Using the File Binding Component in a Project

Sun Microsystems, Inc.
4150 Network Circle
Santa Clara, CA 95054
U.S.A.

Part No: 821–0239
June 2009

090702@22510

Contents

Using the File Binding Component in a Project

What You Need to Know

These topics provide information you need to know before you use the File Binding Component.

Reference Information

These topics provide additional reference information about configuring and using the File Binding Component.

About the File Binding Component

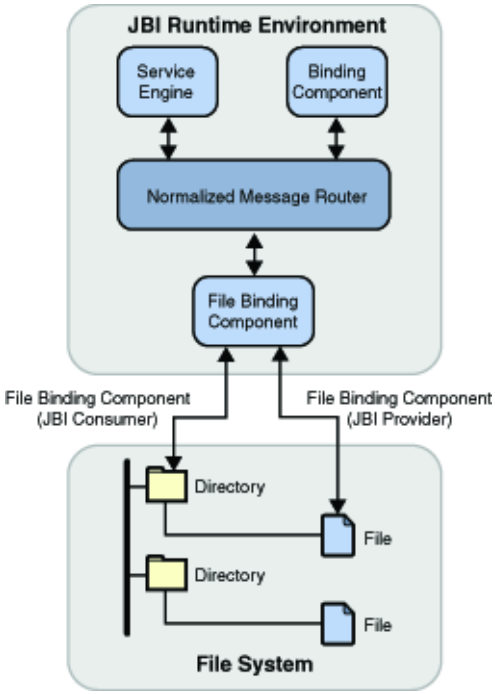
The File Binding Component is a JSR-208 compliant JBI runtime component that provides a transport service to a file system and offers a comprehensive solution to interact with the file system from the JBI environment.

Looking at what File BC does at a very high level: On the server side, File BC polls for inbound messages, stored in file(s), in a specified directory. On the client side, File BC puts messages into file(s) in a designated directory.

When the File Binding Component acts as a JBI provider, the default behavior is that it denormalizes the JBI message and writes the message to a specified destination in the file system, but it could also provide the capability to do a file read if the action is specified explicitly. When the File Binding Component acts as a JBI consumer, it polls the file system for a specified file name (or file names matching a supported pattern), normalizes to a JBI message and routes the message to the Normalized Message Router (NMR) so it can be serviced by other JBI components.

The design time component of File BC is a Netbeans module that provides plug-in to NetBean's project system and thus defines how file binding can be used. The runtime component implements all required component interfaces in JBI specification and provides the functionality to act as a proxy to services enabled using the file protocol.

The following diagram illustrates relationship between the File Binding Component and the other components within the runtime environment.



File Binding Component Features

Following features are supported by the File Binding Component:

- Supports in-only, in-out message exchange modes
- Supports explicit on-demand read action
- Supports Application Variables
- Supports Application Configuration
- Message exchange redelivery capability
- Message exchange graceful recovery
- Inbound throttling to control message handling concurrency
- Multiple Endpoint to poll the same directory
- Polling and writing multiple records per file
- Staging, achieving and error handling files
- Binary attachment support

Common User Scenarios

The following common user scenarios convey how components interact with external systems to achieve specific business goals.

Polling a Directory

A typical scenario for the File Binding Component as a service consumer is polling a directory for files that match specified file names or file name patterns. When polling a directory, you can specify the following:

- Polling Interval - Use the `pollingInterval` message property to specify how frequently to check the directory.
- Pattern Matching - If the `fileNameIsPattern` message property is set, then the `fileName` message property indicates the file name pattern for files to retrieve. Patterns that can be embedded in a file name include an incremental counter, a UUID identifier, or a timestamp.

  Refer to the section "File Name Patterns" on page 23 for more information.

Writing Files to a Directory

A typical scenario for the File Binding Component as a service provider is writing files to a specific directory. When writing the files, you can also use pattern matching to generate file names. When writing the files, you can specify the following:

- File Type - Text, Binary and XML files are supported.
- Pattern Generation - If the `fileNameIsPattern` message property is set, then the `fileName` message property indicates the file name pattern to use when writing the files. Patterns that can be embedded in a file name include an incremental counter, a UUID identifier, or a timestamp.

  Refer to the section "Application Variables in File Name Patterns" on page 24 for more information.

Multiple Records in a File

In the File Binding Component, various message properties facilitate the processing of multiple records in a file.

You can specify multiple records in a file by specifying a delimiter character (for variable length records) or by specifying a record size (for fixed length records). In either case, you must first set the `multipleRecordsPerFile` message property to enable processing of multiple records in a file. You can also use end-of-line characters when processing multiple records in a file.

If the `multipleRecordsPerFile` is set, then use the `recordDelimiter` property to specify the marker between records in a file.

If you want to specify fixed-length records, then do not specify the `recordDelimiter` property. Instead, specify the length of the records with the `maxBytesPerRecord` property. If this property is used to indicate the length of each record in a file, then `multipleRecordsPerFile` must be set.

End-of-Line Characters

You can specify whether to add an EOL (end-of-line) character to a record when writing to a file or whether to discard the EOL character after reading a record. The `addEOL` and `removeEOL` message properties are useful when the EOL character is used to separate multiple records in a file.

Runtime Configuration

The File Binding Component's runtime properties can be configured from the NetBeans IDE, or from a command prompt (command line interface) during installation.

The File Binding Component properties apply to the binding component as a whole, including all provider and consumer endpoints.

▼ Accessing the File Binding Component Runtime Properties

1. **From the Services tab of the NetBeans IDE, expand the `Servers` node.**
2. **Start your application server, for example GlassFish V2. To do this, right-click your application server and select `Start` from the shortcut menu.**
3. **Under the application server, expand the `JBI`→ `Binding Components` nodes and select `sun-file-binding`.**

   The current File Binding properties are displayed at the right side of the NetBeans IDE. You can also double-click `sun–file–binding` to open the Properties window.

   Edit the properties as needed. To apply any changes you make to the Application Configuration and Application Variables properties, shut down and restart the affected composite applications, Any changes in the "Threads" count property will be applied dynamically without restarting the File Binding Component. Any changes to the component Logger setting will be applied dynamically without restarting the File Binding Component.

The File Binding Component Runtime Properties

The File Binding runtime properties are categorized into three types:

- General Properties
- Statistics Properties
- Loggers Properties

General Properties

Name	*Description*	*Default Value*
Description	Indicates the purpose of the File Binding Component. This property is displayed for reference purpose.	Java Enterprise Edition File Binding
Name	Indicates the name of the File Binding Component. This property is displayed for reference purposes.	sun-file-binding
State	Indicates the state of the File Binding Component as "Started" or "Stopped." This property is displayed for reference purposes.	Started
Type	Indicates the type of component. This property is displayed for reference purposes. (service-engine or binding-component)	binding-component
Identification Properties		
Version	File Binding Component specification fully supported by this build.	<number-version
Build Number	Date and time stamp for the current build.	<build_number>
Configuration Properties		

Number of Outbound Processor Threads	Specifies the maximum number of threads to process outbound HTTP/SOAP invocations concurrently. The value can be any integer from 1 to 2147483647. This is a required property.	5
Application Configuration	Specifies the values for a Composite Application's endpoint connectivity parameters (normally defined in the WSDL service extensibility elements), and apply these values to a user-named endpoint ConfigExtension Property. The Application Configuration property editor includes fields for all of the connectivity parameters that apply to that component's binding protocol.	User defined
Application Variables	Specifies a list of name: value pairs for a given stated type. The application variable name can be used as a token for a WSDL extensibility element attribute in a corresponding binding. The Application Variables configuration property offers four variable types: ■ String: Specifies a string value, such as a path or directory ■ Number: Specifies a number value ■ Boolean: Specifies a Boolean value ■ Password: Specifies a password value	User defined

Statistics Properties

Statistics properties include 19 different component activities including exchanges, errors, requests, replies, and so forth. It lists component statistics that are collected for actions such as endpoints activated, average response time, completed exchanges, and so forth. Running statistics are automatically collected and displayed

Loggers Properties

Loggers properties include 13 different component activities that can be recorded by the `server.log`. The Logger properties specify the user-designated level of logging for an event.

