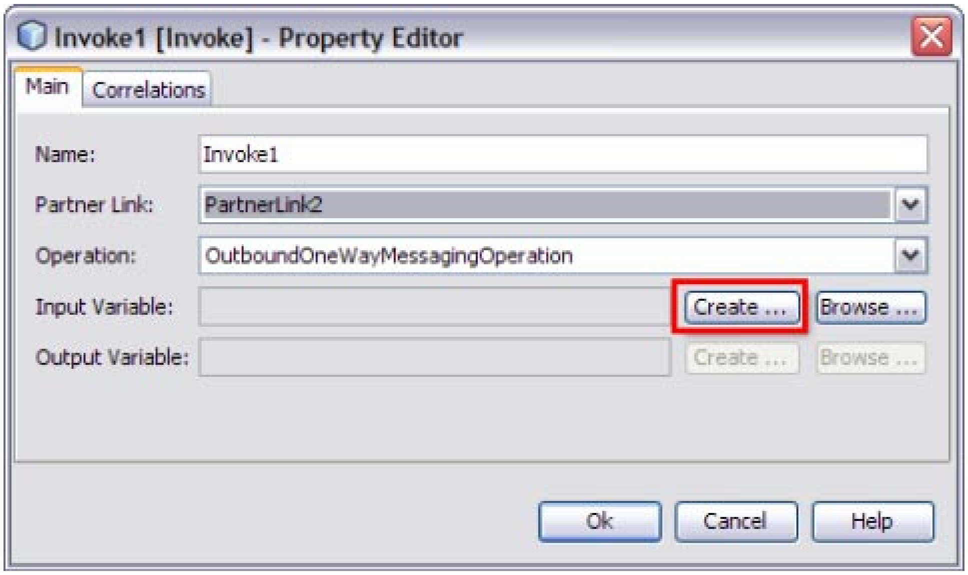
## ▼ To Edit the Web Service : Invoke1

**1 Click Web Service — Invoke1 and click Edit.**

The Invoke1 [Invoke] - Property Editor appears.

2 **Select the properties from the Main tab. In the current example, select PartnerLink2 from the drop-down list.**

The IDE populates OutboundOneWayMessagingOperation against the Operation field .


Invoke1 [Invoke] - Property Editor

Main | Correlations

Name: Invoke1

Partner Link: PartnerLink2

Operation: OutboundOneWayMessagingOperation

Input Variable:

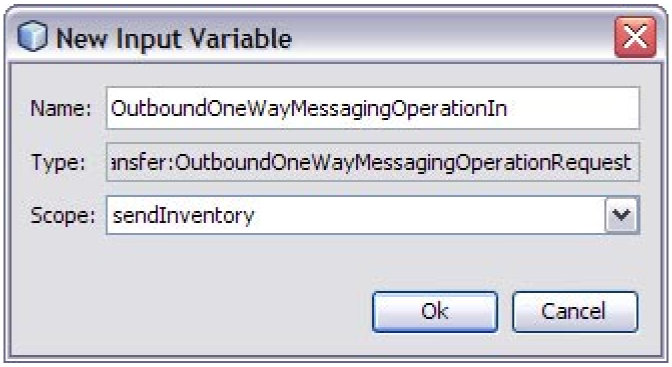
Output Variable:

3 **Do the following to create a New Input Variable:**

- Click the Create button next to the Input Variable field.

  The New Input Variable dialog box appears.
- The default values assigned to the Name, Type and Scope fields are populated for the variable.

  The value in the Name field can be changed.
- Click OK.

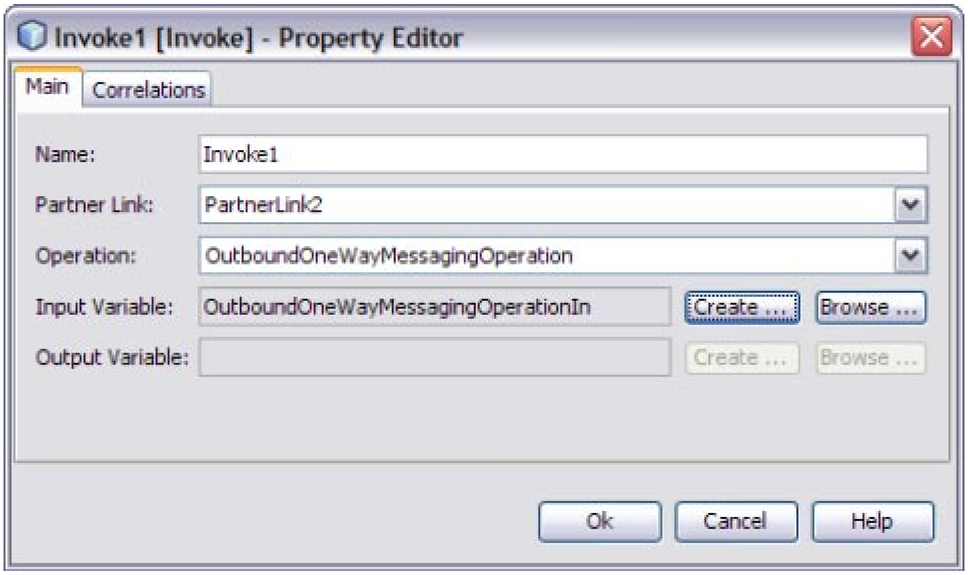

New Input Variable

Name: OutboundOneWayMessagingOperationIn

Type: ansfer:OutboundOneWayMessagingOperationRequest

Scope: sendInventory

The Invoke1 [Invoke] — Property Editor is displayed as shown.


Invoke1 [Invoke] - Property Editor

Name: Invoke1

Partner Link: PartnerLink2

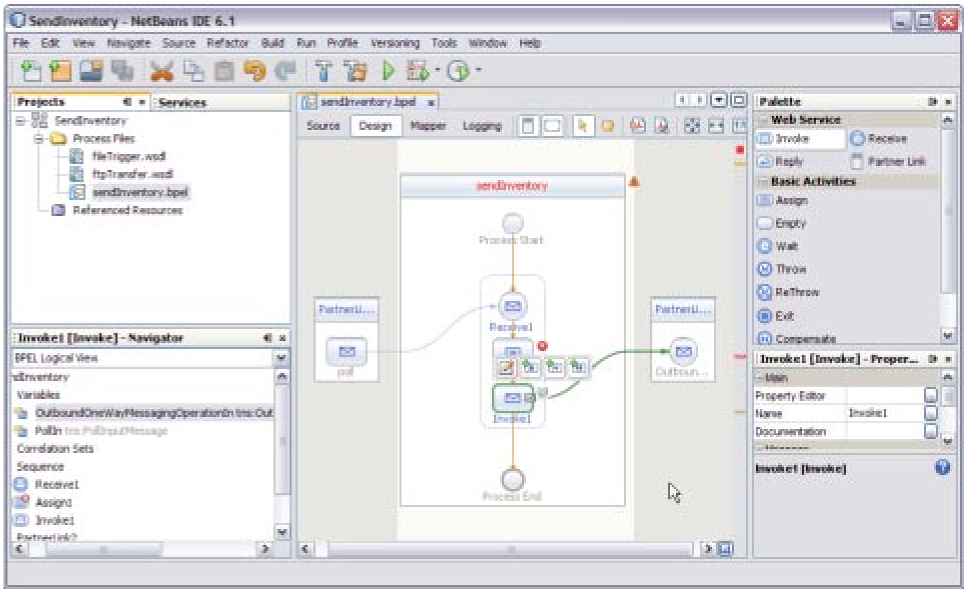
Operation: OutboundOneWayMessagingOperation

Input Variable: OutboundOneWayMessagingOperationIn

Output Variable:

**4 Click OK to close the Receive1 [Receive] - Property Editor.**

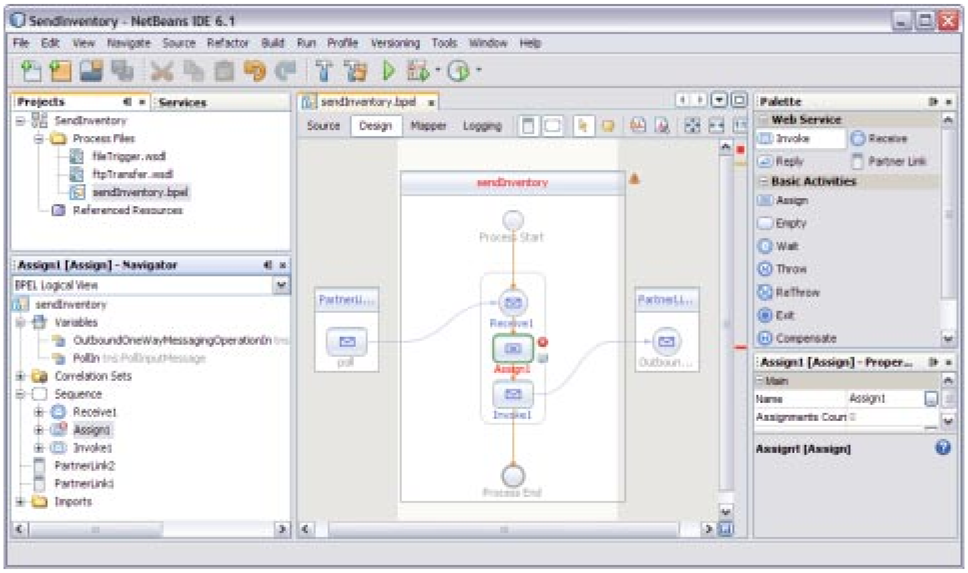
**5 Click Save All.**



## ▼ To Edit the Basic Activities : Assign1

1 **Double-click the Basic Activity : Assign1.**

This displays the BPEL Mapper window.



2 **Expand the node in the Source tree pane (the left pane) of the BPEL Mapper under Output —> Variables (for example, PollIn).**

A part1 node appears under the PollIn node.

3 **Expand the node in the Destination tree pane (the right pane) of the BPEL Mapper under Input —> Variables (for example, OutboundOneWayMessagingOperationIn).**

A part1 node appears under the OutboundOneWayMessagingOperationIn node.

4 **Select the node in the Source tree pane.**

For example, PollIn — part1.

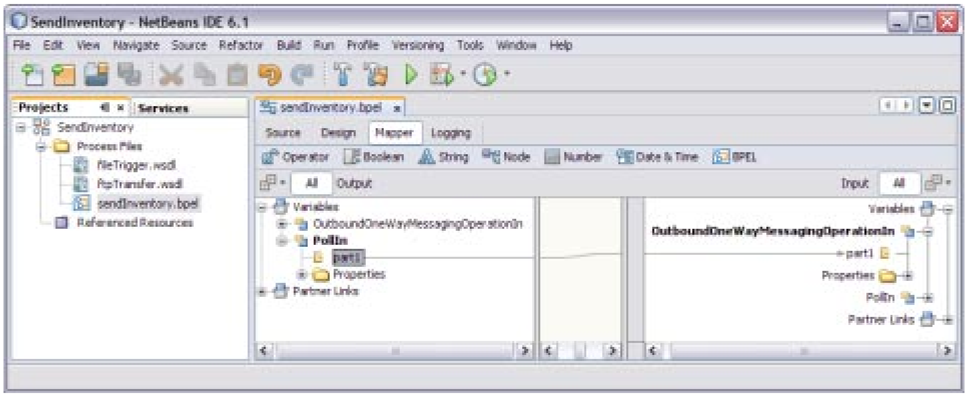
5 **Drag the selection and map it to the node in the Destination tree pane.**

For example, OutboundOneWayMessagingOperationIn — part1.

