Configuring JBI Components

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

Part No: 820–3490–10
June 2008

080619@20490

Contents

1 **Configuring JBI Components** 5
Configuring WSDL Extensibility Elements 5
HTTP WSDL Extensibility Elements 6
HTTP Connectivity 6
HTTP Binding 7
SOAP WSDL Extensibility Elements 10
SOAP Connectivity 10
SOAP Binding 10
Configuring JBI Runtime Properties 17
Configuring the BPEL Service Engine Runtime Properties 18
Starting the Application Server 18
Viewing Service Engine Properties 18
Runtime Property Descriptions 18
HTTP Binding Component Runtime Properties 21
Configuring BPEL Service Engine Clustering and Failover 26
Clustering 26
Failover 26
Clustering/Failover Considerations 27
Configuring the HTTP Binding Component for Clustering 27

CHAPTER 1

Configuring JBI Components

This topic covers the following information:

Configuring WSDL Extensibility Elements

- "Configuring WSDL Extensibility Elements" on page 5
- "HTTP WSDL Extensibility Elements" on page 6
- "SOAP WSDL Extensibility Elements" on page 10

Configuring JBI Component Runtime Properties

- "Configuring JBI Runtime Properties" on page 17
- "Configuring the BPEL Service Engine Runtime Properties" on page 18
- "HTTP Binding Component Runtime Properties" on page 21

Configuring JBI Components for Clustering

- "Configuring BPEL Service Engine Clustering and Failover" on page 26
- "Configuring the HTTP Binding Component for Clustering" on page 27

Configuring WSDL Extensibility Elements

WSDL elements include Service (Connectivity) and Binding WSDL elements. Service elements allow you to establish a connection with a server. Binding elements allow you to convert WSDL messages to and from HTTP/SOAP messages. Both the binding and service section of a WSDL document must be properly filled out to define how a message is transformed and the destination of that message.

Each binding components contain protocol-specific information, defined by WSDL Extensibility Elements. These elements can be edited using the NetBeans IDE WSDL Editor, from the editors Source view, WSDL view, or Partner view. The WSDL Editor supports version 1.1 of the WSDL specification.

For information on using the WSDL Editor, see the Developer Guide to the WSDL Editor. You can also try a tutorial created specifically for the binding component you wish to understand better, for example, the Processing an Order in a Purchase Order Systemtutorial illustrates using the HTTP Binding Component and BPEL Service Engine in an application.

This section contains information on the following WSDL Extensibility Elements:

- "HTTP WSDL Extensibility Elements" on page 6
- "SOAP WSDL Extensibility Elements" on page 10

HTTP WSDL Extensibility Elements

The HTTP extensibility elements describe interactions between a web browser and a web site through the HTTP 1.1 GET and POST verbs. This allows applications other than web browsers to interact with the site. The HTTP WSDL elements are categorized under "HTTP Connectivity" on page 6 or "HTTP Binding" on page 7.

The following protocol-specific information may be specified:

- An indication that a binding uses HTTP GET or POST
- An address for the port
- A relative address for each operation (relative to the base address defined by the port)

The information here is broken up into two main sections: how the user may configure an endpoint to an HTTP service, and how the user specifies configuration to map WSDL messages to HTTP messages

This section contains information on the following WSDL Extensibility Elements:

- "HTTP Connectivity" on page 6
- "HTTP Binding" on page 7

HTTP Connectivity

HTTP Connectivity elements consist of the address element.

The HTTP Address Element

The HTTP `address` extensibility element allows the user to specify connectivity information to the HTTP server.

TABLE 1–1 The HTTP Address Element Attributes

Property	Description	Required or Optional	Example
location	A URL which specifies the connectivity information to connect to the HTTP server.	Required	http://myhost:7676/some/additiona

The following example illustrates the use of the HTTP `address` extensibility element defined for a service port:

```
<port binding="y:binding" name="soapEndpoint">
   <http:address location="http://myhost:7676/some/additional/context" />
</port>
```

HTTP Binding

The HTTP extensibility elements for binding abstract WSDL messages to HTTP messages fall into several sections. Each section signifies how the binding should occur. At the binding level, the configuration applies to the entire port type. At the operation level, the configuration applies only to the operation. At the message level, the configuration applies to that particular message, regardless of whether it is input or output.