Each logger can be set to record information at any of the following levels:

- FINEST: messages provide highly detailed tracing
- FINER: messages provide more detailed tracing
- FINE: messages provide basic tracing
- CONFIG: provides static configuration messages
- INFO: provides informative messages
- WARNING: messages indicate a warning
- SEVERE: messages indicate a severe failure
- OFF: no logging messages

Configuring File BC WSDL Attributes

To configure a JBI component to access web services, you specify both service level WSDL elements and binding level WSDL elements.

Service Level WSDL Elements

Service level WSDL elements allow you to specify the "connectivity" information to a file system. The File address extensibility element is the File Binding Component service level WSDL element.

The service element specifies the connectivity to the file system using the File address element `file:address`. The properties of address further allows you to specify a path to a directory and then to specify whether the path is a absolute path or a relative path.

File Address Element

When you create a WSDL file for a BPEL project in the NetBeans IDE, the new WSDL Document wizard generates the address service definition. You can then edit the properties of address service to specify the path.

The following table describes the properties available for this service:

Property	Description	Required	Example
fileDirectory	Defines the directory path in the file system to read from or write to. Defines the directory path in the file system to read from or write to. If the `relativePath` property is false, then `fileDirectory` represents an absolute path. If `relativePath` property is true, then `fileDirectory` represents a path relative to the path specified in the `pathRelativeTo` property.	Required	/home/joe/data
relativePath	Specifies whether the directory specified in the `fileDirectory` property is a relative path.	Optional	true
pathRelativeTo	Defines the base directory for the directory defined by `fileDirectory`. Select one of the three possible values: ■ User Home (The home directory of the user) ■ Current Working Directory ■ Default System Temp Dir	Optional	User Home

Property	Description	Required	Example
lockName	Specifies the lock file name which is created under the target directory specified by `fileDirectory`. Inbound readers use the lock to synchronize their concurrent access to the target directory.	Optional	filebc.lck
seqName	Specifies the name of the file where the current value of the persistence backed sequence number is stored. The file resides in the target directory specified by `fileDirectory`.	Optional	filebc.seq
workArea	Specifies the temp directory name where UUID tagged input files wait to be further processed. It is a functioning "staging area" for inbound processing, and is relative to the target directory specified by `fileDirectory`.	Optional	filebc_tmp

The following example illustrates the WSDL service element:

```
<service name="FileService">
   <port name="FILEPort" binding="tns:FILEBinding">
      <file:address fileDirectory="C:\open-jbi-components\driver-test\ftpbc\FTPBCTests\
      HelloDukeCompApplication\test\HelloDukeTest\in_out"/>
   </port>
</service>
```

Binding Level WSDL Elements

Binding level WSDL elements allow you to define the file "transport" specific information for operations and messages. The File Binding Component binding level WSDL elements include the File binding, operation, and message extensibility elements.

File Binding Element

The file binding extensibility element allows the association of a binding to be file protocol specific. The message format and protocol for the File Binding Component is always a file that is supported on the native file system. When you create a WSDL file for a BPEL project in the NetBeans IDE, the New WSDL Document wizard generates the file binding definition, which includes a name you specify and a type that is generated by the wizard.

The following example illustrates the File binding element:

```
<binding name="FileInFileOutBinding" type="tns:FileInFileOutPortType">
  <file:binding/>
</binding>
```

File Operation Element

The file operation element defines the supported operations. For the File Binding Component the operations that can be supported are one-way and request-response.

The following example illustrates the File operation element:

```
<binding name="FileBinding" type="tns:FilePortType">
    <file:binding/>
    <operation name="FileOp">
        <file:operation/>
</binding>
```

File Message Element

The File message element extends the binding element to specify properties associated with reading input files or writing output files in the file system. In the NetBeans IDE, select the properties for a file:messages() element to specify the behavior for the message.

The following table describes the message properties available for reading from or writing to the file system.

TABLE 1 Message Properties to Read/Write to a File System

Property	Description	Required	Example
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.	Required	Literal (Default)
fileName	Defines the file name relative to the specified directory to read from or write to. If `fileNameIsPattern()` is false, 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.	Required	data.xml (Default)
fileNameIsPattern	Indicates whether or not the `fileName()` attribute designates a filename pattern.	Optional	false

TABLE 1 Message Properties to Read/Write to a File System *(Continued)*

Property	Description	Required	Example
recordDelimiter	Defines the record delimiter when multiple records are present. The value of this attribute is considered only if `multipleRecordsPerFile()` is true. If no value is specified for this attribute but `maxBytesPerRecord()` is defined when reading a file, it is assumed that each record is of fixed length with the length being the value defined for `maxBytesPerRecord()`. Otherwise, the default record delimiter is the linefeed character.	Optional	\r\n
pollingInterval	Defines the polling interval that the File Binding Component searches for input files in the specified directory. The polling interval is specified in milliseconds and has a default value of 1000ms. This attribute has no effect for writing.	Required	1000 (Default)
fileType	Specifies whether the file is a text file or binary file.	Optional	text (Default)
encodingStyle	Specifies the encoding type associated with the message (or message part). This also defines the encoder type responsible to process the encoded data.	Optional	customencoder-1.0

TABLE 1 Message Properties to Read/Write to a File System *(Continued)*

Property	Description	Required	Example
part	The abstract WSDL configuration for message can have one or more parts. The part() property references one of the parts that is named in the abstract WSDL configuration.	Optional	Part1
addEOL	Defines whether newline characters (EOL characters) should be appended to the outbound message or to each record incase of multiple records.. This attribute has no effect for reading	Optional	false
removeEOL	Defines whether newline characters (EOL characters) should be removed before processing. or be removed for each record incase of multiple records. This attribute has no effect for writing	Optional	false
maxBytesPerRecord	Defines the maximum number of bytes to be read per record. If only a single record is present, this attribute defines the total number of bytes to be read. This attribute has no effect for writing.	Optional	1024
multipleRecordsPerFile	Specifies whether the file to read from or write to contains multiple records or should be considered a single payload.	Optional	false

TABLE 1 Message Properties to Read/Write to a File System *(Continued)*

Property	Description	Required	Example
archive	Specifies whether a message is archived after it is read for processing by the File Binding Component. When true, the message is moved to the directory specified by `archiveDirectory`. When a file is processed, the File Binding Component renames the file to `input_file_name_processed()` or input_file_name_error(), depending on the success of the operation. If the `archive()` property is true, the file is then moved to the `archiveDirectory` and a UUID is appended to the file name.	Optional	true

TABLE 1 Message Properties to Read/Write to a File System *(Continued)*

Property	Description	Required	Example
archiveDirectory	If the `archive()` property is set to `true()`, `archiveDirectory()` specifies the directory to which processed (read) messages will be moved. Filenames for archived messages are tagged with a UUID to avoid file name collision in the archive directory. If the `archiveDirIsRelative()` property is `true()`, then the directory specified here is a relative directory. The parent directory is the directory specified in the `fileDirectory()` property for the `file:address()` element.	Optional	true
archiveDirIsRelative	Indicates whether the directory specified in `archiveDirectory()` is an absolute directory or a relative directory. If `true()`, the parent directory is the directory specified in the `fileDirectory()` property for the `file:address()` element.	Optional	true

TABLE 1 Message Properties to Read/Write to a File System *(Continued)*

Property	Description	Required	Example
protect	Indicates if overwrite protection is required before writing messages to a file. When `true()`, existing files of the same name will be moved to the directory specified by `protectDirectory` before the current message is written. When `false()`, the files of the same name will be overwritten.	Optional	true
protectDirectory	If the protect property is set to true, `protectDirectory` specifies the directory to which files will be moved to prevent them from being overwritten. Filenames for protected messages are tagged with a UUID to avoid file name collision in the protect directory. If the `protectDirIsRelative()` property is `true()`, then the directory specified here is a relative directory. The parent directory is the directory specified in the `fileDirectory()` property for the `file:address()` element.	Optional	false