6 **Map the following Input and Output Variables**

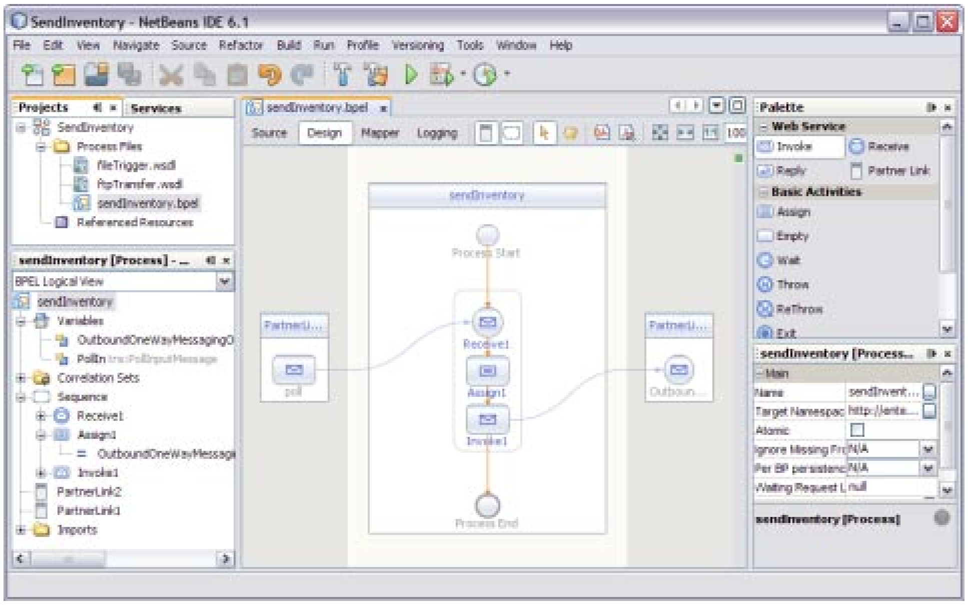
**part1 — part1**

7 **Click Save All.**


SendInventory - NetBeans IDE 6.1
File Edit View Navigate Source Refactor Build Run Profile Versioning Tools Window Help
Projects
Services
SendInventory
Process Files
fileTrigger.wsdl
ftpTransfer.wsdl
sendInventory.bpel
Referenced Resources
sendInventory.bpel
Source Design Mapper Logging
Operator Boolean String Node Number Date & Time BPEL
All Output
Input All
Variables
OutboundOneWayMessagingOperationIn
PollIn
part1
Properties
Partner Links

8 **Click the Design tab.**

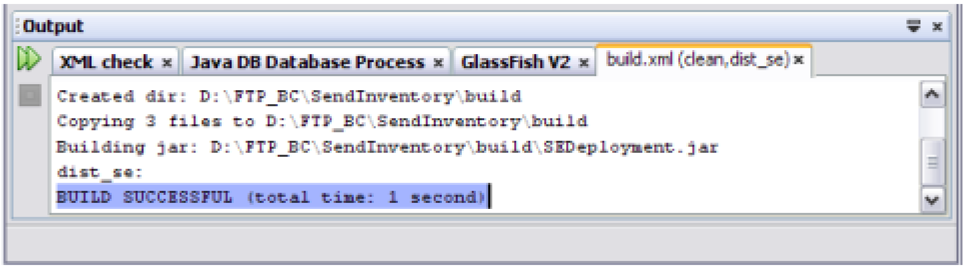
The final output is as shown in the illustration.



9 **Right-click the sendInventory project and select Clean and Build.**

The following message appears after the build.

```
BUILD SUCCESSFUL (total time: 1 seconds).
```



```
Created dir: D:\FTP_BC\SendInventory\build
Copying 3 files to D:\FTP_BC\SendInventory\build
Building jar: D:\FTP_BC\SendInventory\build\SEDeployment.jar
dist_se:
BUILD SUCCESSFUL (total time: 1 second)
```

**10 Click Save All.**

## Validating BPEL

The BPEL Designer has a built-in BPEL code validation functionality that helps create well-formed, valid and standard-compliant code. The code is checked for errors and the user is notified, if validation fails.

### ▼ To Invoke Explicit Validation

Perform any one of the following to invoke explicit validation.

**1 In the Source view, right-click in the source code and choose Validate XML (Alt-Shift-F9) from the menu that appears.**

**2 In the Design view, click the Validate XML button (Alt-Shift-F9) on the Editor toolbar.**

## Design View : Notifications

The validation errors are notified to the user on the Output window, Design window and the Navigator.

### The Design View

The Design view shows the results of both real-time and explicit validation in call-out windows on the diagram and the error stripe.

In the illustration,

---

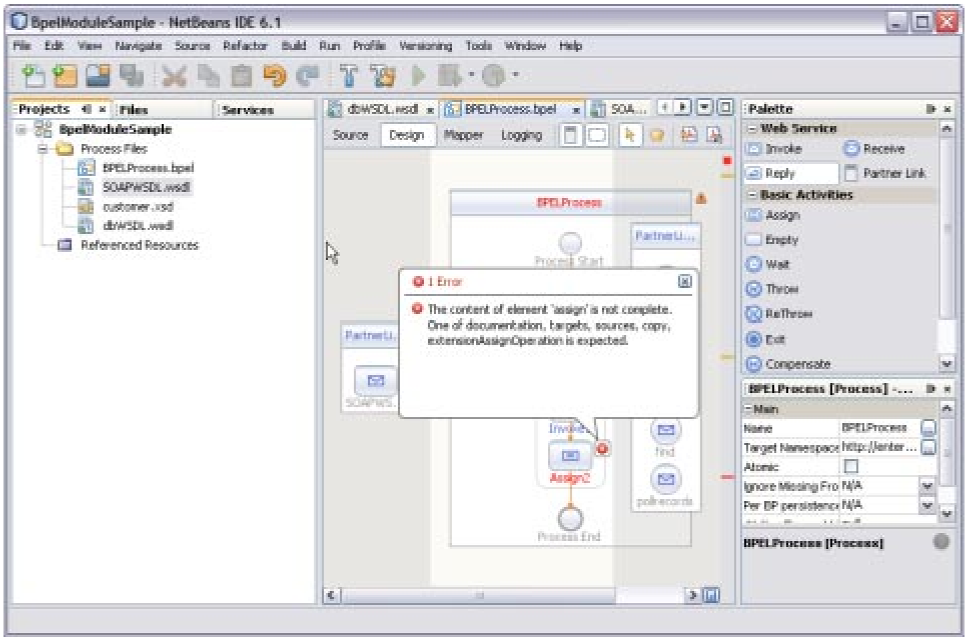
**Note –** A red cross next to an element means that the element has not passed validation and the output contains errors.

A yellow triangle with an exclamation mark means that the element has not passed validation and the output contains warnings.

A red cross in the Design view means there are both errors and warnings.

---

If you click the cross or the triangle, a call-out window appears with a list of errors and warnings.

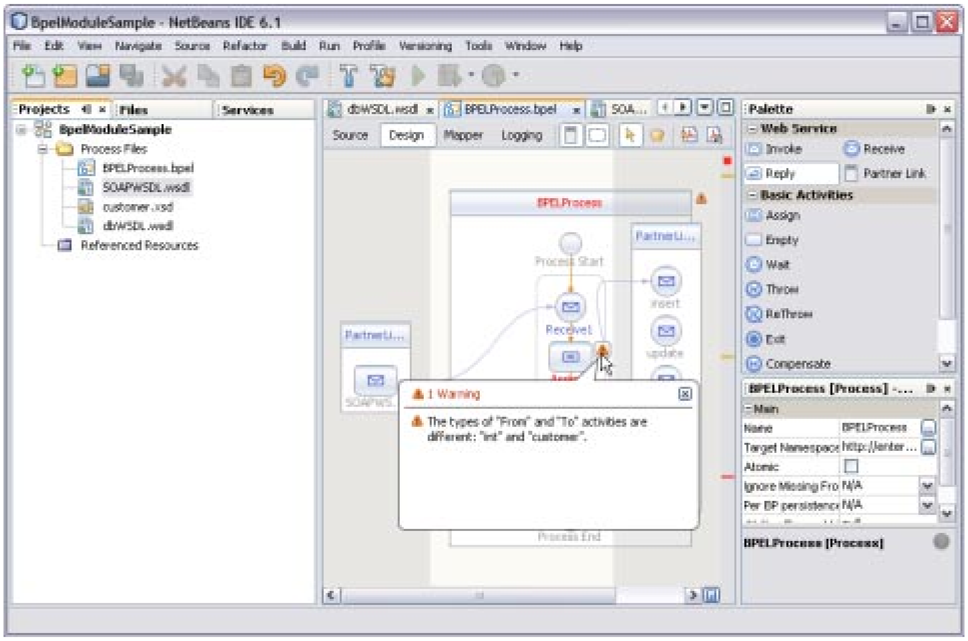

BpelModuleSample - NetBeans IDE 6.1
File Edit View Navigate Source Refactor Build Run Profile Versioning Tools Window Help
Projects Files Services
BpelModuleSample
Process Files
BPELProcess.bpel
SOAPWSDL.wsdl
customer.xsd
dbWSDL.wsdl
Referenced Resources
dbWSDL.wsdl BPELProcess.bpel SOA...
Source Design Mapper Logging
BPELProcess
Process Start
Process End
Assign2
Palette
Web Service
Invoke Receive Reply Partner Link
Basic Activities
Assign Empty Wait Throw ReThrow Exit Compensate
BPELProcess [Process]
Main
Name BPELProcess
Target Namespace http://enter...
Atomic
Ignore Missing Fro N/A
Per BP persistenc N/A

1 Error

The content of element 'assign' is not complete. One of documentation, targets, sources, copy, extensionAssignOperation is expected.

The call–out window includes messages related to validation in accordance with all the criteria listed above. Messages related to the real-time validation are constantly updated.

In the Design view an error stripe displays validation results. An error stripe is a strip next to the right of the scroll bar that contains red marks if some elements have not passed validation. The error stripe represents the entire diagram and not just the portion that is on display. You can immediately see if your BPEL process contains errors without having to scroll through the entire diagram. You can click a red mark to jump to the element that is causing problems. The small square in the error stripe is green, if no errors are detected.



1 Warning

The types of "From" and "To" activities are different: "int" and "customer".

## Creating a Composite Application

Add the JBI module to the BPEL Module project before deploying the Composite Application. Deploying the project makes the service assembly available to the application server, thus allowing its service units to execute.

### ▼ To Create a Composite Application

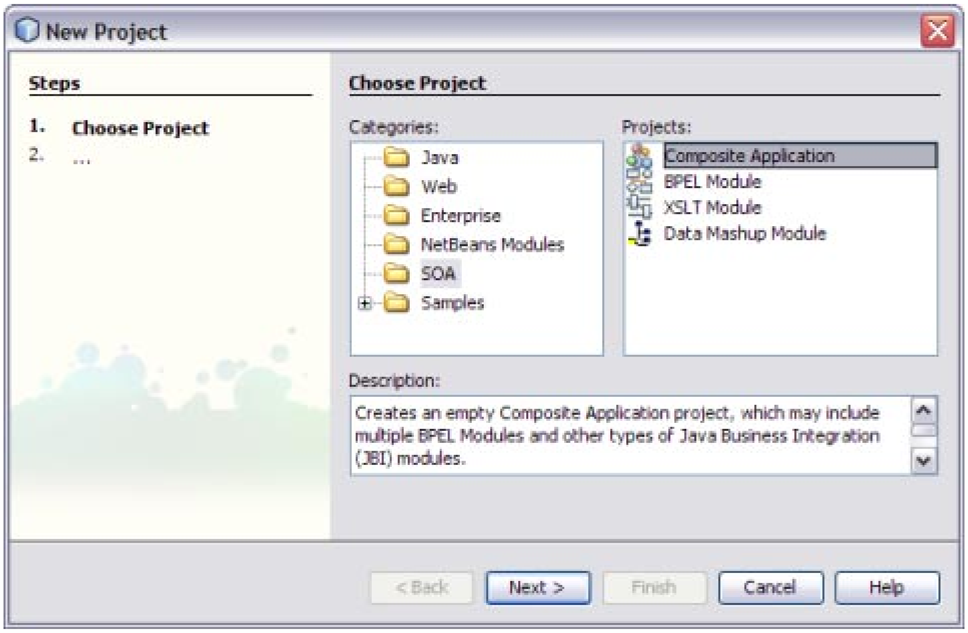
1 **Choose File —> New Project from the main menu.**

The New Project wizard appears.