The HTTP Binding Element

The purpose of the HTTP `binding` element is to signify that the binding is bound to the HTTP protocol.

TABLE 1–2 The HTTP Binding Element Attributes

Property	Description	Required or Optional	Example
verb	Indicates to which transport of HTTP this binding corresponds.	Required	GET

The HTTP `binding` element MUST be present when using the HTTP binding. The following example illustrates the HTTP binding element:

```
<definitions .... >
    <binding .... >
        <http:binding verb="nmtoken" />
    </binding>
</definitions>
```

The value of the required **verb attribute** indicates the HTTP verb. Common values are GET or POST, but others may be used. Note that HTTP verbs are case sensitive.

The HTTP Operation Element

The purpose of the HTTP `operation` element is to provide binding information from the abstract operation to the concrete HTTP operation.

TABLE 1–3 The HTTP Operation Element Attributes

Property	Description	Required or Optional	Example
location	Indicates the relative URI. Combined with the address location attribute.	Required	o1

The following example illustrates the WSDL operation element:

```
<definitions .... >
    <binding .... >
        <operation .... >
           <soap:operation location="uri" />
        </operation>
    </binding>
</definitions>
```

The **location attribute** specifies a relative URI for the operation. This URI is combined with the URI specified in the http:address element to form the full URI for the HTTP request. The URI value MUST be a relative URI.

The HTTP urlEncoded Element

The `urlEncoded` element indicates that all of the message parts are encoded into the HTTP request URI using the standard URI-encoding rules (name1=value&name2=value...). The names of the parameters correspond to the names of the message parts. Each value contributed by the part is encoded using a name=value pair. This may be used with GET to specify URL encoding, or with POST to specify a FORM-POST. For GET, the “?” character is automatically appended as necessary.

Example:

```
<definitions .... >
    <binding .... >
        <operation .... >
           <input .... >
               <http:urlEncoded/>
           </input>
```

```
            <output .... >
                <-- mime elements -->
            </output>
         </operation>
     </binding>
</definitions>
```

The HTTP urlReplacement Element

The `urlReplacement` element indicates that all the message parts are encoded into the HTTP request URI using a replacement algorithm:

- The relative URI value of http:operation is searched for a set of search patterns
- The search occurs before the value of the http:operation is combined with the value of the location attribute from http:address
- There is one search pattern for each message part. The search pattern string is the name of the message part surrounded with parenthesis "(" and ")"
- For each match, the value of the corresponding message part is substituted for the match at the location of the match
- Matches are performed before any values are replaced (replaced values do not trigger additional matches)

Message parts MUST NOT have repeating values.

The HTTP urlReplacement Elements

Example:

```
<definitions .... >
    <binding .... >
        <operation .... >
           <input .... >
               <http:urlReplacement/>
           </input>
           <output .... >
               <-- mime elements -->
           </output>
        </operation>
    </binding>
</definitions>
```

SOAP WSDL Extensibility Elements

The SOAP WSDL elements allow you to configure two sets of information for the HTTP Binding Component: "SOAP Connectivity" on page 10 information and "SOAP Binding" on page 10 information, to convert WSDL messages to and from SOAP messages.

This section contains information on the following WSDL Extensibility Elements:

- "SOAP Connectivity" on page 10
- "SOAP Binding" on page 10

SOAP Connectivity

Soap Connectivity elements consist of the address element.

The SOAP Address Element

The SOAP `address` extensibility element allows the user to specify the connectivity information to the SOAP server.

The SOAP Address Element

The SOAP `address` extensibility element allows the user to specify the connectivity information to the SOAP server.