TABLE 1 Message Properties to Read/Write to a File System *(Continued)*

Property	Description	Required	Example
protectDirIsRelative	Indicates whether the directory specified in `protectDirectory()` is an absolute directory or a relative directory. If `true()`, then the parent directory is the directory specified in the `fileDirectory()` property for the `file:address()` element.	Optional	true
stage	Indicates if staging is enabled. If set to `true()`, then messages are first written to the directory specified by `stagingDirectory`. When the message is completely written to the staging directory, then it is moved to the target directory for the message. Filenames for staged messages are tagged with a UUID to avoid file name collision in the staging directory.	Optional	false

TABLE 1 Message Properties to Read/Write to a File System *(Continued)*

Property	Description	Required	Example
stageDirectory	If the stage property is set to true, `stageDirectory()` specifies the directory to which files will first be written before being moved to the target directory for the message. If the `stageDirIsRelative()` property is true, then the directory specified here is a relative directory. The parent directory is the directory specified in the `fileDirectory()` property for the `file:address()` element.	Optional	false
stageDirIsRelative	Indicates whether the directory specified in `stageDirectory` is an absolute directory or a relative directory. If true, then the parent directory is the directory specified in the `fileDirectory()` property for the `file:address()` element.	Optional	true

File Name Patterns

File name patterns are used as a name filter for inbound message processing and as a name generator for outbound message processing. The multiple processing threads that make up the runtime File Binding Component fall into two types:

- Inbound Processors: Message consumers that poll an input directory at a specified interval for a specific file name.
- Outbound Processors: Message provisioners that write messages to an output directory by a specific file name.

Literal File Name

Input and output file names can be literal, or use a pattern.

- Input: If the input file name is a literal, the inbound processor polls the input directory for a file by that name, and the content of the file is converted to a normalized message.
- Output: If the output file name is a literal, the outbound processor writes the denormalized message to a file by that name.

Pattern File Name

In most cases, the file names specified for inbound or outbound processing are patterns. File name pattern is a proprietary mechanism of the File Binding Component.

- Inbound: When used by the inbound processor, the pattern serves as a filter. That is, if a file name matches the pattern, it is selected by the inbound processor and its content is read, normalized, and sent.
- Outbound: When used by the outbound processor, the pattern serves as concrete name generator. That is, the special pattern symbols, such as %d(), %u(), %t(), and %{seq_name}(), that appear in the file name pattern specified as output file, will be expanded. The symbols are substituted with their current value and derive a concrete file name, to which the denormalized message is written.

When you set `fileNameIsPattern()` on a message property, you can specify patterns for generating filenames for outbound messages or for reading filenames for inbound messages.

Application Variables in File Name Patterns

You can use application variables together with file name patterns to specify paths.

The following example shows how to use an application variable when specifying a path name to a file that uses the %d pattern marker:

```
${base_dir}/subdir/data_%d_in.xml

${base_dir}/subdir/data_%d_out.xml
```

This example specifies a directory defined by the application variable base_dir. The application variable is defined as a File Binding component runtime property. If base_dir is defined as /duke/home, then this example reads or writes the following files:

```
/duke/home/file_n_in.xml

/duke/home/file_0_out.xml
```

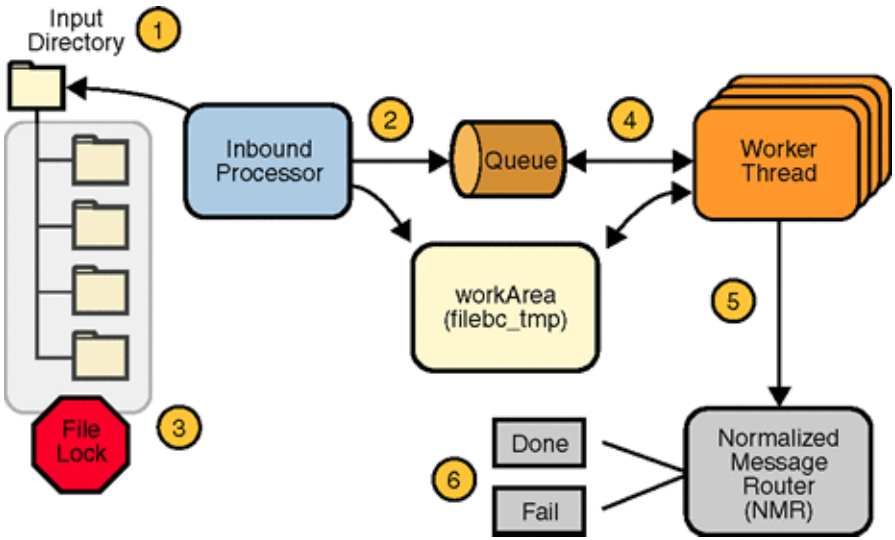
Inbound Message Processing

It is important that the Inbound files are picked up only once by one inbound thread and the outbound files are not overwritten by simultaneous threads.

To make certain files are not overwritten by simultaneous threads, a locking mechanism is used to synchronize the threads that poll the same endpoint or input directory.

Each endpoint is unique, and is associated with a physical file folder path, for example: `C:\temp`. This is where the inbound messages are pulled. It is invalid to deploy a composite application that contains endpoints that poll the same physical directory for the same file.

The life cycle of an inbound message can be expressed using the following illustration.



The process flow can be described by the following steps:

1. The inbound processor polls the input directory.
2. If a file name matches the given file name pattern, the inbound processor moves the file to the `workArea` directory (filebc_tmp), and the name and path to the selected file is put into the queue.

3. The selected files are locked by a file lock to prevent other inbound processors in a clustered environment from selecting the same file.
4. Five inbound worker threads wait on the queue to process the selected inbound files
5. When a worker thread selects a file, it reads the file content, normalizes the content, and sends normalized message to the NMR (Normalized Message Router).
6. If sending the message to the NMR is a success, the message is moved from the workArea to an archive, and the suffix "_processed" is added to the file name.

If sending the message to the NMR fails, the message is retained in the `workArea` to be processed by the user. An "_error" suffix is added to the file name. "_error" file contains details of the failure.

File Binding Component Processing Protocol

The following attributes are used for the implementation of the locking/dispatch mechanism:

- file:address/@lockName- This is the file name used for the F_LOCK. This is located under the input directory specified by the `file:message/@fileDirectory()`, default value: `filebc_lock()`.
- file:address/@workArea- The sub directory name used as a working directory. This is relative to the input directory specified by the `file:message/@fileDirectory()`, default value: `filebc_tmp()`.
- file:address/@seqName- The file name used as the persistence of a sequence number. This is located under the input directory specified by the `file:message/@fileDirectory()`, default value: `filebc_seq()`

Persisted Sequencing

The symbol % { <seq_name}() indicates a reference to a persisted sequence number by name, where the sequence numbers's current value is incremented by one and the reference is substituted.

The lexical definition is: <seq_name> =: (0–9a-zA-Z-_) + ()

The current sequence value is persisted so that it survives if the application is shutdown or undeployed, or if the JBI container (application server) is shutdown.

Mapping Persisted Sequences to File Based Persistences

A file is used as the persistence storage of sequences. For a File Binding Component service, as defined in a WSDL, the scope of the sequences are the endpoints associated with the bindings that contain the references to these sequences. As displayed in the following WSDL, there are two bindings which reference sequences by the name seq-v.1.