2 **Select the SOA node from the Categories list.**

3 **Select the Composite Application node from the Projects list.**

4 **Click Next.**

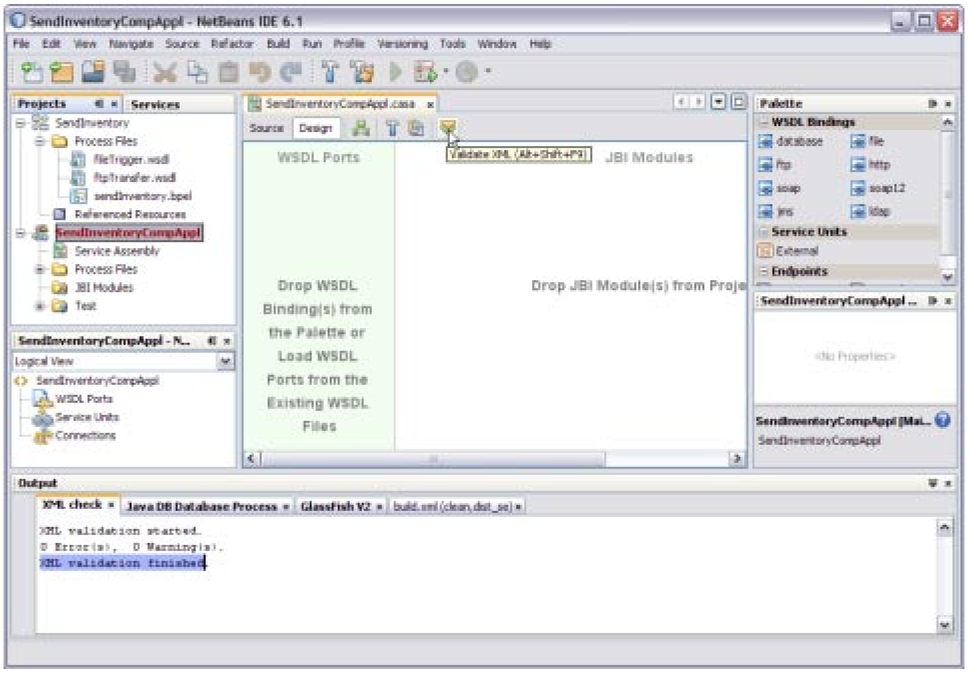


Creates an empty Composite Application project, which may include multiple BPEL Modules and other types of Java Business Integration (JBI) modules.

5 **Type `SendInventoryCompAppl` in the Project Name field.**

6 **Click Finish.**

The Projects window now contains a project node for a Composite Application project called SendInventoryCompAppl.



```
XML validation started.
0 Error(s),  0 Warning(s).
XML validation finished.
```

This action displays a message in the Output console.

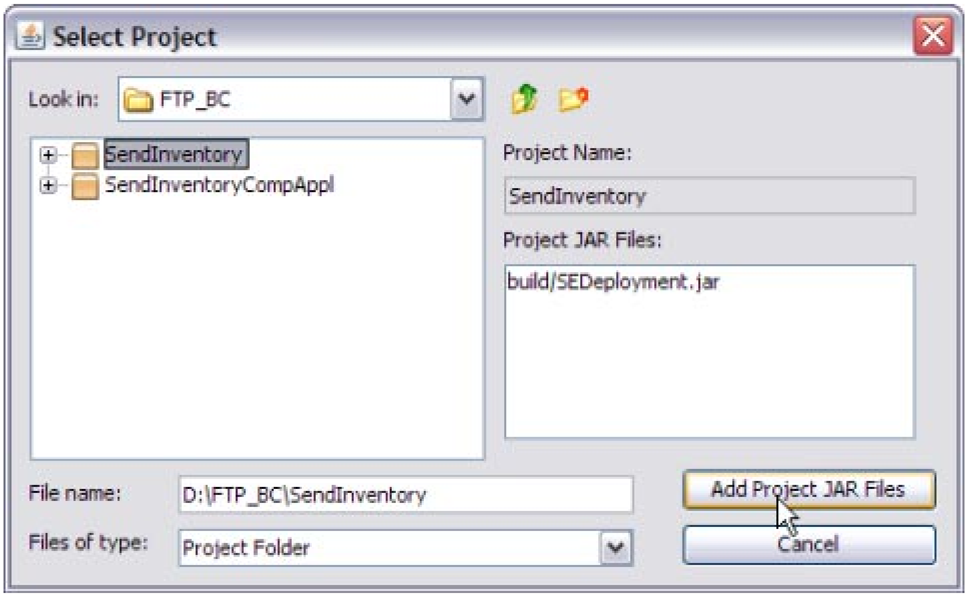
```
XML validation finished
```

7 **Either right-click the SendInventoryCompAppl Composite Application Project node or expand the node. Select JBI Modules.**

8 **Select Add JBI Module.**

**9 Select the Project. Click Add Project JAR Files.**

The Project JAR Files is **build/SEDeployment.jar**.



Select Project

Look in: FTP_BC

SendInventory
SendInventoryCompAppl

Project Name: SendInventory

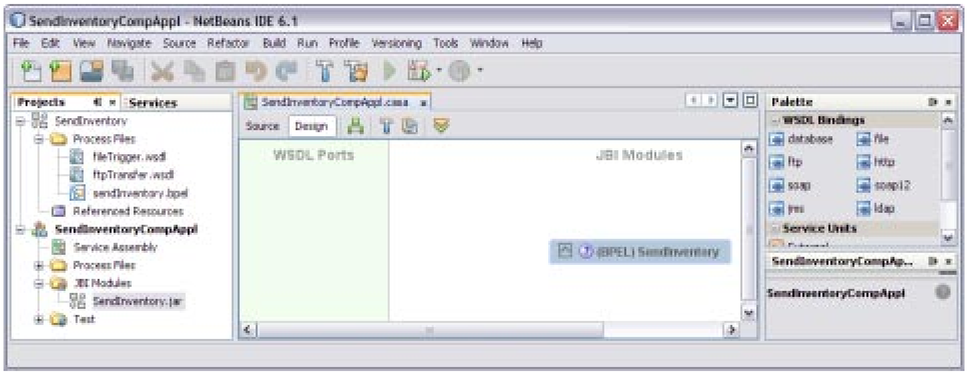
Project JAR Files: build/SEDeployment.jar

File name: D:\FTP_BC\SendInventory

Files of type: Project Folder

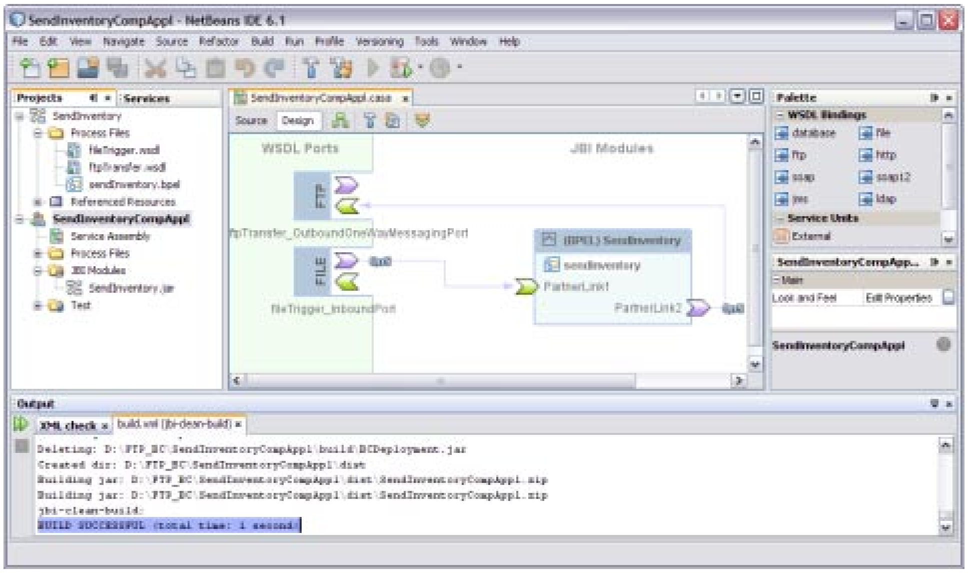
The SendInvetory.jar is added to the project.

**10 Click Save All.**



SendInventoryCompAppl - NetBeans IDE 6.1

**11 Right-click the SendInventoryCompAppl Composite Application project node. Select Clean and Build.**



```
Deleting: D:\FTP_BC\SendInventoryCompAppl\build\BCDeployment.jar
Created dir: D:\FTP_BC\SendInventoryCompAppl\dist
Building jar: D:\FTP_BC\SendInventoryCompAppl\dist\SendInventoryCompAppl.zip
Building jar: D:\FTP_BC\SendInventoryCompAppl\dist\SendInventoryCompAppl.zip
jbi-clean-build:
BUILD SUCCESSFUL (total time: 1 second)
```

This action displays a message in the Output console:

```
BUILD SUCCESSFUL (total time: 1 seconds)
```

**12 Click Save All.**

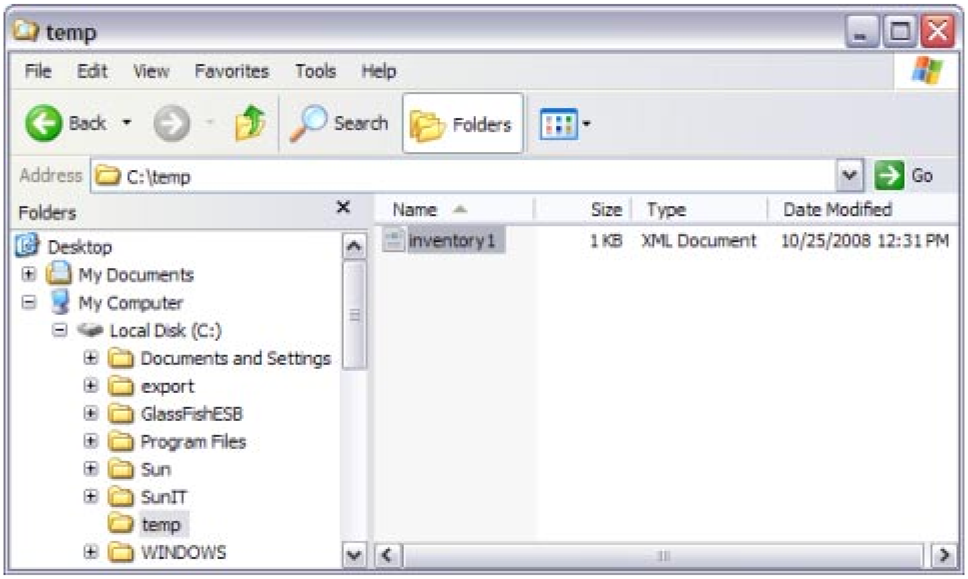
## Deploying the Composite Application

The File Binding Component picks up messages from a local directory. The FTP Binding Component transports messages between the consumer and the provider using FTP.

## ▼ To Deploy the Composite Application

1 **Create a folder and save the file in the local directory.**

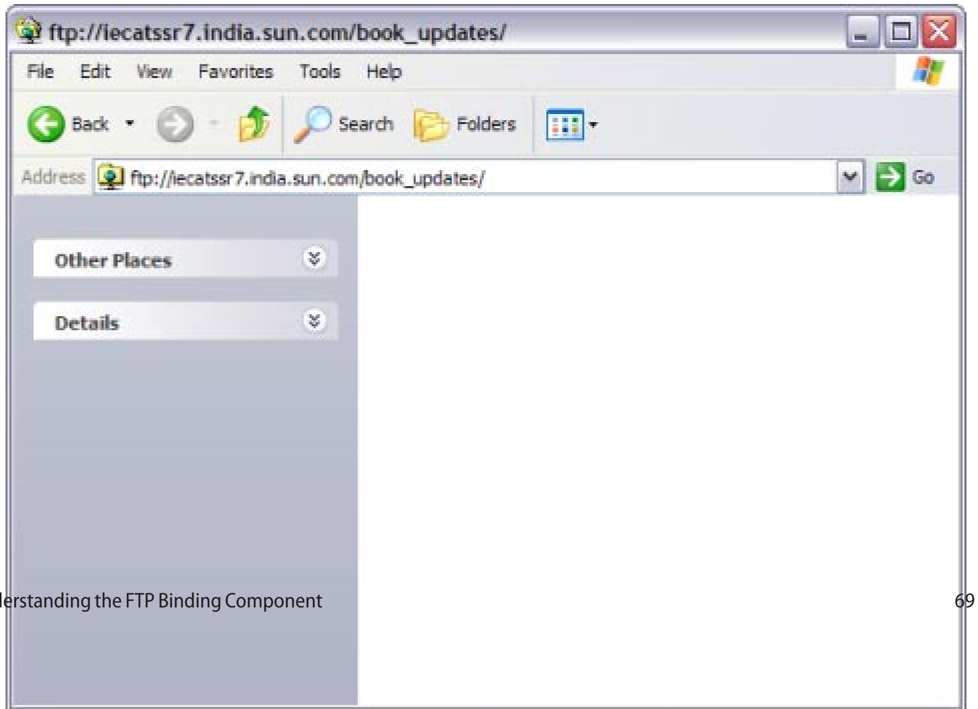
For example, use a folder name of `c:/temp` and file name of `inventory%d.xml`.



| Name | Size | Type | Date Modified |
|---|---|---|---|
| inventory1 | 1 KB | XML Document | 10/25/2008 12:31 PM |