TABLE 1–4 The SOAP Address Element Attributes

Property	Description	Required or Optional	Example
location	A URL which specifies the connectivity information used to connect to the SOAP server.	Required	http://myhost:7676/some/additional/conte

The following example illustrates the use of the SOAP address element:

```
<port binding="y:binding" name="soapEndpoint">
   <soap:address location="http://myhost:7676/some/additional/context" />
</port>
```

SOAP Binding

The SOAP extensibility elements for binding abstract WSDL messages to SOAP messages fall into several sections. Each section signifies how the binding should occur. At the binding level, the configuration applies to the entire port type. At the operation level, the configuration applies only to the operation. At the message level, the configuration applies to that particular message, regardless of whether the message is input or output.

The SOAP Binding Element

The purpose of the SOAP `binding` element is to indicate that the binding is bound to the SOAP protocol format: Envelope, Header and Body. This element makes no claim as to the encoding or format of the message (e.g. that it necessarily follows section 5 of the SOAP 1.1 specification).

TABLE 1–5 The SOAP Binding Element Attributes

Property	Description	Required or Optional	Example
transport	Indicates to which transport of SOAP this binding corresponds.	Optional	http://schemas.xmlsoap.org/soap/ht
style	Indicates the default style of this particular SOAP binding.	Optional	rpc

The SOAP `binding` element MUST be present when using the SOAP binding. The following example illustrates the use of the SOAP binding element:

```
<definitions .... >
    <binding .... >
        <soap:binding transport="uri"? style="rpc|document"?>
    </binding>
</definitions>
```

The **style attribute** value is the default style attribute for each contained operation. If the style attribute is omitted, it is assumed to be "document".

The value of the required **transport attribute** indicates the transport to use to deliver SOAP messages. The URI value http://schemas.xmlsoap.org/soap/http corresponds to the HTTP binding in the SOAP specification. Other URIs may be used here to indicate other transports (such as SMTP, FTP, and so forth).

The SOAP Operation Element

The purpose of the SOAP `operation` element is to provide binding information from the abstract operation to the concrete SOAP operation.

TABLE 1–6 The SOAP Operation Element Attributes

Property	Description	Required or Optional	Example
soapAction	Indicates the soapAction that should be put into the HTTP header.	Optional	urn:someSoapAction
style	Indicates the default style of this particular SOAP operation.	Optional	rpc

The following example illustrates the use of the SOAP operation element:

```
<definitions .... >
    <binding .... >
        <operation .... >
           <soap:operation soapAction="uri"? style="rpc|document"?>?
        </operation>
    </binding>
</definitions>
```

The **style attribute** indicates whether the operation is RPC-oriented (messages containing parameters and return values) or document-oriented (messages containing documents). This information may be used to select an appropriate programming model. The value of this attribute also affects the way in which the body of the SOAP message is constructed. If the attribute is not specified, it defaults to the value specified in the soap:binding element. If the soap:binding element does not specify a style, it is assumed to be "document".

The **soapAction attribute** specifies the value of the `SOAPAction` header for this operation. This URI value should be used directly as the value for the `SOAPAction` header. No attempt should be made to make a relative URI value absolute when making the request. For the HTTP protocol binding of SOAP, this value is required (it has no default value). For other SOAP protocol bindings, it MUST NOT be specified, and the soap:operation element can be omitted.

The SOAP Body Element

The purpose of the SOAP `body` element is to provide binding information from the abstract operation to the concrete SOAP operation.