- For seq-v.1() in FILE_OB_SEQBinding(), its scope is the EP0 identified by FILE_OB_SEQService() + tns:FILE_OB_SEQBinding()
- For seq-v.1() in FILE_OB_SEQBinding1(), its scope is the EP1 identified by FILE_OB_SEQService1() + tns:FILE_OB_SEQBinding1()

The mapping of a sequence to a file system file is demonstrated in the WSDL file below.

```
<definitions name="FILE_OB_SEQRIT17157" targetNamespace="http:j2ee.netbeans.org/wsdl/
FILE_OB_SEQ"
xmlns="http:schemas.xmlsoap.org/wsdl/"
    xmlns:wsdl="http:schemas.xmlsoap.org/wsdl/"
    xmlns:soap="http:schemas.xmlsoap.org/wsdl/soap/"
    xmlns:xsd="http:www.w3.org/2001/XMLSchema" xmlns:tns="http:j2ee.netbeans.org/wsdl/
    FILE_OB_SEQ"
    xmlns:plnk="http:docs.oasis-open.org/wsbpel/2.0/plnktype" xmlns:file="http:schemas
    .sun.
    com/jbi/wsdl-extensions/file/">
<types/>
<message name="FILE_OB_SEQOperationRequest">
    <part name="part1" type="xsd:string"/>
</message>
<portType name="FILE_OB_SEQPortType">
    <wsdl:operation name="FILE_OB_SEQOperation">
        <wsdl:input name="input1" message="tns:FILE_OB_SEQOperationRequest"/>
    </wsdl:operation>
</portType>
<portType name="FILE_OB_SEQPortType1">
    <operation name="FILE_OB_SEQOperation">
        <input name="input1" message="tns:FILE_OB_SEQOperationRequest"/>
    </operation>
</portType>
<binding name="FILE_OB_SEQBinding" type="tns:FILE_OB_SEQPortType">
    <file:binding/>
    <wsdl:operation name="FILE_OB_SEQOperation">
        <file:operation/>
        <wsdl:input name="input1">
            <file:message use="literal" fileName="output%{seq-v.1}.out"
            pollingInterval="1000" fileNameIsPattern="true"/>
        </wsdl:input>
    </wsdl:operation>
</binding>
<binding name="FILE_OB_SEQBinding1" type="tns:FILE_OB_SEQPortType1">
    <file:binding/>
    <operation name="FILE_OB_SEQOperation">
        <file:operation/>
        <input name="input1">
            <file:message use="literal" fileName="output%{seq-v.1}.out"
            pollingInterval="1000" fileNameIsPattern="true"/>
        </input>
    </operation>
</binding>
<service name="FILE_OB_SEQService">
    <wsdl:port name="FILE_OB_SEQPort" binding="tns:FILE_OB_SEQBinding">
        <file:address fileDirectory="\\jfu-tecra\TEST_NETWORK_FILE\area_polled_EP0"
        lockName="filebc.lck" workArea="filebc_tmp" seqName="filebc.seq"/>
    </wsdl:port>
</service>
<service name="FILE_OB_SEQService1">
    <port name="FILE_OB_SEQPort" binding="tns:FILE_OB_SEQBinding1">
        <file:address fileDirectory="\\jfu-tecra\TEST_NETWORK_FILE\area_polled_EP1"
        lockName="filebc.lck" workArea="filebc_tmp" seqName="filebc.seq"/>
    </port>
</service>
<plnk:partnerLinkType name="FILE_OB_SEQRIT171571">
    <!-- A partner link type is automatically generated when a new port type is added.
Partner link types are used by BPEL processes.
In a BPEL process, a partner link represents the interaction between the BPEL process
and a partner service. Each partner link is associated with a partner link type.
A partner link type characterizes the conversational relationship between two services.
The partner link type can have one or two roles.-->
        <plnk:role name="FILE_OB_SEQPortTypeRole" portType="tns:FILE_OB_SEQPortType"/>
    </plnk:partnerLinkType>
</definitions>
```

In this example, %{se-v.1}() gives reference to a different sequence number when it appears in different bindings.

The persisted storage for a sequence is a file with the same name under the directory specified by the file:address->fileDirectory of the corresponding service binding.

For example, in the above WSDL, the persisted files are:

- %{seq-v.1() in EP0 is persisted in the file:
  jfu-tecra\TEST_NETWORK_FILE\area_polled_EPO\filebc.seq\seq-v.1 %{seq-v.1}
- %{seq-v.1() in EP1 is persisted in the file:
  jfu-tecra\TEST_NETWORK_FILE\area_polled_EP1\filebc.seq\seq-v.1 %{seq-v.1}

Initial Value of a Persisted Sequence

The initial value of a persisted sequence is 0. The value is then incremented by 1 every time that sequence is referenced. To create a sequence the starts with a number that is greater than zero, edit the start value in the persisted storage file when the file is not in use.

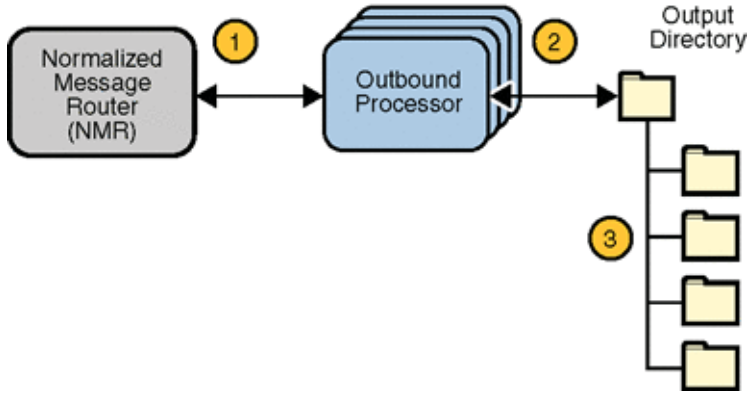
Concurrence Control of Access to the Sequence Storage File

Access to a specific sequence in persistence storage file is thread and clustering safe. This means that the file is read and updated by a single thread across clustered servers and JVMs, this can be on one host or multiple hosts.

Outbound Message Processing

Outbound message processing always utilizes File Name Patterns to ensure that messages are processed once only and are not overwritten.

The life cycle of an outbound message can be explained using the following illustration.



The process flow can be described by the following steps:

1. Outbound processors wait on the Normalized Message Router for outbound messages. The number of available outbound processors is configured by the runtime parameter `Outbound Threads()`.

2. When an outbound message is available, an outbound processor takes the message, denormalizes the message, and writes the message payload to the specified file destination.
3. A file name pattern is used to generate the unique message name. The file name pattern is specified by the WSDL File message element attribute `fileNameIsPattern()`.

For example, a UUID can be added to the file name using the following value: `output._%u.dat()`. When persisted sequence numbering is used to provide a file name pattern, the outbound processor reads the sequence number from the `my_sequence` directory and adds this number to the file name. It then increments the number by one and writes the new number back to the `my_sequence` directory. This process is synchronized so that only one outbound process can access the `my_sequence` directory at any time, ensuring that the persisted sequence number retains its integrity.

Application Variable Support

The binding component Application Variables property allows you to define a list of `name:value` pairs for a given stated type. The application variable name can be used as a token for a WSDL extensibility element attribute in a corresponding binding. For example, if you were defining an application variable for the hostname as FOO, then the WSDL attribute would be ${FOO}. In the Application Variables property you would enter a String value of FOO for the name, and the desired attribute as the value. When you deploy an application that uses application variables, any variable that is referenced in the application's WSDL is loaded automatically.