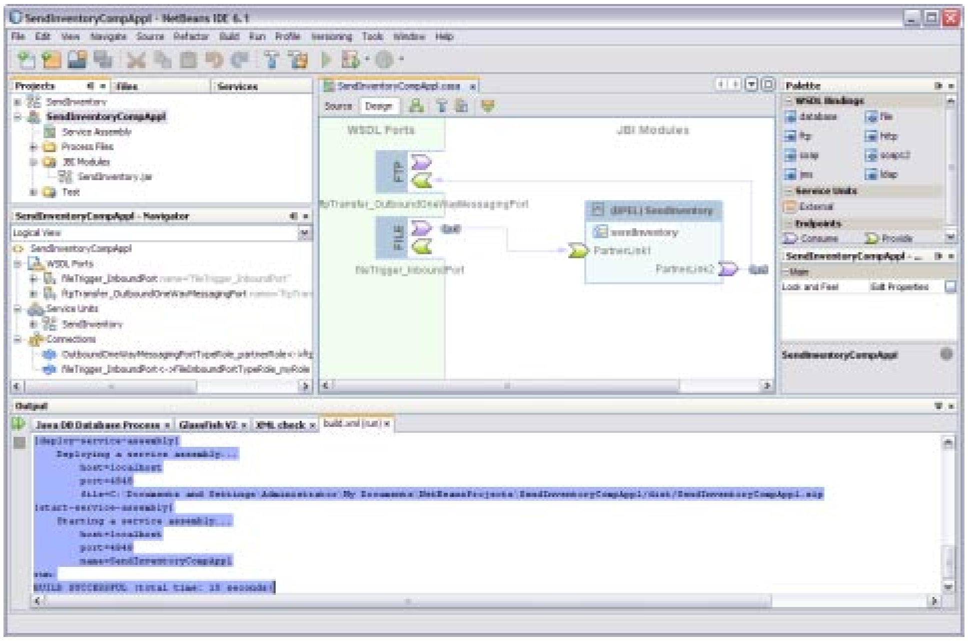
2 **Create a folder on the FTP Server named book_updates.**

The folder is empty before deployment.


ftp://iecatssr7.india.sun.com/book_updates/
File Edit View Favorites Tools Help
Back
Search
Folders
Address ftp://iecatssr7.india.sun.com/book_updates/
Go
Other Places
Details

**3** **Select the SendInventoryCompAppl project node in the Projects window.**

**4** **Right-click and choose Deploy.**



```
[deploy-service-assembly]
    Deploying a service assembly...
        host=localhost
        port=4848
        file=C:\Documents and Settings\Administrator\My Documents\NetBeansProjects\SendInventoryCompAppl/dist/SendInventoryCompAppl.zip
[start-service-assembly]
    Starting a service assembly...
        host=localhost
        port=4848
        name=SendInventoryCompAppl
run:
BUILD SUCCESSFUL (total time: 18 seconds)
```

After deployment of the project, the following message appears in the Output window:

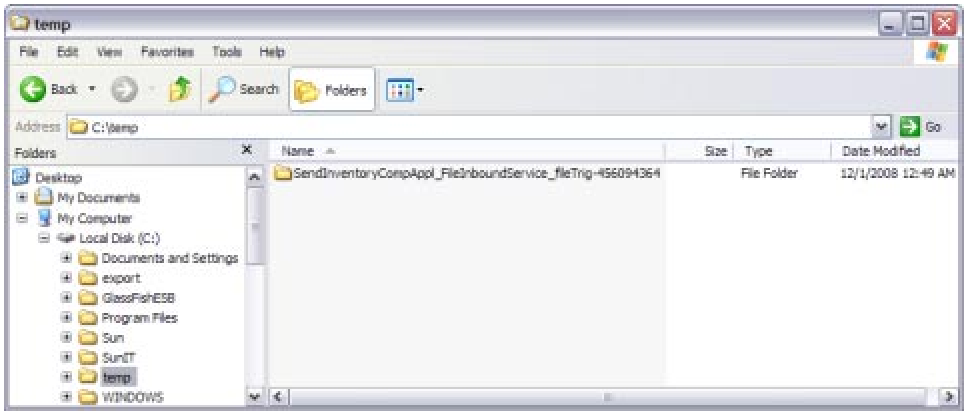
```
BUILD SUCCESSFUL (total time: 18 seconds)
```

**5** **Check for the folder in the local directory.**

Two folders are created under SendInventoryCompAppl_FileInboundService_fileTrig-45609436–4:

**a. archive**

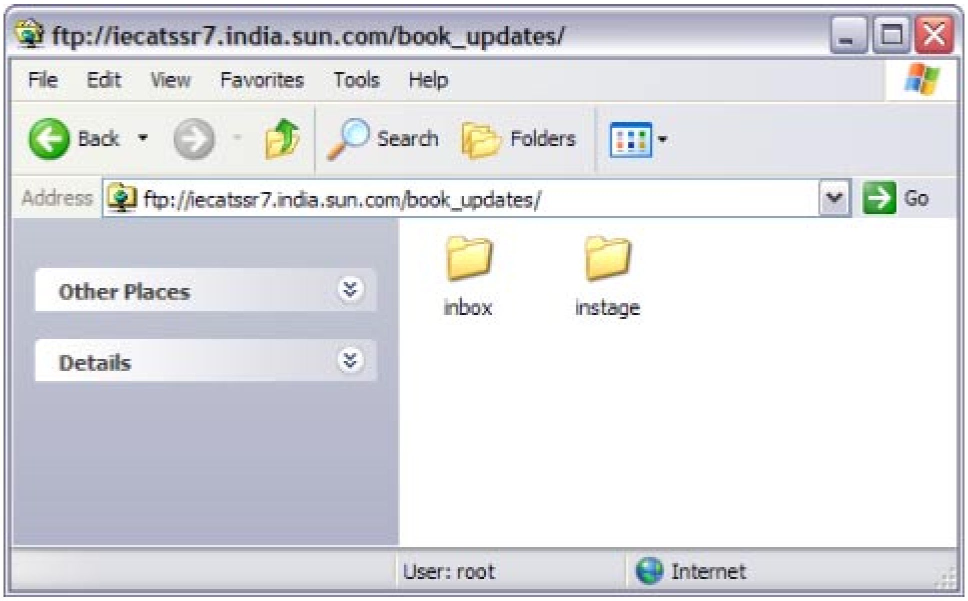
b. **filebc-in processing**



| Name | Size | Type | Date Modified |
|---|---|---|---|
| SendInventoryCompAppl_FileInboundService_fileTrig-456094364 | | File Folder | 12/1/2008 12:49 AM |

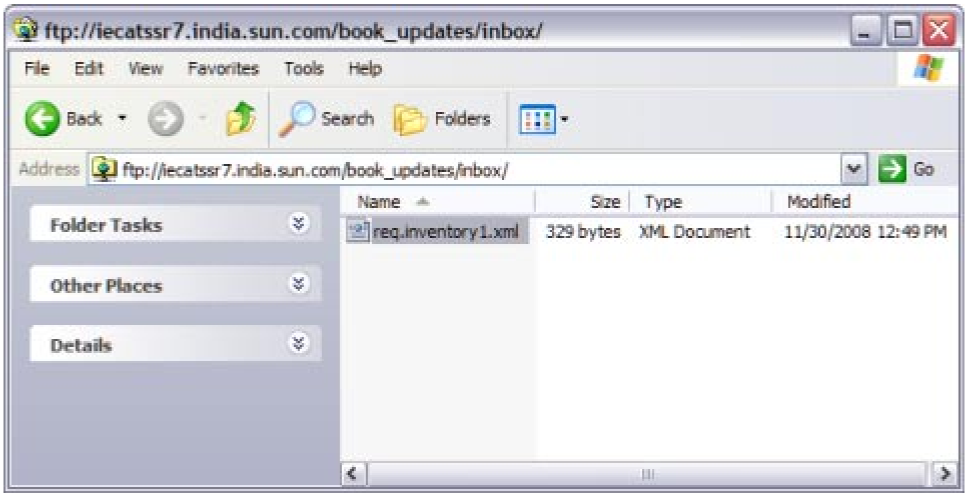
**6 Check the folders on the FTP Server.**

After deployment, inbox and instage folders are created.


ftp://iecatssr7.india.sun.com/book_updates/
File Edit View Favorites Tools Help
Back Search Folders
Address ftp://iecatssr7.india.sun.com/book_updates/ Go
Other Places
Details
inbox
instage
User: root
Internet

**7 Double-click the folder /inbox to check the output.**

The message routing starts with the consumer invoking a service (INVOKE in BPEL script). On the other side of the NMR, FTP BC OutboundProcessor accepts the request message, de-normalizes the message and labels the message body (which is the payload to a FTP file) with name as req.


ftp://iecatssr7.india.sun.com/book_updates/inbox/

| Name | Size | Type | Modified |
|---|---|---|---|
| req.inventory1.xml | 329 bytes | XML Document | 11/30/2008 12:49 PM |

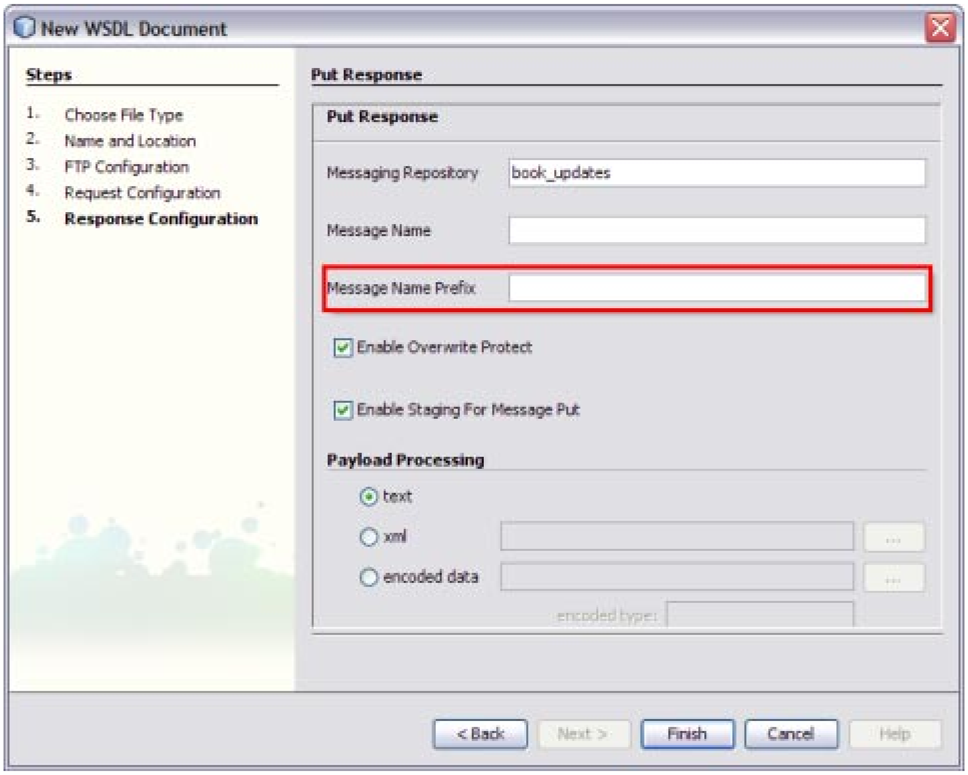
## Working With Various Binding Types

This topic explains the functional behavior of various binding types.