TABLE 1–7 The SOAP Body Element Attributes

Property	Description	Required or Optional	Example

TABLE 1–7 The SOAP Body Element Attributes *(Continued)*

parts	Indicates the parts from the WSDL message that will be included in the body element.	Optional	part1
use	Indicates how message parts are encoded in the SOAP body.	Optional	literal
encodingStyle	Indicates a particular encoding style to use.	Optional	http://someencodingstyle
namespace	Indicates the namespace of the wrapper element for RPC style messages.	Optional	urn:someNamespace

The following example illustrates the SOAP body element:

```
<definitions .... >
    <binding .... >
        <operation .... >
           <input>
               <soap:body parts="nmtokens"? use="literal|encoded"?
                          encodingStyle="uri-list"? namespace="uri"?>
           </input>
           <output>
               <soap:body parts="nmtokens"? use="literal|encoded"?
                          encodingStyle="uri-list"? namespace="uri"?>
           </output>
        </operation>
    </binding>
</definitions>
```

The optional **parts attribute** of type `nmtokens` indicates which parts appear somewhere within the SOAP Body portion of the message (other parts of a message may appear in other portions of the message, such as when SOAP is used in conjunction with the multipart/related MIME binding). If the parts attribute is omitted, then all parts defined by the message are assumed to be included in the SOAP Body portion.

The required **use attribute** indicates whether the message parts are encoded using some encoding rules, or whether the parts define the concrete schema of the message.

If `use` is encoded, then each message part references an abstract type using the `type` attribute. These abstract types are used to produce a concrete message by applying an encoding that is specified by the `encodingStyle` attribute. The part names, types and value of the `namespace` attribute are all inputs to the encoding, although the `namespace` attribute only applies to content that is not explicitly defined by the abstract types. If the referenced encoding style allows variations in its format (as does the SOAP encoding), then all variations MUST be supported ("reader makes right").

If `use` is literal, then each part references a concrete schema definition using either the `element` or `type` attribute. In the first case, the element referenced by the part will appear directly under the Body element (for document style bindings) or under an accessor element named after the message part (in RPC style). In the second, the type referenced by the part becomes the schema type of the enclosing element (Body for document style or part accessor element for RPC style).

For an example that illustrates "defining the contents of a composite Body using a type", see section 2.3.1. The value of the encodingStyle attribute may be used when the use is literal, to indicate that the concrete format was derived using a particular encoding (such as the SOAP encoding), but that only the specified variation is supported ("writer makes right").

The value of the **encodingStyle attribute** is a list of URIs, each separated by a single space. The URI's represent encodings used within the message, in order of most restrictive to least restrictive (exactly like the encodingStyle attribute defined in the SOAP specification).

The SOAP Fault Element

The `fault` element specifies the contents of SOAP Fault Details element. It is patterned after the body element.

TABLE 1–8 The SOAP Fault Element Attributes

Property	Description	Required or Optional	Example
name	Indicates the name of the part from the WSDL message that will be included in the fault element.	Required	part1
use	Indicates how message parts will be encoded in the SOAP fault.	Required	literal
encodingStyle	Indicates a particular encoding style to use.	Optional	http://someencodingstyle
namespace	Indicates the namespace of the wrapper element for RPC style messages.	Optional	urn:someNamespace

The following example illustrates the SOAP fault element:

```
<definitions .... >
    <binding .... >
        <operation .... >
           <fault>*
               <soap:fault name="nmtoken" use="literal|encoded"
                                 encodingStyle="uri-list"? namespace="uri"?>
           </fault>
        </operation>
    </binding>
</definitions>
```

The **name attribute** relates the soap:fault to the wsdl:fault defined for the operation. The fault message MUST have a single part.

The **use**, **encodingStyle** and **namespace attributes** are all used in the same way as with Body, only style="document" is assumed, since faults do not contain parameters.

The SOAP Header and Headerfault Elements

The `header` and `headerfault` elements allow headers to be defined that are transmitted inside the Header element of the SOAP Envelope. It is not necessary to exhaustively list all headers that appear in the SOAP Envelope using header. For example, extensions to WSDL may imply specific headers should be added to the actual payload and it is not required to list those headers here.