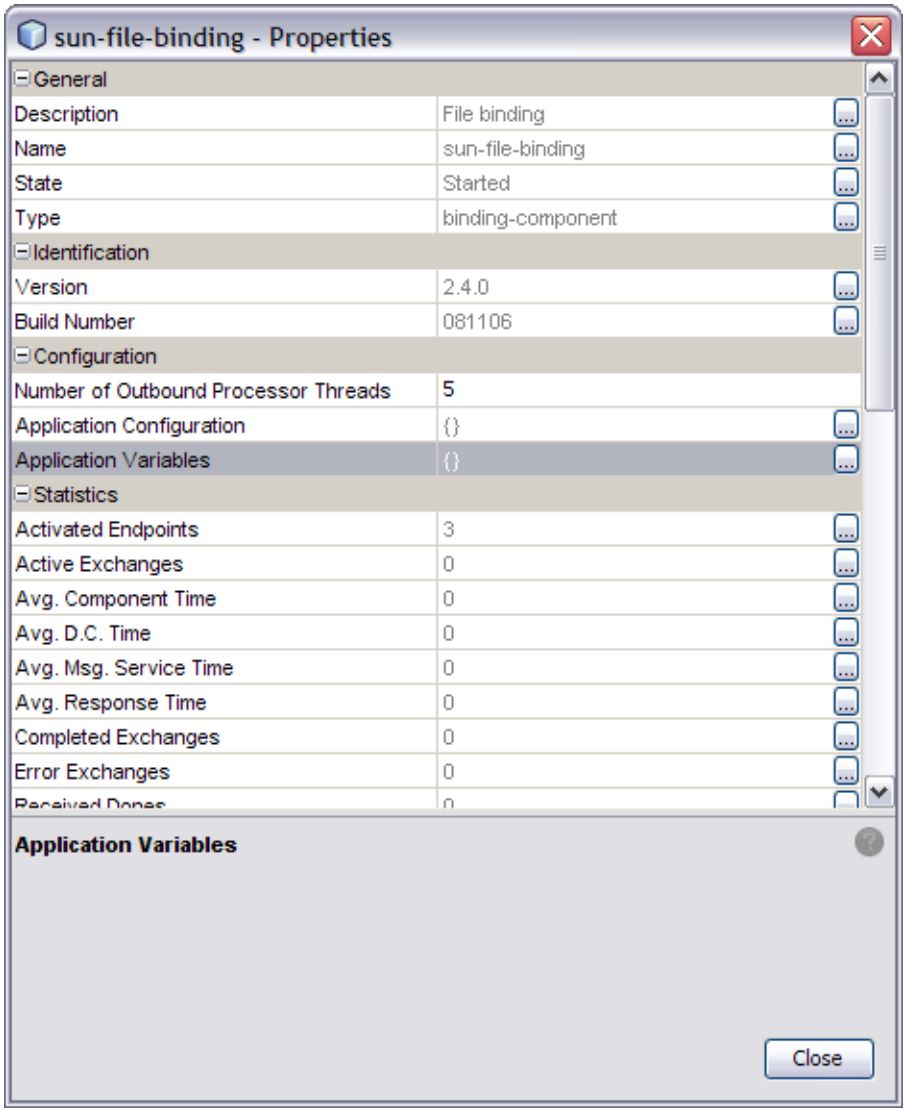
The Application Variables configuration property offers four variable types:

- String: Specifies a string value, such as a path or directory.
- Number: Specifies a number value.
- Boolean: Specifies a Boolean value. The VALUE field provides a checkbox (checked = true).
- Password: Specifies a password value. The password is masked and displays only asterisks.

Variables also allow greater flexibility for your WSDL files. For example, you can use the same WSDL for different runtime environments by using application variables to specify system specific information. These values can then be changed from the binding component runtime properties as needed, for any specific environment.

When you deploy an application that uses Application Variables, all of the Application Variables that are referenced in the application's WSDL files are loaded automatically. If you attempt to start an application and an Application Variables value is not defined (no value is specified for the Application Variable) an exception is thrown.

To change a property when the application is running, change your Application Variable property value, then right-click your application in the Services window under Servers > GlassFish V2> JBI > Service Assemblies, and click Stop in the popup menu. When you restart your project, your new settings will take effect.


sun-file-binding - Properties
General
Description
File binding
Name
sun-file-binding
State
Started
Type
binding-component
Identification
Version
2.4.0
Build Number
081106
Configuration
Number of Outbound Processor Threads
5
Application Configuration
{}
Application Variables
{}
Statistics
Activated Endpoints
3
Active Exchanges
0
Avg. Component Time
0
Avg. D.C. Time
0
Avg. Msg. Service Time
0
Avg. Response Time
0
Completed Exchanges
0
Error Exchanges
0
Application Variables
Close

Application Configuration Support

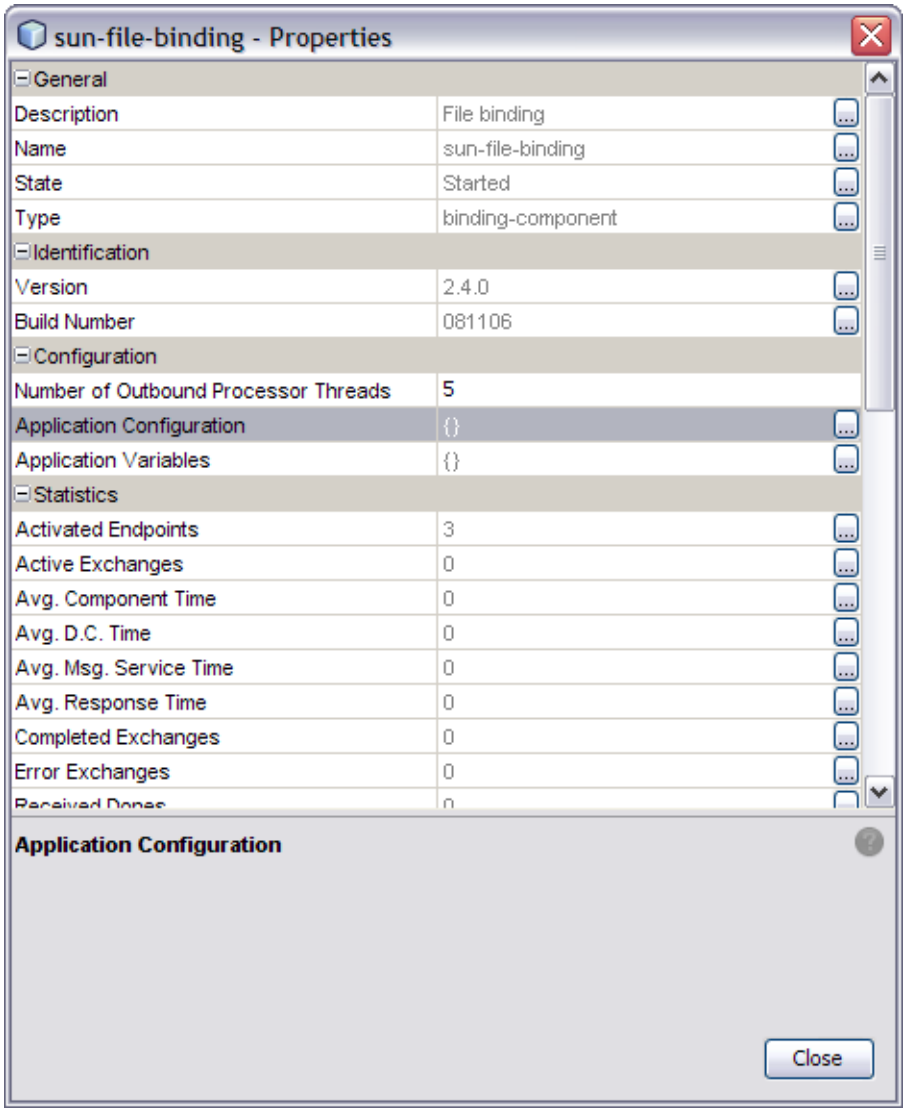
An Application Configuration Object (ACO) defines a set of values which can be used to override `file:address` attributes such as "fileDirectory" defined in the WSDL.

The Application Configuration property allows you to configure the external connectivity parameters for an application that you have created, such as a service assembly, and without changing or rebuilding the application, deploy the same application into a different system. For example, you could take an application that is running in a test environment, and deploy it to a production environment without rebuilding the application.

From the Application Configuration property, you can specify values for a Composite Application's external connectivity parameters, which are normally defined in the WSDL service extensibility elements. You can then apply these values to a user-named endpoint ConfigExtension Property. The Application Configuration property editor includes fields for all of the connectivity parameters that apply to that component's binding protocol. When you enter the name of a saved ConfigExtension and define the connectivity parameters in the Application Configuration editor, these values override the WSDL defined connectivity attributes when your project is deployed. To change these connectivity parameters again, you simply change the values in the Application Configuration editor, then shutdown and start your Service Assembly to apply the new values.