1. **Poll Request Message and Put Response**
   a. **Poll Request**
      - **Message Name Prefix**: Prefix for inbound (IB) message name.
      - **Poll Interval in milli-seconds**: Polling interval in milliseconds when message is polled from a remote target.

   b. **Put Response**

      **Message Name Prefix**: Prefix for outbound (OB) message name.



Put Response

Messaging Repository: book_updates

Message Name:

Message Name Prefix:

[✓] Enable Overwrite Protect

[✓] Enable Staging For Message Put

Payload Processing

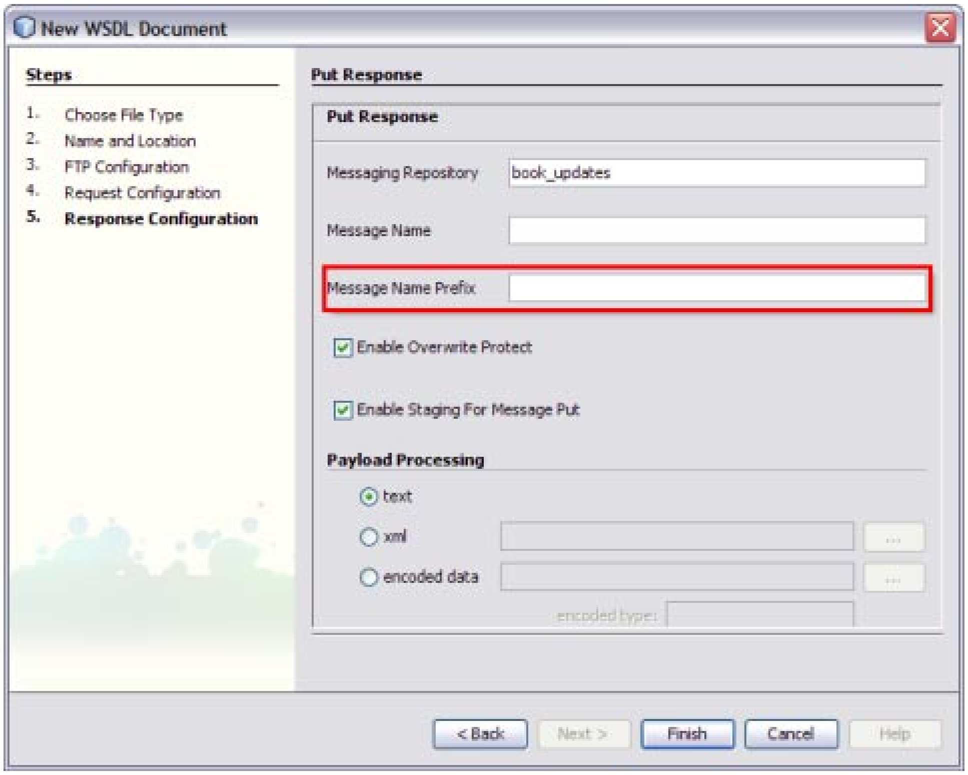
[● text] ○ xml ○ encoded data

encoded type:

2. **Put Request Message and Poll Response**
   a. **Put Request**

      **Message Name Prefix**: Prefix for outbound (OB) message name.

   b. **Poll Response**
      - **Message Name Prefix**: Prefix for inbound (IB) message name.
      - **Poll Interval in milli-seconds**: Polling interval in milliseconds when message is polled from a remote target.


New WSDL Document

Steps

1. Choose File Type
2. Name and Location
3. FTP Configuration
4. Request Configuration
5. **Response Configuration**

Put Response

Put Response

Messaging Repository: book_updates

Message Name:

Message Name Prefix:

[✓] Enable Overwrite Protect

[✓] Enable Staging For Message Put

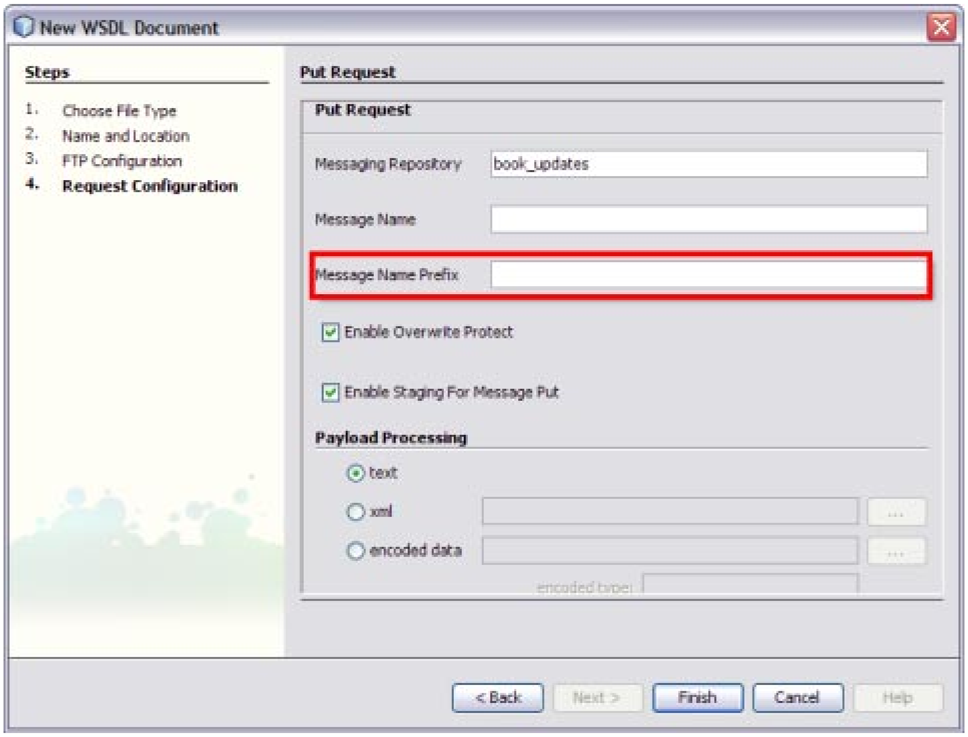
Payload Processing

[● text] ○ xml ○ encoded data

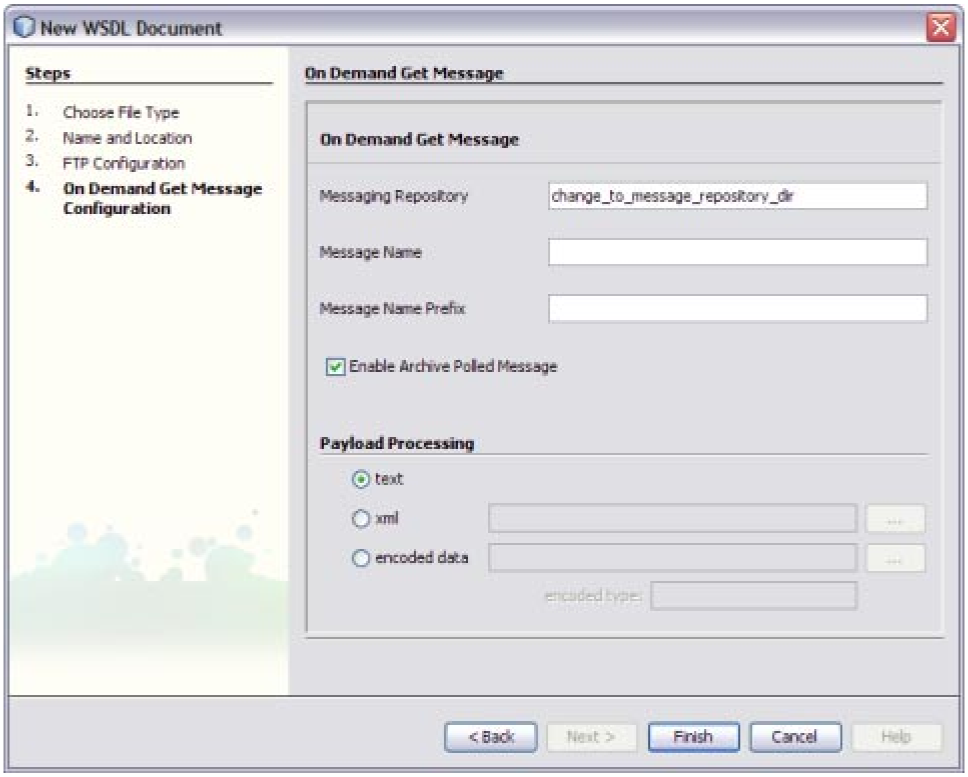
encoded type:

3. **Put Request Message**

   **Message Name Prefix**: Prefix for outbound (OB) message name.



4. **On Demand Get Message**
   a. **Message Name Prefix**: Prefix for inbound (IB) message name.
   b. **Enable Archive Polled Message**: Indicates if archive is required for processed message. If true, processed message is archived, otherwise, it is removed.



On Demand Get Message

Messaging Repository: change_to_message_repository_dir

Message Name:

Message Name Prefix:

[✓] Enable Archive Polled Message

Payload Processing

[● text] ○ xml ○ encoded data

encoded type:

5. **Receive Request**

   **Poll Request**

   a. **Receive Source (From)**: Path pointing to a file on remote FTP server where the transferred data will be read (receiveFrom), the path components could be literals or regular expressions.

   b. **Receive Source (From)**: Has Regular Expressions. Indicates if 'receiveFrom' has regular expressions. When 'receiveFrom' contains regular expressions, these are used as filters to filter out those directory/file entries that match the corresponding regular expressions.

      For example, if 'receiveFromHasRegexs' = FTP_IN_BOX/archive200[1-6]/invoice_[0-1][1-9].bak

      At runtime, FTP BC gets a directory listing from FTP_IN_BOX, iterate through each one of them and finds the first match for regular expression 'archive200[1-6]'.

      For example, archive2001, get a directory listing from FTP_IN_BOX/archive2001, iterate through each one of them and find the first match for regular expression 'invoice_[0-1][1-9].bak', say, invoice_01.bak, now FTP_IN_BOX/archive2001/invoice_01.bak is found as the first match, and it will be used as the resolved value for 'receiveFrom', otherwise, if no match found for regular expression 'invoice_[0-1][1-9].bak', FTP BC will go back to the parent level, and try the next match of 'archive200[1-6]', and repeat the above process until found a path matching all the regular expressions as corresponding path components or no matching path found after exhausted all paths under FTP_IN_BOX.

c. **Pre Receive Operation (Command)**: Operation performed before receiving starts.
   - NONE - No operation is performed before receiving starts.
   - COPY - Make a copy of the target file (specified by 'receiveFrom') to a file specified by 'preReceiveLocation' before receiving starts.
   - RENAME - Move the target file (specified by 'receiveFrom') to a file specified by 'preReceiveLocation' before receiving starts.

d. **Pre Receive Operation Location**: Destination file for operation to be performed before receiving starts.

e. **Pre Receive Location Has Patterns**: Indicate if 'preReceiveLocation' contains patterns, where 'pattern' is a string containing special characters escaped by percentage sign, the following are all the symbols supported:
   i. directory/file name replacement (%p/%f), usually used in pre/post operation's 'receiveFrom'/'sendTo' path.

      For example, when 'sendTo' is my_in_box/invoice.dat, then a pattern like %p_backup/%f.bak will be my_in_box_backup/invoice.dat.bak after expansion.
   ii. UUID %u, will be substituted by a UUID value compliant with Java 1.5 UUID.
   iii. sequence number reference %0, %1, %2, %3, %4, %5, %6, %7, %8, %9, this symbol will be replaced by the current value of sequence number, which is an integer count that increments after each reference.

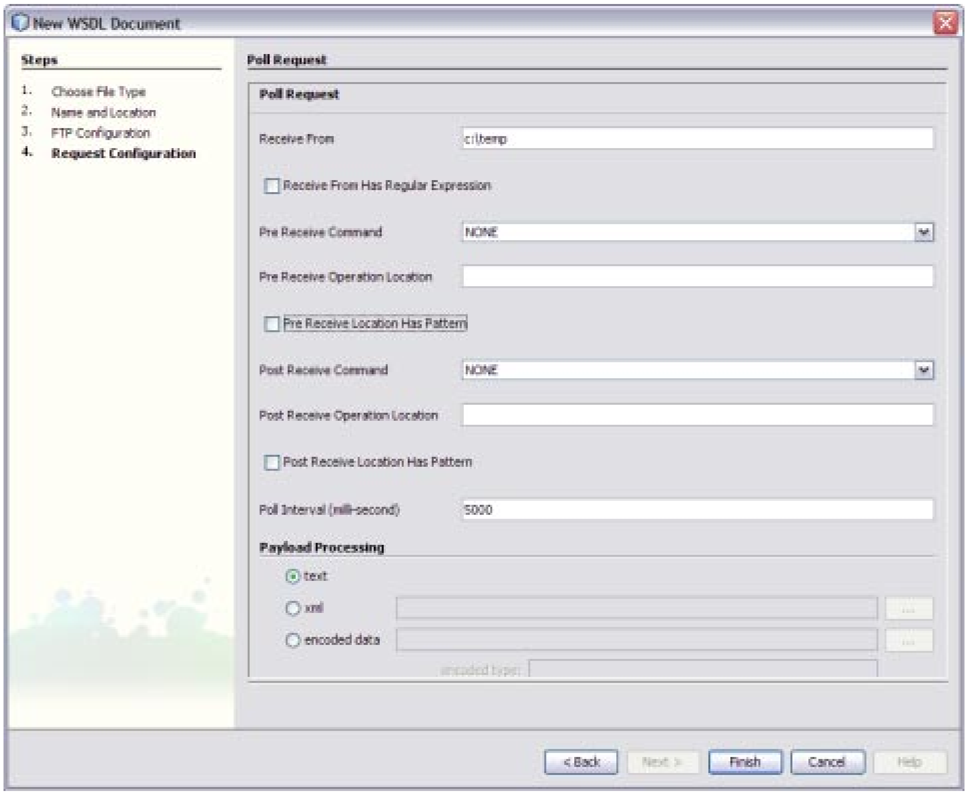
   For Java Timestamp Patterns, see Table 4.

f. **Post Receive Operation (Command)**: Operation performed after receiving completes:
   - NONE - no operation performed after receiving completes.
   - DELETE - delete the target file (specified by 'receiveFrom') after receiving completes.
   - RENAME - move the target file (specified by 'receiveFrom') to a file specified by 'postReceiveLocation' after receiving completes.

g. **Post Receive Operation Location**: Destination file for operation to be performed after receiving completes.

h. **Post Receive Location Has Patterns**: Indicates if 'postReceiveLocation' contains patterns, where 'pattern' is a string containing special characters escaped by percentage sign. The symbols supported are similar to **Pre Receive Location Has Patterns**.

i. **Poll Interval**: Polling interval in milliseconds when data is polled from a location specified by 'receiveFrom'.