TABLE 1–9 The SOAP Header Element Attributes

Property	Description	Required or Optional	Example
message	Indicates WSDL message that will be used in binding to the header element.	Required	part1
part	Indicates the parts from the WSDL message that will be included in the header element.	Required	part1
use	Indicates how message parts will be encoded in the SOAP header.	Required	literal
encodingStyle	Indicates a particular encoding style to use.	Optional	http://someencodingstyle
namespace	Indicates the namespace of the wrapper element for RPC style messages.	Optional	urn:someNamespace

TABLE 1–10 The SOAP Headerfault Element Attributes

Property	Description	Required or Optional	Example
name	Indicates WSDL message that will be used in binding to the headerfault element.	Required	part1
part	Indicates the parts from the WSDL message that will be included in the headerfault element.	Required	part1
use	Indicates how message parts will be encoded in the SOAP headerfault.	Required	literal
encodingStyle	Indicates a particular encoding style to use.	Optional	http://someencodingstyle
namespace	Indicates the namespace of the wrapper element for RPC style messages.	Optional	urn:someNamespace

The following example illustrates the SOAP header and headerfault elements:

```
<definitions .... >
    <binding .... >
        <operation .... >
           <input>
             <soap:header message="qname" part="nmtoken" use="literal|encoded"
                          encodingStyle="uri-list"? namespace="uri"?>*
               <soap:headerfault message="qname" part="nmtoken" use="literal|encoded"
                                 encodingStyle="uri-list"? namespace="uri"?/>*
             <soap:header>
           </input>
           <output>
               <soap:header message="qname" part="nmtoken" use="literal|encoded"
                            encodingStyle="uri-list"? namespace="uri"?>*
                 <soap:headerfault message="qname" part="nmtoken" use="literal|encoded"
                                   encodingStyle="uri-list"? namespace="uri"?/>*
               <soap:header>
           </output>
        </operation>
    </binding>
</definitions>
```

The **use**, **encodingStyle** and **namespace attributes** are all used in the same way as with Body, only style="document" is assumed since headers do not contain parameters.

Together, the **message attribute** (of type QName) and the **part attribute** (of type nmtoken) reference the message part that defines the header type.

The optional **headerfault elements** which appear inside the header and have the same syntax as the header, allow specification of the header types used to transmit error information pertaining to the header, and defined by the header. The SOAP specification states that errors pertaining to headers must be returned in the headers. This mechanism allows specification of the format of such headers.

Configuring JBI Runtime Properties

The JBI node, located under the Servers > *your application server*, in the Servers window, contains the JBI component files for Service Engines, Binding Components, Shared Libraries, and Service Assemblies.

To access the properties editor for the JBI components, do the following:

- From the Services tab of the NetBeans IDE, expand the Servers node.
- If your application server has not been started, right-click the application server node and select `Start` from the popup menu.
  After the application server is running, expand the application server node.
- Expand the JBI node and expand the node for the component type you want to access. For example, expand the JBI > Binding Components node.
- Properties can be accessed from the Properties Editor, or from the Properties window.

  To access the Properties Editor, right-click the component that you want to configure and select Properties from the popup menu. For example, right—click sun-http-binding to open the properties editor for the HTTP Binding Component. The Properties Editor appears.

  To use the Properties window, select a JBI component and the properties for that component are displayed in the Properties window. If the Properties window is not visible, open the Properties window by select Properties from the NetBeans Windows menu.
- Edit the properties as needed.
  When you make a change to a property, an Information dialog box appears with directions on how to apply the changes to your application. For example, you may need to restart the application that uses this component before the changes take effect.

This section contains information on the following JBI components:

- “Configuring the BPEL Service Engine Runtime Properties” on page 18
- “HTTP Binding Component Runtime Properties” on page 21

Configuring the BPEL Service Engine Runtime Properties

Changes to the BPEL Service Engine runtime properties are made from within NetBeans.

This section contains information on the following topics for the BPEL Service Engine runtime properties:

- "Starting the Application Server" on page 18
- "Viewing Service Engine Properties" on page 18
- "Runtime Property Descriptions" on page 18

Starting the Application Server

Configuration of BPEL Service Engine runtime properties requires first starting the Sun Java System Application Server in the NetBeans IDE Services window.