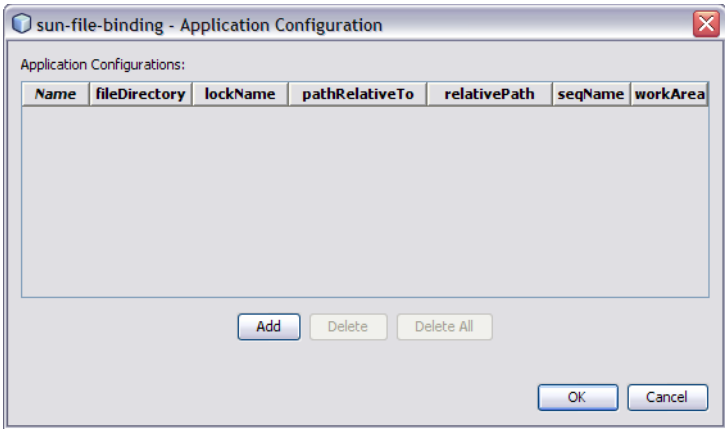
The Application Configuration property editor allows you to create several application configurations referenced by their own user-defined names. Note that different binding component protocols will have different attributes. The File binding attributes are not the same as the JMS or the HTTP binding attributes, and therefore, the Application Configuration property editors for each of these binding components will contain different attributes.

To change a property when the application is running, change your Application Configuration property value, then right-click your application in the Services window under Servers > GlassFishV2 > JBI > Service Assemblies, and click Stop in the popup menu. When you restart your project, your new settings will take effect.



The File Binding Component's Application Configuration property contains 6 parameters:

- fileDirectory
- lockName
- pathRelativeTo
- relativePath
- seqName
- workArea



Processing Protocols and Capabilities

The consumer mode in JBI or the server side processing activities are referred as Inbound processing. Error Handling and Recovery are an examples for Inbound processing.

The provider mode in JBI or the client side processing activities are referred to as Outbound processing. File write and on-demand read protocols will be covered in this section.

Inbound Processing

The following list describes the Inbound processing file poll protocols and their capabilities:

- Polling for file(s) in a specified directory.
- Instead of renaming the original files, create a work directory relative to the input directory. Name of the work directory is configurable via the `workArea` setting in file:address, with the default being "filebc-in-processing".
- Move each input file to work directory, and tag the original file names with a UUID.
- Inbound worker thread reads files from the work directory and processes contents as follows:
  - If multiple records per file setting is false (default), the content of the file will be treated as the payload and set in the normalized message content.
  - If multiple records per file is true, create separate normalized messages one for each record. How a record is read from the file depends on maxBytesPerRecord and recordDelimiter settings.
- All successfully processed input files will be archived, if the "archive" setting is true.
  - A "processed_files" directory will be created relative to the input directory

- Original files will be renamed with “_processed” suffix and put into the "processed_files" directory

Error Handling

In case of errors while processing an input file:

- An "error" folder will be included in the "new" folder created by the Binding Component in the input directory.
- Original UUID tagged file (from the work directory) will be moved to the errors directory and a additional file with "_error" suffix will be generated. "_error" file will contain details of the failure.

Recovery

Recovery is a mechanism used to prevent message loss in case of system crash.

On startup, Inbound Processor will first check the work directory and put any unprocessed files in the queue for the worker thread. This ensures that any files left unprocessed in the work directory will be processed when the systems restarts.

Outbound Processing

The following list describes the Outbound processing file write protocols and their capabilities:

- Writing file(s) in a specified directory
- Write single (overwrite mode) or multiple records (append mode) to a file
- On Demand Read function acts when a JBI service invokes the File Binding Component to read a specific message from a file directory.
- Protects existing files in a directory from being overwritten by new files with the same name
- If staging attribute is true, and not in append mode (multiple records per file), Outbound messages are written first to a staging directory, then the completed file is moved to target directory. This ensures that the output file is exposed only when message has been written completely.

Normalized Message Properties

Normalized Message properties are commonly used to specify metadata that is associated with message content. `javax.jbi.security.subject` and `javax.jbi.message.protocol.type` are two examples of standard normalized Message properties defined in the JBI Specification.

Normalized Message properties are used to provide additional capabilities in Open ESB, such as:

- Getting and Setting transport context properties. For example, HTTP headers in the incoming HTTP request, or file names read by the File Binding Component
- Getting and Setting protocol specific headers or context properties (SOAP headers)
- Getting and Setting additional message metadata. For example. a unique message identifier, or an endpoint name associated with a message
- Dynamic configurations. For example, to dynamically overwrite the statically configured destination file name at runtime

  Some of the use cases mentioned above require protocol/binding specific properties, typically used by a particular binding component. Other properties are considered common or general purpose properties that all participating JBI components make use of, for example, the message ID property, which can be utilized to uniquely identify or track a given message in the integration.

Normalized Message Properties Defined by the File Binding Component

The following table describes the Inbound NM properties defined by the File Binding Component.

TABLE 2 Inbound NM Properties

Property	Description	Type
org.glassfish.openesb.file.inbound.filedirectory	File directory which is polled for the input file	String
org.glassfish.openesb.file.inbound.filename	Input file being read/poll	String
org.glassfish.openesb.file.inbound.datatype	Data type of the file, can be text, binary, xml	String
org.glassfish.openesb.file.inbound.batchid	A file may have multiple records, batchid represents the file	String
org.glassfish.openesb.file.inbound.recordnumber	Given a batch id , the record number represents a particular record	String
org.glassfish.openesb.file.inbound.lastrecord	Value can be either "true" or "false", this property is sent in case of last record	String
org.glassfish.openesb.file.inbound.endpointname	Represents the service name and endpoint name	String

The following table describes the Outbound NM properties defined by the File Binding Component.

TABLE 3 Outbound NM Properties

Property	Description	Type
org.glassfish.openesb.file.outbound.fileDirectory	The directory containing the file to read from or write to	String
org.glassfish.openesb.file.outbound.fileName	The file to read from or write to	String
org.glassfish.openesb.file.outbound.dataType	Data type of the file; can be text, binary, xml	String
org.glassfish.openesb.file.outbound.tempFileDirectory	The temp directory can be relative to user-home,current working dir, system default temp dir	String
org.glassfish.openesb.file.outbound.addEol	The value can be true or false, should end-of-the-line character be appended to a record/message	String
org.glassfish.openesb.file.outbound.append	The value can be true or false, signify multiple records per file	String
org.glassfish.openesb.file.outbound.appendDelimiter	If multiple records are true, this represents the record delimiter	String
org.glassfish.openesb.file.outbound.overwriteExistingFile	The value can be true or false, overwrite existing file if true	String

General Normalized Message Properties

Normalized Message properties are either General, available to all participating JBI components, or protocol/binding specific, used by a particular binding component.

The following General NM properties are available to all binding components.

TABLE 4 General NM Properties

Property	Description and Use	Type
org.glassfish.openesb.messaging.groupid	Uniquely identifies a message with the group to which a message belongs. For example, it applies the RM sequence group number for SOAP messages, or a time stamped file name (where the file record message comes from). This property is optional.	java.lang.String
org.glassfish.openesb.messaging.messageid	Uniquely identifies a message. For batch processing this might be a record number (for example, a particular record in a file), or a GUID. This property is mandatory.	java.lang.String
org.glassfish.openesb.messaging.lastrecord	The value is a string representation of boolean ("true" or "false"). This property can be used to signal the last record in a group, e.g. the last record in a RM sequence for SOAP messages, or the last record in a file when multiple record processing is turned on for File BC. This property is optional.	java.lang.String
org.glassfish.openesb.exchange.endpointname	The value a string representation of the endpoint name set on the exchange. This represents the endpoint name of the "owner" of the message, and could be made available by JBI runtime.	java.lang.String

Consistent Logging Strategies

The File Binding Component runtime Logger properties include 8 different component activities that can be monitored and recorded at user-designated levels. Logging levels are set separately for each of these activities from the File Binding Component Properties Editor.