Receive From: c:\temp
Receive From Has Regular Expression
Pre Receive Command: NONE
Pre Receive Operation Location:
Pre Receive Location Has Pattern
Post Receive Command: NONE
Post Receive Operation Location:
Post Receive Location Has Pattern
Poll Interval (milli-second): 5000

6. **Receive Request and Send Response**

   **Send Response (Put Response)**

   a. **Send Destination (To)**: Path pointing to a file on remote FTP server, where the transferred data will be stored (sendTo), the path components could be literal or patterns, see 'sendTohasPatterns' for a detailed definition of pattern.

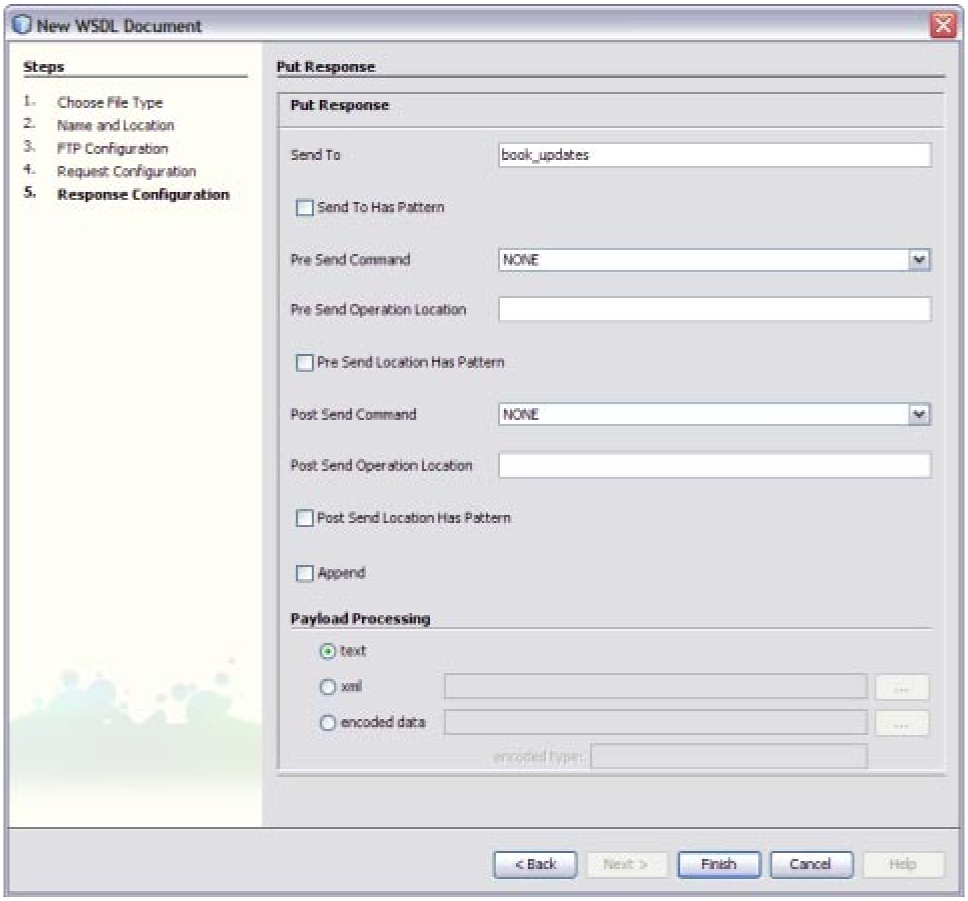
   b. **Send Destination (To)**: Has patterns. See **Pre Receive Location Has Patterns**.

   c. **Pre Send Operation (Command)**: Operation performed before sending starts.

      - NONE - No operation performed before sending starts.
      - COPY - Make a copy of the target file (specified by 'sendTo') to a file specified by 'preSendLocation' before sending starts.
      - RENAME - Move the target file (specified by 'sendTo') to a file specified by 'preSendLocation' before sending starts.

   d. Pre Send Operation Location Destination file for operation to be performed before sending starts.

e. Pre Send Location Has Patterns Indicate if 'preSendLocation' contains patterns, where 'pattern' is a string containing special characters escaped by percentage sign. The supported symbols are similar to **Pre Receive Location Has Patterns**
f. Post Send Operation (Command) Operation performed after sending completes:
   - NONE - no operation performed after sending completes.
   - DELETE - delete the target file (specified by 'sendTo') after sending completes.
   - RENAME - move the target file (specified by 'sendTo') to a file specified by 'postSendLocation' after sending completes.
g. **Post Send Location (Command)**: Destination file for operation to be performed after sending completes.
h. **Post Send Location Has Patterns**: Indicates if 'postSendLocation' contains patterns, where 'pattern' is a string containing special characters escaped by percentage sign. The symbols supported are similar to **Pre Receive Location Has Patterns**.
i. **Append Payload To Target File**: Indicates if the message will be appended at the end of the target file.



Put Response

Put Response

Send To: book_updates

[ ] Send To Has Pattern

Pre Send Command: NONE

Pre Send Operation Location:

[ ] Pre Send Location Has Pattern

Post Send Command: NONE

Post Send Operation Location:

[ ] Post Send Location Has Pattern

[ ] Append

Payload Processing

● text

○ xml

○ encoded data

encoded type:

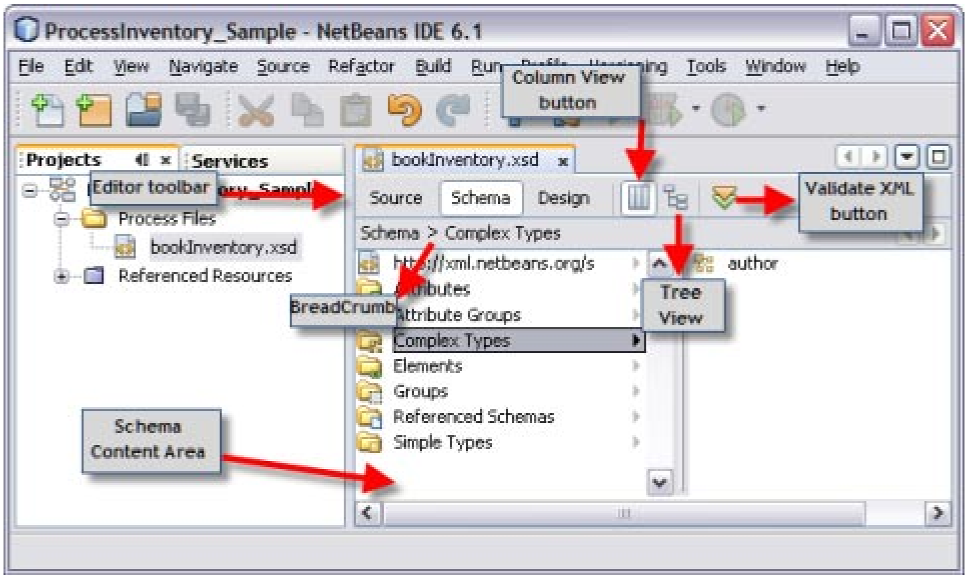
## Exploring the XML Schema

This section illustrates using the Schema view of the XML schema editor and the Navigator window's Schema View to explore a sample schema.

### About the Schema View

The Schema view of the Schema Editor allows you to visualize and scalability edit an XML schema. The Schema view is the view that opens in the Source Editor when you first double-click a schema file (.xsd) node in the Projects window.

The Schema view has the following parts:



1. **Editor Toolbar**: The Editor Toolbar is located at the top of the view, just below the tab for the XML schema file. The Editor toolbar has the following buttons:
   a. **Navigation buttons**: The Source, Schema, and Design buttons let you switch to the views of the XML schema.
   b. **View buttons**: These help you view data in columns or a tree structure.

      The Schema view has two sub-views.

      - The column view
      - The tree view of schema components

        The column view is the default view. Use the column and tree buttons in the editor toolbar to switch between the column and tree view.
   c. **Validate XML button**: Use this button to validate the XML in your schema.
2. **Breadcrumb area**: This area appears immediately below the Editor toolbar when you are using the columns view of the Schema view. Click Breadcrumbs to retrace the steps. The first entry in this area is always labeled "Schema" for the root of the schema. If the entries extends beyond the visible area, the IDE enables the scroll buttons so that you can continue to navigate through the breadcrumbs.
3. **Schema content area**. This area contains the column view or the tree view of the XML schema. The nodes in both views lets you drill down into the schema. Each folder node represents slices of the schema, such as attributes, complex types, and elements. The schema content area comprises the following:

a. **Column View**: In this view, the Schema content area initially contains one column. Each time you select a node that has children, another column is added to the right of the column where you made your selection. The nodes that have child nodes are indicated by a black arrow next to the node in the column. The arrow is light gray if a node does not have child nodes.

b. **Tree View**: In this view, the Schema content area contains one tree view of the XML schema. Expand the nodes to drill down on the schema components.

## Creating the XML Schema

In this section the user adds a new XML schema file and XML schema components to the BPEL Module project.

The XML schema allows you to visualize and edit XML schemas. Using the XML schema, you can reference external schemas and use advanced queries to analyze the schemas.

Oracle Java CAPS comes bundled with a rich set of tools to work with various XML documents such as XML Schema, WSDL, BPEL, and XML instance documents. The tools provide several options to edit and visualize XML documents. In addition it also provides refactoring support, search, queries and find usage, seamless navigation between views, design pattern and schema aware code completion support.

Using the XML schema functionality, you can:

- Rapidly design complex types and elements, across multi-file schemas using the Design view.
- Easily navigate deep schema structures using the Schema view.
- Rapidly create WSDL documents using the WSDL editor.

### ▼ To Create XML Schema

1 **Expand the project node. Right-click either the BPEL Module node or Process Files. Choose New —> Other in the Projects Window.**

The New File wizard opens.

2 **In the New File wizard, perform the following:**

a. **Select the XML node in the Choose File Type page — Categories list. Select the XML Schema node in the File Types list and click Next.**

b. **Type the File Name in the File Name field.**

For example, bookInventory

c. **Click Finish.**

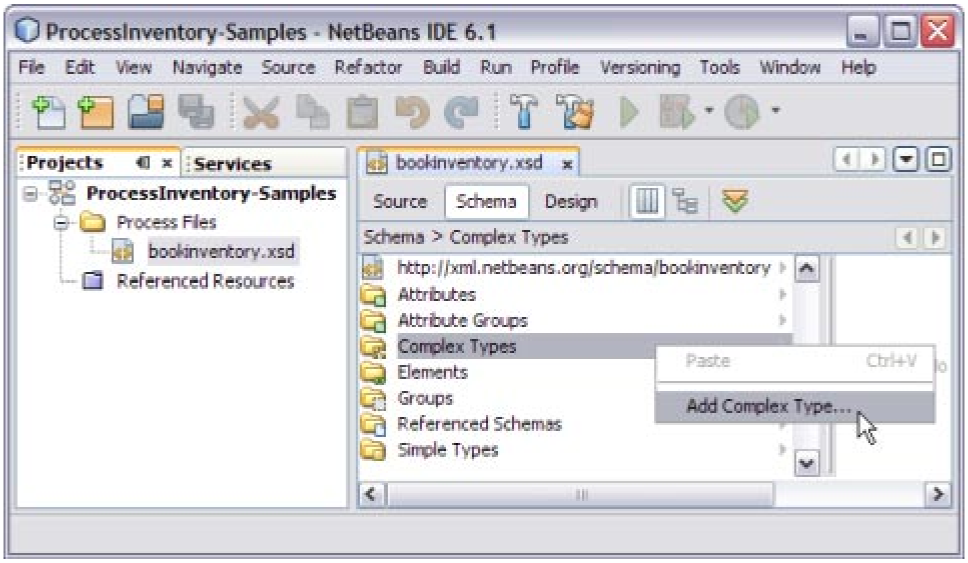
In the Projects window, the Process Files node now contains a subnode labeled bookInventory.xsd. The Source Editor contains a tab for the XML schema file named bookInventory.xsd. The Schema view for this file is also displayed.