To start the Sun Java System Application Server:

1. Select the Services window.
2. Right-click the Servers > Sun Java System Application Server 9 node and select Start.

Viewing Service Engine Properties

After you have started the application server, you can access the service engine properties.

To view BPEL Service Engine properties:

1. In the Services window, expand the following nodes: `Sun Java System Application Server 9` > `JBI` > `Service Engines`.
2. Right-click `sun-bpel-engine` and select Properties.

   The `sun-bpel-engine` Properties window appears.

Runtime Property Descriptions

The following tables include descriptions for the BPEL Service Engine runtime properties

TABLE 1–11 General Properties

Property Name	Description	Default Value
Description	Description of the JBI Component.	This is a bpel service engine.

TABLE 1–11 General Properties *(Continued)*

Name	Name of the JBI Component. Specifies a unique name in the JBI environment. If you are installing more than one BPEL Service Engine in a JBI environment, make sure that each is unique. This can be changed in the descriptor (jbi.xml) for the component. When the service unit deploys the component, it is matched with target component name defined in its descriptor – jbi.xml.	sun-bpel-engine
State	State of the JBI Component. Started, Stopped, or Shutdown.	Started
Type	Type of the JBI Component (service-engine or binding-component)	service-engine

TABLE 1–12 Identification Properties

Property Name	Description	Default Value
Build Number	Date and time stamp for the current build.	<build_number>
Spec Version	BPEL specification fully supported by this build.	<spec_version>

TABLE 1–13 Configuration Properties

Property Name	Description	Default Value
DatabaseJNDIName	Attribute exposed for management. Enter the database JNDI name. For more information, see PersistenceEnabled.	jdbc/bpelseDB
DebugEnabled	Attribute exposed for management. When set to true, you can attach the debugger and debug the business process flow.	false

TABLE 1–13 Configuration Properties *(Continued)*

DebugPort	Attribute exposed for management. Specifies the port number at which the debug server listener listens for the debugger UI to connect. The default value is sufficient for most cases. Make sure that this port is not used by other applications. If you are running more than one instance of BPEL Service Engine on this computer, make sure that you assign non conflicting unique ports.	3343
EngineExpiryInterval	Attribute exposed for management. An integer that specifies the frequency (in seconds) that the BPEL Service Engine registers itself as available. Use this attribute primarily for configuring BPEL Service Engine failover. Set this attribute so that it registers itself as available frequently enough to meet the demands of your system. Optimizing this attribute might require some testing. This attribute also applies to the interval for the recovery of dangling instances. The default setting is 15.	15
MaxThreadCount	Attribute exposed for management. An integer that specifies the number of threads executing BPEL definitions. This is the number of computers needed for maximum throughput. Note that you must set associated binding components with same number of threads. This combination is the optimal configuration for most cases.	10
PersistenceEnabled	Attribute exposed for management. When set to true, the BPEL Service Engine persists critical data to a database. Upon restart after a system failure, the BPEL Service Engine resumes processing from the point at which service engine processing was interrupted.	false

TABLE 1–14 Class Loggers

Class Name	Description	Default Logging Level
sun-bpel-engine		INFO
BPELSEDeployer		INFO
BPELSEHelper		INFO
BPELSEInOutThread		INFO
BPELSELifeCycle		INFO
BPELSERuntimeConfiguration		INFO
DeploymentBindings		INFO
EngineChannel		INFO
BPELInterpreter		INFO
EngineImpl		INFO

HTTP Binding Component Runtime Properties

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

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

TABLE 1–15 HTTP Binding Component Runtime Properties

Property	Description	Required or Optional	Example
General Properties			
Description	Indicates the purpose of the HTTP Binding Component. This property is displayed for reference purposes.	Automatic	HTTP Soap Binding to send SOAP messages, for example, to and from BPEL service engine.
Name	Indicates the name of the HTTP Binding Component. This property is displayed for reference purposes.	Automatic	com.sun.httpsoapbc-1.0-2