Each logger can be set to record information at any of the following levels:

- FINEST: messages provide highly detailed tracing
- FINER: messages provide more detailed tracing
- FINE: messages provide basic tracing
- CONFIG: provides static configuration messages
- INFO: provides informative messages

- WARNING: messages indicate a warning
- SEVERE: messages indicate a severe failure
- OFF: no logging messages

Message Exchange Redelivery Capability

Before we proceed to Redelivery capability of the FIle Binding Component, it is important to know about the Quality of Service (QOS) attributes. Redelivery is a part of the QOS properties.

The QOS attributes are configured from the Config QoS Properties Editor, accessed from the Composite Application Service Assembly (CASA) Editor.

The following table describes the QOS attributes.

TABLE 5 QOS Attributes and Their Description

Attribute	Description	Example
Consumer Settings		
Service Name	Specifies the consumer service name. Click the ellipses button to open the QName Editor. Select a pre-existing Namespace URL or enter a new Namespace URL and prefix.	http://j2ee.netbeans.org/wsdl/PollInOut/Poll
Endpoint Name	Specifies the consumer endpoint name. Click the ellipses button to open an edit window.	PollIn_InboundPort
Provider Settings		
Service Name	Specifies the provider service name. Click the ellipses button to open the QName Editor. Select a pre-existing Namespace URL or enter a new Namespace URL and prefix.	http://enterprise.netbeans.org/bpel/PollInOu
Endpoint Name	Specifies the Provider endpoint name. Click the ellipses button to open an edit window.	FileInboundPortTypeRole_myRole
Redelivery Extension Settings		
Max Attempts	Specifies the number of times redelivery will be attempted before using the on failure option.	20

TABLE 5 QOS Attributes and Their Description *(Continued)*

Attribute	Description	Example
Wait Time	Specifies time (in milliseconds) to wait between redelivery attempts.	300
On Failure	Specifies the type of action to be taken when message exchange (ME) redelivery attempts have been exhausted. The on failure options are: ■ Delete ■ Error ■ Redirect ■ Suspend The File Binding Component supports only Error and Suspend.	Error
Throttling Extension Settings		
Max Concurrency Limit	Specifies the maximum number of concurrent messages that can be processed on a specific connection. This number is used to set up the maximum number of concurrent messages that the internal endpoint sends to the provider endpoint.	10

Configuring Redelivery

Redelivery is a Quality of Service mechanism that handles message delivery when first-time delivery fails. Redelivery allows you to define the number of attempts that the system makes to deliver a message, the time between attempts, and the final result for an undeliverable message or non-responsive endpoint. Redelivery is configured for a specific connection from the Composite Application Service Assembly (CASA) Editor, by clicking the QoS icon for that connection. This opens the Config QoS Properties for that connection. From the Redelivery Extension section of the editor, configure the Redelivery properties.

If ERROR status is returned for an Inbound message sent by File Binding Component, the BC will attempt to redeliver the message, based on Redelivery QoS settings. For example, a message could have ERROR status if BPEL Service Engine encounters a problem while processing the message.

The Redelivery configuration parameters are:

- Max Attempts: Specifies the number of times that the project attempts to re-deliver a message. An error status is returned to the JBI component for each failed attempt.

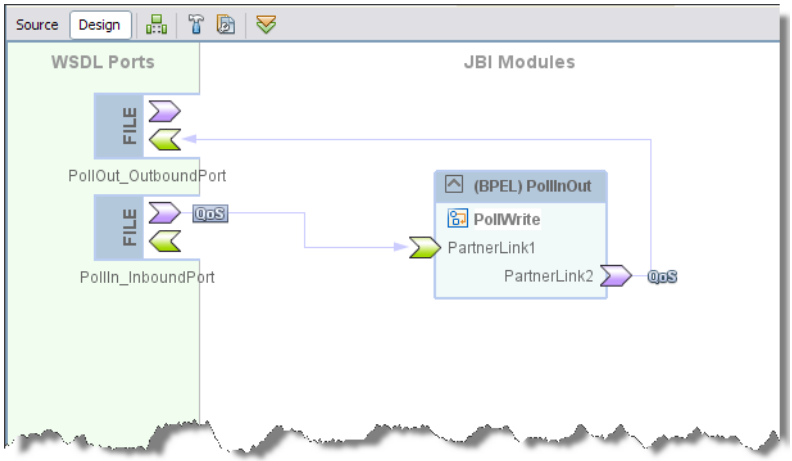
- Wait Time: Specifies the time, in milliseconds, that the project waits between redelivery attempts.
- On Failure: Specifies the actions taken and the message destination when the specified redelivery attempts have been exhausted. This parameter has four options: delete, redirect, suspend, and error.

  The On Failure parameter has four options: delete, redirect, suspend, and error.
- Delete: The delete option specifies that when the final attempt to redeliver the message has failed, the QoS utility deletes the message and returns a Done status to the JBI component, at which time the component proceeds to its next process. The delete option only supports In-Only message exchanges.
- Error: The error option specifies that when the final attempt to redeliver the message is exhausted, the providing JBI component will set the endpoint status to "ERROR", and will raise an exception and populate the Error property with it. This option is supported for both In-Only and In-Out message exchanges.
- Redirect: The redirect option specifies that after the final attempt to redeliver the message has failed, the QoS utility redirects the message to a user-defined endpoint, such as a “dead-message” folder. Upon successful delivery to the redirect endpoint, the QoS utility returns a Done status to the JBI component, at which time the component proceeds to its next process. The redirect option only supports In-Only message exchanges.
- Suspend: The suspend option specifies that when the final attempt to redeliver the message has failed, the JBI component suspends the process instance . This option is only supported if monitoring is enabled in the JBI Component, since the user must use the monitoring tool to resume a suspended instance. This option is supported for both In-Only and In-Out message exchanges.

▼ To Configure Redelivery

1 **From the NetBeans IDE Projects window, right-click the Service Assembly node under your composite application, and select `Edit` from the popup menu.**

  The CASA Editor opens containing your composite application.