3 **Click the Design button to open the Design view of the XML schema editor.**

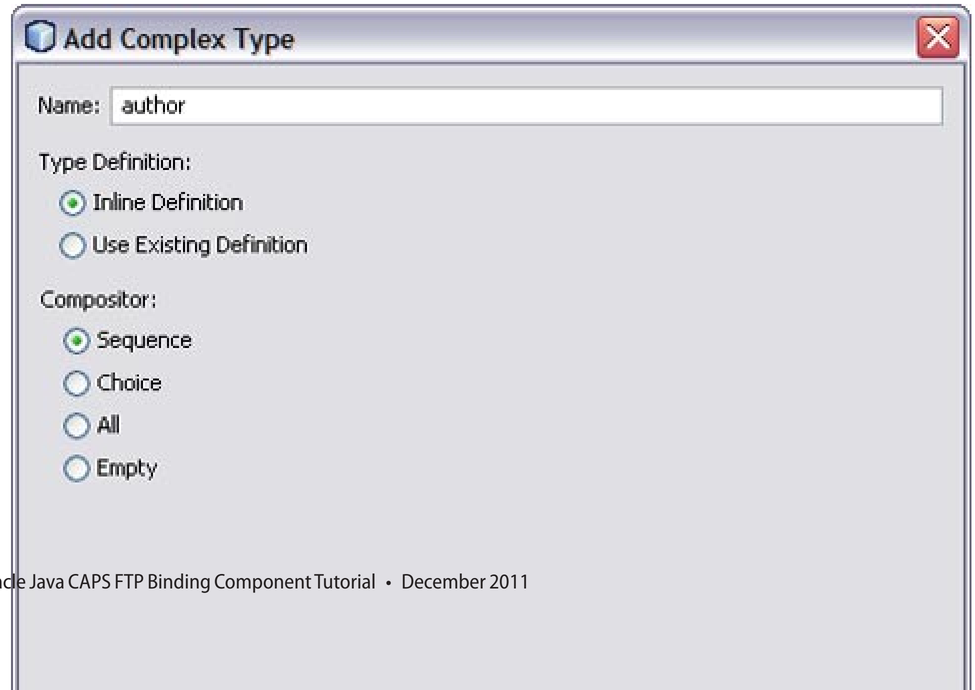
## ▼ To Add a Complex and a Global Complex Type to the XML Schema

**1** **Select the Complex Types node in the first column of the Schema view.**

**2** **Right-click and choose Add Complex Type.**



This opens the Add Complex Type dialog box.



Name: author

Type Definition:

[✓ Inline Definition]
Use Existing Definition

Compositor:

[✓ Sequence]
Choice
All
Empty

3 **Type the name in the Name field.**

For example, author

a. **Select Type Definition: Inline Definition**

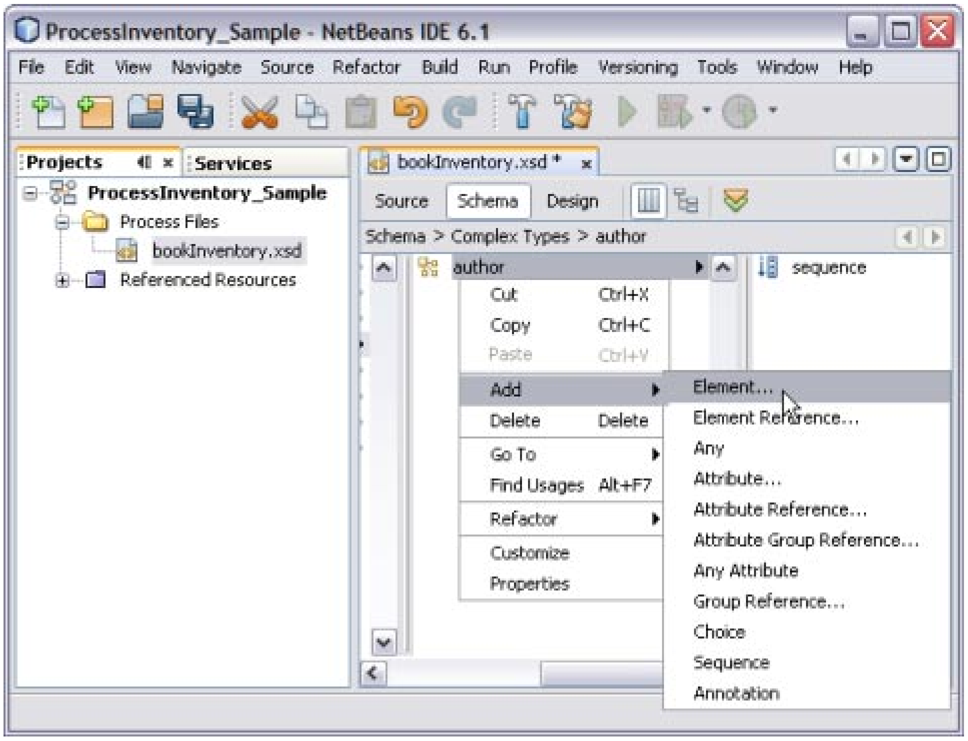
b. **Select Compositor: Sequence**

A preview of the XML code is also displayed at the bottom of the box.

4 **Click OK.**

## ▼ To Add Element to the XML Schema

1 **Select the Complex Types —> author in the Schema view. Right-click on either author or sequence and choose Add —> Element.**

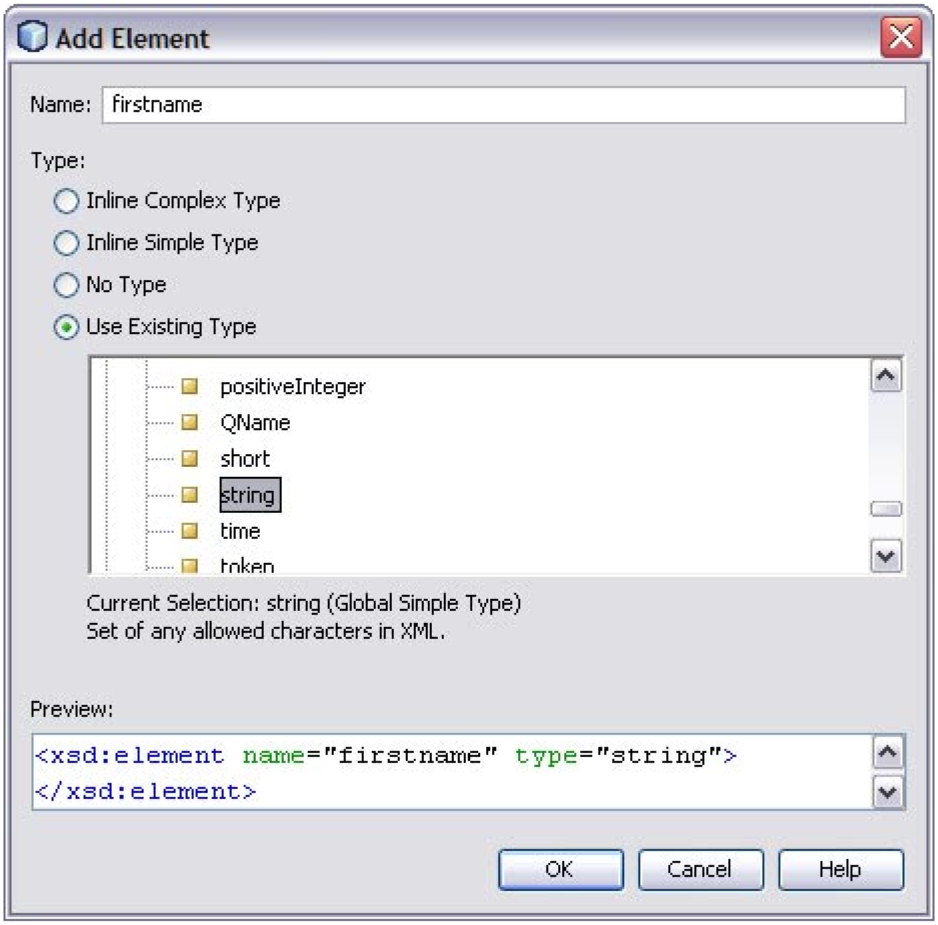


This opens Add Element dialog box.

2 **Type the Name of the Element.**

For example, firstname

3 **Type from the list of radio button options. In the current example, choose the Use Existing Type radio button. In the listing area beneath the Type radio button, expand the Built-in Types node. Select string.**



Name: firstname

Type: [✓ Use Existing Type]

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

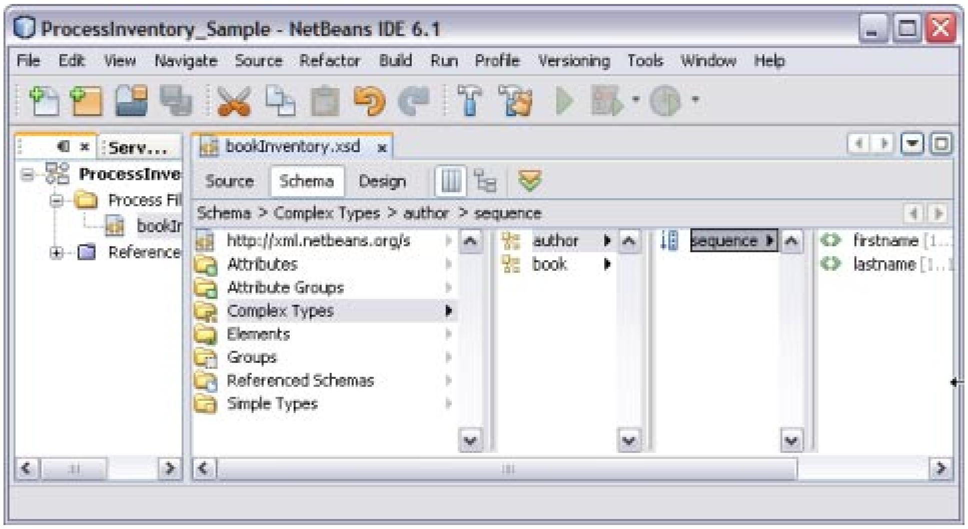
```
<xsd:element name="firstname" type="string">
</xsd:element>
```

4 **Click OK.**

The Schema view now contains a node for the firstname element, whose parent is the sequence under the author Complex Types.

**5 Click Save All.**

Similarly, create another Element — lastname. Repeat steps 1 through 5.



**See Also** Click Complex Types — Add Complex Type.

- Type Name: book

  Click OK.