TABLE 1–15 HTTP Binding Component Runtime Properties *(Continued)*

State	Indicates the state of the HTTP Binding Component as "Started" or "Stopped." This property is displayed for reference purposes.	Automatic	Started
Type	Indicates the type of component. This property is displayed for reference purposes.	Automatic	binding-component
Identification Properties			
SpecVersion	Indicates the component specification version.	Static	1.0
Build Number	Indicates the component build number.	Static	080311_4
Configuration Properties			
Default HTTP Port Number	Specifies the default HTTP port number for the HTTP Binding Component instance. This property is required for clustering and allows each HTTP Binding Component to be differentiated by its unique default port number. A default port number is calculated and preassigned when the binding component is initially installed in the application server instance. A file containing the persisted configuration is stored for each component. This is used to assign a unique default port number for each HTTP Binding Component instance on a computer.	Required	8180
Default HTTPS Port Number	Specifies the default HTTP Secure port number for the HTTP Binding Component instance. This property is required for clustering and allows each HTTP Binding Component to be differentiated by its unique default port number. A default port number is calculated and preassigned when the binding component is initially installed in the application server instance. A file containing the persisted configuration is stored for each component. This is used to assign a unique default port number for each HTTP Binding Component instance on a computer.	Required	8280
Sun Access Manager Configuration Directory	Specifies the location of the Sun Access Manager configuration directory, which contains the Access Manager properties file.	Optional	

TABLE 1–15 HTTP Binding Component Runtime Properties *(Continued)*

ProxyType	Specifies the proxy type as SOCKS, HTTP, or DIRECT. Enter one of the following String values: SOCKS The proxy server is a SOCKS (version 4 or version 5) server. HTTP The proxy is an HTTP proxy server. DIRECT The connection does not go through any proxy.	Required	SOCKS
ProxyHost	Specifies the proxy host name or IP address.	Optional	polaris.sun.com
ProxyPort	Specifies the proxy port number.	Required	2080
NonProxyHosts	Specifies the list of hosts that you do not want to go through the proxy. Each host is separated with a pipe "|".	Optional	localhost|127.0.0.4
ProxyUserName	Specifies the user name used to access the proxy server. For SOCKS-v4, no authentication is required. For SOCKS-v5, the binding component supports no authentication, and Username/Password authentication. For HTTP Proxy, the binding component supports Basic Authentication, Digest Access, and NTLM. Basic Authentication requires a specified username and password. Digest Access and NTLM require a dedicated proxy server for support.	Required in some cases	
ProxyPassword	Specifies the password used in conjunction with the the ProxyUserName to access the proxy server. For SOCKS-v4, no authentication is required. For SOCKS-v5, the binding component supports no authentication, and Username/Password authentication. For HTTP Proxy, the binding component supports Basic Authentication, Digest Access, and NTLM. Basic Authentication requires a specified username and password. Digest Access and NTLM require a dedicated proxy server for support.	Required in some cases	

TABLE 1–15 HTTP Binding Component Runtime Properties *(Continued)*

Use Default JVM Proxy Settings	Indicates whether the HTTP Binding Component's proxy settings are specified by the existing JVM settings or by the HTTP Binding Component properties. The options indicate the following: true The proxy is controlled by the existing JVM system settings. The settings are outside of this binding component, so all additional proxy settings are ignored. false The proxy is controlled by the binding component proxy settings.	Required	false
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. 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.	Optional	*The user-defined name of the ConfigExtension you want and define, and the values for the connection parameters.*

TABLE 1–15 HTTP Binding Component Runtime Properties *(Continued)*

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.	Optional	*Enter the name value, such as* PASSWORD, *and enter the variable Value, such as* SECRET. *For Boolean values, the Value field provides a checkbox (checked = true).* *For Password values, the Value entered is masked as asterisks.*
Statistics Properties			
Includes 19 different component activities: Activated Endpoints, Active Exchanges, Avg. Component Time, Avg. D.C. Time, Avg. Msg. Service Time, Avg. Response Time, Completed Exchanges, Error Exchanges, Received Dones, Received Errors, Received Faults, Received Replies, Received Requests, Sent Dones, Sent Errors, Sent Faults, Sent Replies, Sent Requests, Up Time.	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.	Automatic	240
Loggers Properties			