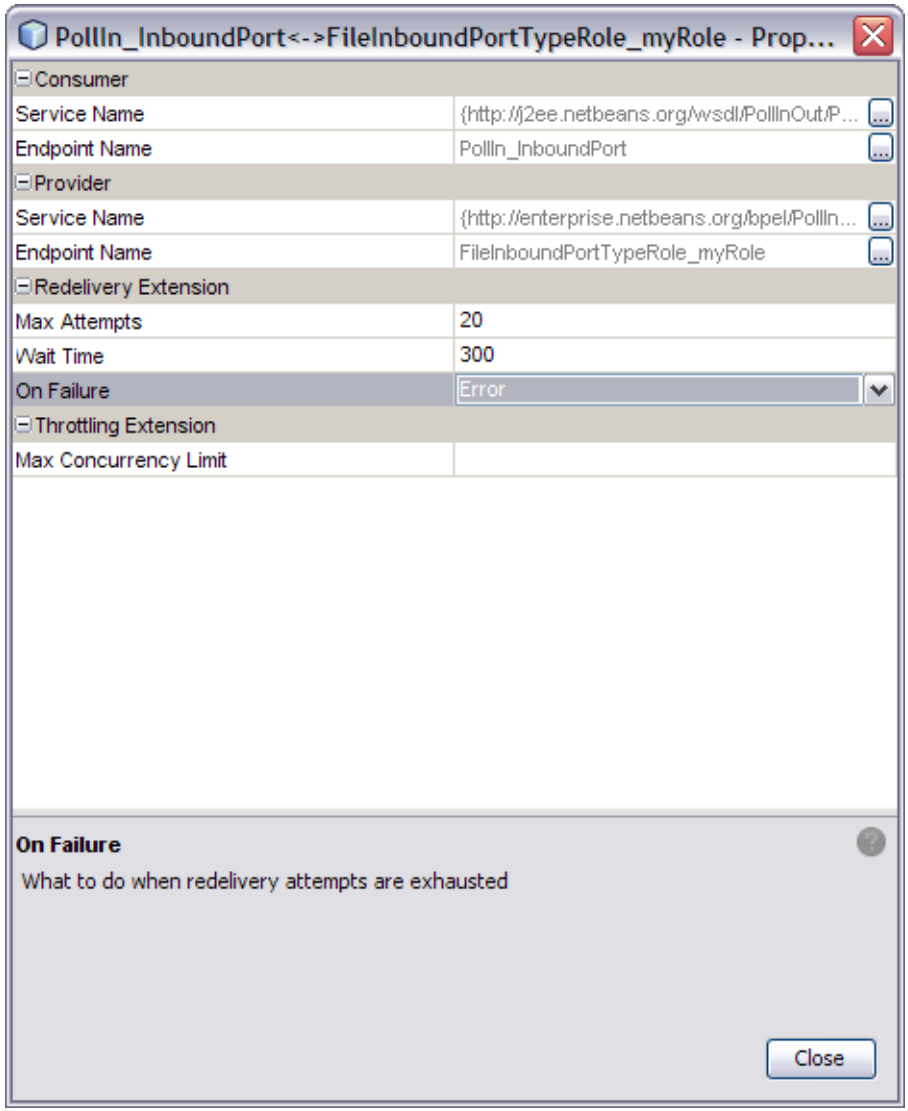


2 **In the CASA Editor, click the `QoS` icon located on the link between your JBI Module and the WSDL port you want to configure.**

The QOS Properties Editor appears.

3 **In the QOS Properties Editor, click the property field for Max Attempts under Redelivery Extension, and enter an integer for the maximum number of redeliveries to be attempted.**

4 **Similarly set the `Wait Time` and `On Failure` attributes and click `Close`.**



Endpoints Statistics and Monitoring Management

Endpoints statistics feature provides the File Binding Component's statistics data for endpoints including active consuming endpoints, provisioning endpoints, the requests/response these endpoints sent/received and so forth.

The File Binding Component records and maintains statistics for 19 different component activities including exchanges, errors, requests, replies, and so forth. These statistics are

recorded during the lifecycle of an endpoint, and accessed from the File Binding Component Properties Editor. For example: statistics for the number of times that a send request has been completed are available in the application's File Binding Component properties as the current value for `Statistics`→ `Sent Requests`.

The File Binding Component's performance measurements are always set to "on". The supported categories for performance measurements are "Normalization" and "Denormalization". The performance measurement data is returned as a javax.management.openmbean.TabularDat.

Throttling and Serial Processing

Throttling allows you to set the maximum number of concurrent messages that are processed by a particular endpoint. Increased message load and large message payloads can cause memory usage spikes that can decrease performance. Throttling limits resource consumption so that consistent performance is maintained.

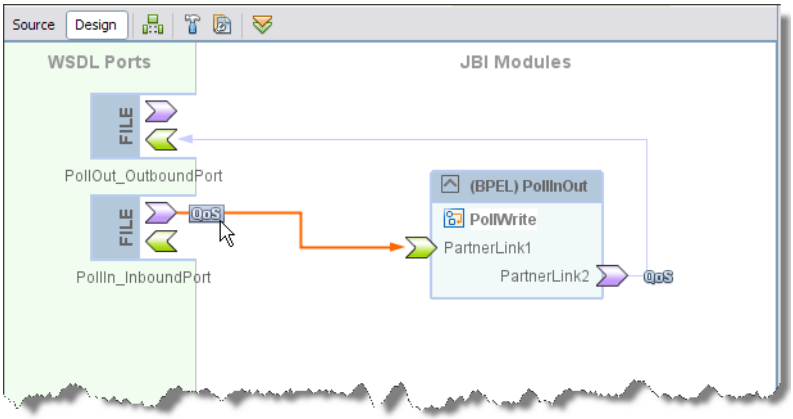
When Max Concurrency Limit is set to 1, File Binding Component will process messages in a serial fashion. That is, after sending one inbound message, next message will be sent only after a response/acknowlegement is received for the first message.

▼ To Configure Throttling

1 **From the NetBeans IDE Projects window, right-click the Service Assembly node under your composite application, and select `Edit` from the pop-up menu.**

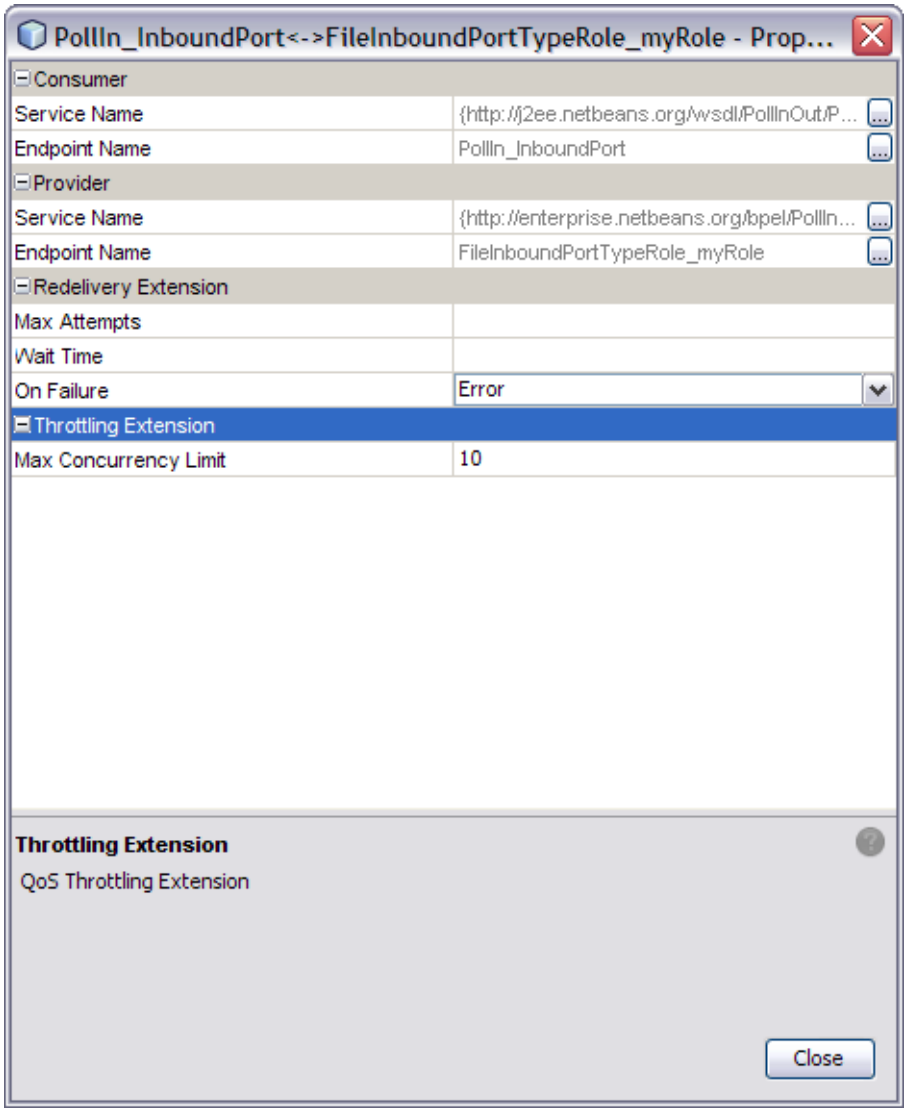
The CASA Editor opens containing your composite application.

2 **In the CASA Editor, click the `QoS` icon located on the link between your JBI Module and the WSDL port you want to configure.**



The QOS Properties Editor appears.

3 **In the QOS Properties Editor, click the property field for `Max Concurrency Limit` under Throttling Extension, and enter an integer for the maximum number of concurrent messages allowed for this endpoint.**



4 **Click** `Close`.

The appropriate throttling configuration for the connection is generated in the project's jbi.xml file, when the service assembly is built.