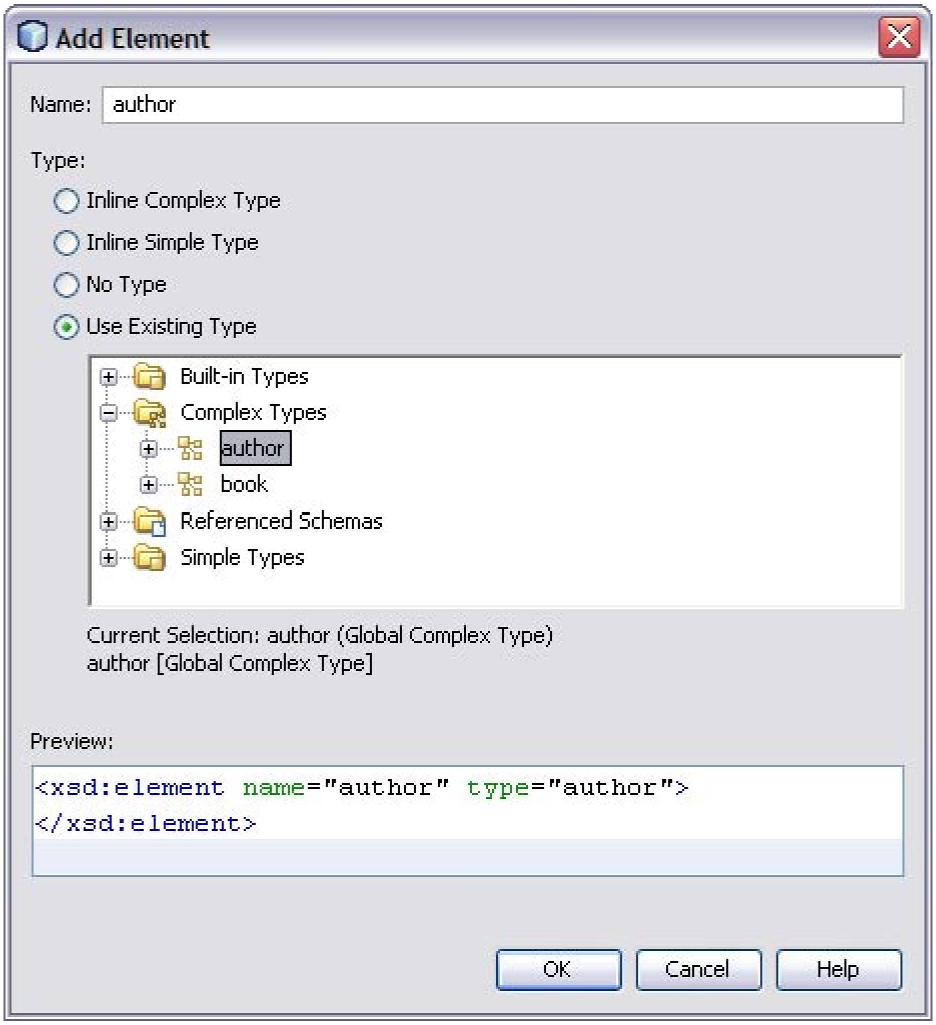
  - Click either Complex Types: book or sequence —> Add —> Element —> genre

    Use Existing Types —> Build-in Types —> string
  - Click either Complex Types: book or sequence —> Add —> Element —> title

    Use Existing Types —> Build-in Types —> string
  - Click either Complex Types: book or sequence —> Add —> Element —> author

    Use Existing Types —> Complex Types —> author

    In the current example, the author is a Global Complex Type because it comprises of two Element Types (firstname and lastname).



Name: author

Type:

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

Current Selection: author (Global Complex Type)
author [Global Complex Type]

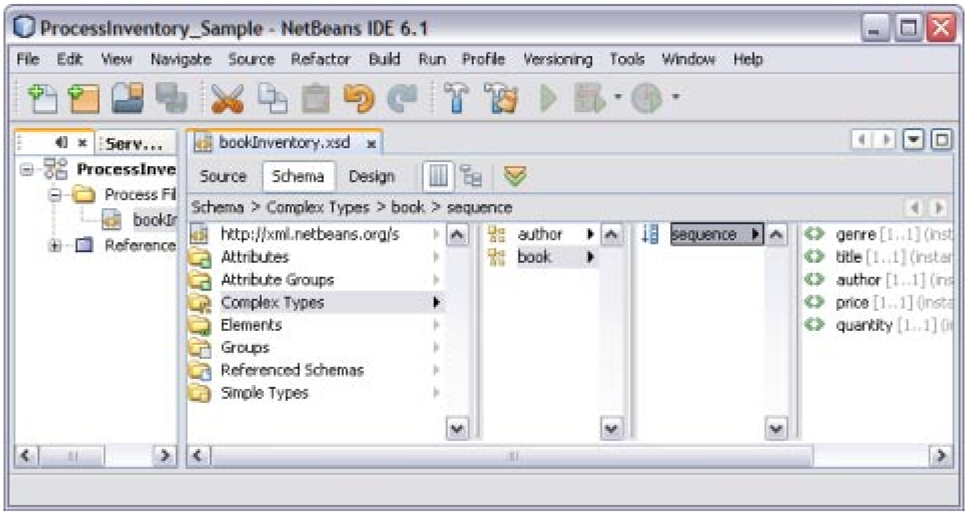
Preview:

```
<xsd:element name="author" type="author">
</xsd:element>
```

- Click either Complex Types: book or sequence —> Add —> Element —> price

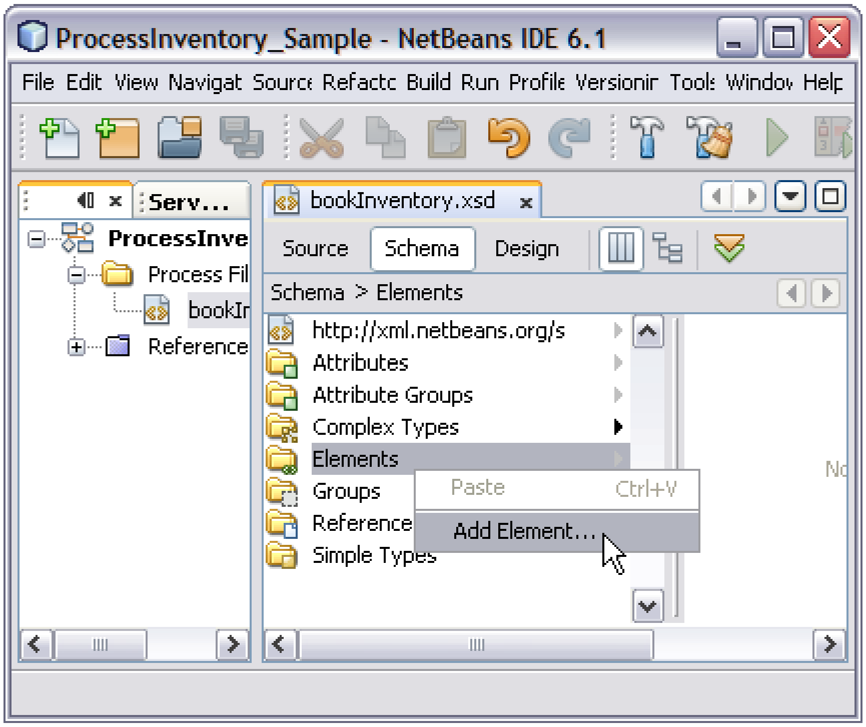
  Use Existing Types — Build-in Types — double
- Click either Complex Types: book or sequence —> Add —> Element —> quantity

  Use Existing Types — Build-in Types — unsignedInt


ProcessInventory_Sample - NetBeans IDE 6.1
File Edit View Navigate Source Refactor Build Run Profile Versioning Tools Window Help
Serv...
ProcessInve
Process Fi
bookIr
Reference
bookInventory.xsd
Source Schema Design
Schema > Complex Types > book > sequence
http://xml.netbeans.org/s
Attributes
Attribute Groups
Complex Types
Elements
Groups
Referenced Schemas
Simple Types
author
book
sequence
genre [1..1] (inst
title [1..1] (instar
author [1..1] (ins
price [1..1] (insta
quantity [1..1] (i

## ▼ To Add Elements to the XML Schema

1. **Select Elements in the first column of the Schema view.**
2. **Right-click and choose Add Element.**



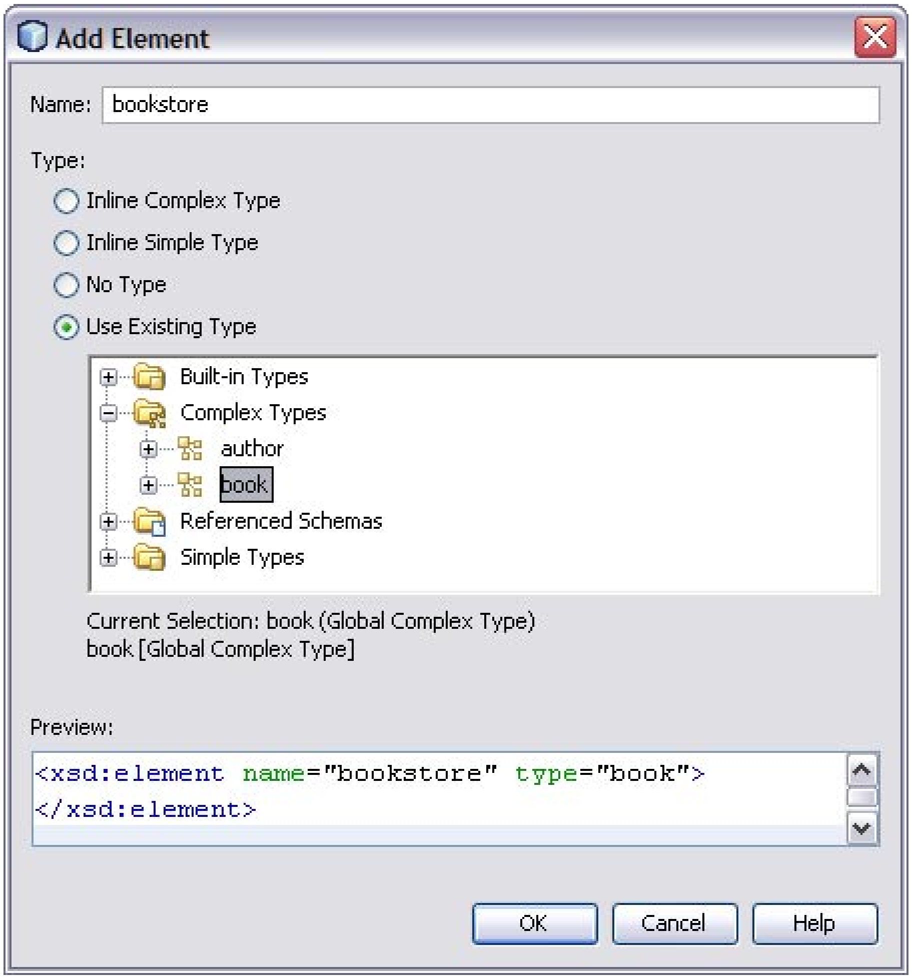
This opens Add Element dialog box

3. **Type the Name of the Element.**

   For example, bookstore

**4 Select the Use Existing Type radio button. In the listing area beneath the Type radio buttons, expand the Complex Types node. Select book.**

For example, book



Name: bookstore

Type:

Inline Complex Type

Inline Simple Type

No Type

[✓ Use Existing Type]

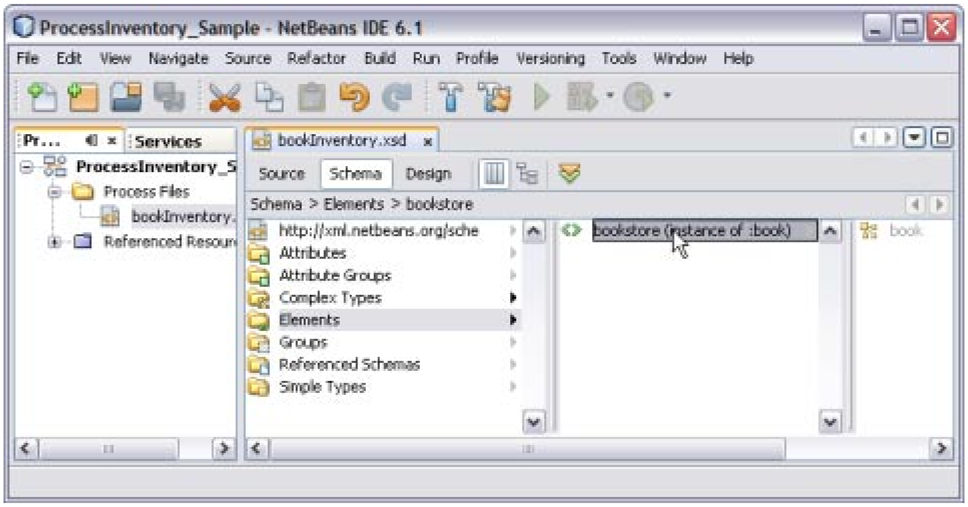
Current Selection: book (Global Complex Type)
book [Global Complex Type]

Preview:

```
<xsd:element name="bookstore" type="book">
</xsd:element>
```

In the current example, book is a Global Complex Type because it comprises of five Element Types (genre, titles, author (Global Complex Type), price, and quantity).

**5 Click OK.**


ProcessInventory_Sample - NetBeans IDE 6.1
File Edit View Navigate Source Refactor Build Run Profile Versioning Tools Window Help
Pr... Services
ProcessInventory_S
Process Files
bookInventory.
Referenced Resour
bookInventory.xsd
Source Schema Design
Schema > Elements > bookstore
http://xml.netbeans.org/sche
Attributes
Attribute Groups
Complex Types
Elements
Groups
Referenced Schemas
Simple Types
bookstore (instance of :book)
book

**6 Click Save All.**