TABLE 1–15 HTTP Binding Component Runtime Properties *(Continued)*

Includes over 30 different component activities that can be recorded by the server.log.	Specifies the level of logging for each event. There are eight levels of logging, FINEST (most detailed), FINER, FINE, CONFIG, INFO, WARNING, SEVERE (failure messages only), and OFF.	Optional	WARNING

Configuring BPEL Service Engine Clustering and Failover

The BPEL Service Engine supports clustering and failover in order to optimize and ensure business process throughput on highly scalable systems. Clustering distributes processing over multiple BPEL server engines via multiple BPEL service units. Failover prevents processing from being interrupted by picking up business processes from any failed systems and processing them to completion.

This section contains the following topics for configuring BPEL Service Engine clustering and failover:

- "Clustering" on page 26
- "Failover" on page 26
- "Clustering/Failover Considerations" on page 27

Clustering

When a business process needs to be scaled to meet heavier processing needs, you can distribute it across multiple service engines to increase throughput. The BPEL Service Engine's clustering algorithm automatically distributes processing across multiple engines. To configure clustering, deploy your composite applications across multiple BPEL Service Engines running on multiple processors or systems.

For details about setting up a cluster of application servers with BPEL Service Engines, see the documentation for Sun Java System Application Server.

Failover

When your business process is configured for clustering, the BPEL Service Engine's failover capabilities ensure throughput of running business process instances. When business process instances encounter an engine failure, any suspended instances are picked up by all available BPEL Service Engines.

To configure failover, in the BPEL Service Engine properties, set the `EngineExpiryInterval` property so that it registers itself as alive frequently enough to meet the demands of your system. Optimizing this property setting might require some testing. The default setting is 15.

Clustering/Failover Considerations

In order to configure a cluster of BPEL Service Engines, you must adhere to the following guidelines.

- Persistence must be enabled for both clustering and failover.
- To run persistence, all BPEL Service Engines must be restarted.
- Service assemblies must be deployed manually across all clustered JBI environments.
- Clustering/failover is implemented consistently for the specific protocol and binding components involved in a given business process.
- Only a single database can be used for all BPEL Service Engines when implementing clustering/failover.
- The database must be highly available; should the database fail, clustering/failover will fail.
- When a BPEL Service Engine fails, a single BPEL Service Engine picks up those instances without distributing them across the cluster. Consequently, a large number of failed over instances can overload an entire cluster, one service engine at a time—a sort of domino effect.
- All BPEL Service Engines in a cluster must reside in the same time zone.
- The following inbound message activities (within flows) are not supported by failover: `Receive`, `On-Message`, `On-Event`.

Configuring the HTTP Binding Component for Clustering

For the most part, configuring the HTTP Binding Component for clustering is handled by Sun Java System Application Server. The HTTP Binding Component is a pre-installed component in the application server. Default HTTP and HTTPS port numbers are calculated and pre-assigned when the binding components are installed in the server instances. A web service, serviced by an HTTP Binding Component, is identified by a unique URL identifier with this structure: "`http://<hostname>:<port>/<context>` ".

Each component instance in the cluster must have exclusive access to the resource, therefore a unique port number is assigned to each component instance. A predefined token name is used in the WSDL artifact to resolve the actual port value when the component is deployed into each instance.

Predefined token names:

- "${HttpDefaultPort}" for the HTTP port
- "${HttpsDefaultPort}" for the HTTPS port

These token names are used in lieu of a real port number in the endpoint URL (soap:address) to allow the application client to direct HTTP requests to the default port. The value of the token is then resolved by the HTTP Binding Component, based on the configured default values when an application is deployed.

Note – If you reinstall an HTTP Binding Component, you must reconfigure the default ports properly for each component instance.
