Configuring GlassFish ESB for Clustering

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

Part No: 820–7848–11
June 2009

090610@22510

Contents

Configuring GlassFish ESB for Clustering

The topics listed here provide procedures, conceptual information, and reference information for configuring GlassFish ESB components in a clustered environment.

What You Need to Know

These topics provide information you should know about clustering.

What You Need to Do

These topics provide instructions on how to set up a cluster and how to configure GlassFish ESB for clustering.

More Information

This topic provides additional information you should know about GlassFish server, Java System Message Queue (JMQ), and Java Message Service (JMS) clusters.

- *Sun GlassFish Enterprise Server 2.1 High Availability Administration Guide*
- Chapter 5, "Configuring HTTP Load Balancing," in *Sun GlassFish Enterprise Server 2.1 High Availability Administration Guide*.
- Article on GlassFish Clustering (for version 2.0)
- Chapter 8, "Broker Clusters," in *Sun Java System Message Queue 4.1 Administration Guide*
- Chapter 10, "Java Message Service Load Balancing and Failover," in *Sun GlassFish Enterprise Server 2.1 High Availability Administration Guide*

Clustering in GlassFish ESB

GlassFish ESB components rely on the clustering capabilities of GlassFish Enterprise Server (ES) for clustering and high-availability. In addition, most components can be configured for different levels of load-balancing and failover. This document describes how GlassFish ESB uses the clustering capabilities of GlassFish, but does provide detailed information about GlassFish clustering. A section is provided on basic cluster setup. For detailed information about high availability and clustering in GlassFish, see the *Sun GlassFish Enterprise Server 2.1 High Availability Administration Guide*.

The following topics provide overview information about clustering, GlassFish ESB clustering, and the steps required to set up a cluster for GlassFish ESB.

- "Clustering Overview" on page 6
- "Component Support for Clustering" on page 7
- "Clustering Setup Summary" on page 7

Clustering Overview

A cluster is a collection of application server instances that can distribute processing among clustered application instances. This provides optimal performance and scalability, along with high availability and failover capabilities for reliability. Implementing GlassFish ESB applications in a clustered environment ensures continuous processing even when there is a hardware or software failure.

The instances in a cluster share the same set of GlassFish ESB applications, resources, and configuration information, and can be implemented on one server or can extend over multiple servers. A clustered instance belongs to only one cluster, through there can be multiple clusters on a domain. All settings for a cluster are defined in a single configuration file named *cluster_name*`-config`.

In a cluster configuration for GlassFish ESB components, each component is added to a GlassFish cluster and any Service Assemblies (SA) are deployed to the cluster. Adding a component to a cluster adds the component to all instances of the cluster. Similarly, deploying an SA to a cluster deploys it to all the cluster instances.

Component Support for Clustering

All GlassFish ESB components are supported in a clustered environment. Some components work inherently in a clustered environment and some require special configuration, notably in the areas of load balancing and failover. Stateful GlassFish ESB service engines (SEs), such as BPEL SE and IEP SE, use a persistence database to track cluster instances and distribute work. Other components, such as the Scheduler BC, Data Mashup SE, and LDAP BC, work inherently in a clustered environment without needing to know about the different instances.

The BPEL SE and IEP SE provide additional support for clustering at a component level for backwards compatibility. This means that multiple SE instances can be deployed on one GlassFish server in a non-clustered environment. This allows for persistence, high-availability, and failover without using a GlassFish server cluster for all components.

Load Balancing and Failover

Load balancing and failover functions depend on the individual components in the cluster. For example, load balancing is not applicable to Service Engines (SE), and is protocol specific for some of the Binding Components (BC). For example, the HTTP Load Balancer is required for SOAP transactions, while the File BC, JMS BC, and FTP BC all have built-in load balancing.

GlassFish provides additional support for high availability, load balancing, and failover with the HTTP Load Balancer plug-in. The HTTP Load Balancer distributes incoming HTTP and HTTPS transactions among the instances in a cluster and also fails over requests to another server instance if the original server instance becomes unavailable. The HTTP BC in GlassFish ESB does not have its own load balancing capabilities, making the HTTP Load Balancer plug-in a useful tool when implementing the HTTP BC in a clustered environment. For more information, see Chapter 5, "Configuring HTTP Load Balancing," in *Sun GlassFish Enterprise Server 2.1 High Availability Administration Guide*.

Load balancing and failover for the JMS BC is provided through a JMS broker cluster. For more information, see Chapter 10, "Java Message Service Load Balancing and Failover," in *Sun GlassFish Enterprise Server 2.1 High Availability Administration Guide*.

Clustering Setup Summary

Below are the general steps for setting up clustering for GlassFish ESB components. Each step is described in detail in later topics. Note that all setup and configuration tasks are performed using command line functions or the GlassFish Admin Console. NetBeans cannot be used for clustering setup because it does not support or recognize GlassFish clustering.

1. Create a GlassFish cluster, as described in "Creating a GlassFish Cluster" on page 8.
2. Add any necessary libraries to the cluster, as described in "To Add a Shared Library to a Cluster" on page 11.
3. Add the Service Engines and Binding Components to the cluster, as described in "To Add a GlassFish ESB Component to a Cluster" on page 12.
4. Make any necessary configuration changes to the Service Engines and Binding Components. This is described in individual sections for each component.
5. Deploy the Service Assembly to the cluster, as described in "Deploying a Service Assembly to a Cluster" on page 39.

Instructions for standalone (component-level) clustering are provided in the following topics:

- "Configuring the BPEL Service Engine for Multiple Standalone Instances" on page 41
- "Configuring the IEP Service Engine for Multiple Standalone Instances" on page 42

Creating a GlassFish Cluster

This topic provides general instructions for creating a cluster with GlassFish using `asadmin` commands and the GlassFish Admin Console. This procedure creates a node agent to manage the cluster, a cluster, and the cluster instances.

For complete instructions on creating and configuring GlassFish clusters, see the *Sun GlassFish Enterprise Server 2.1 High Availability Administration Guide*.

▼ To Create a GlassFish Cluster

1 If it is not already running, start the GlassFish server you want to configure for clustering.

You can start the server from the command line or from the Services tab in NetBeans.

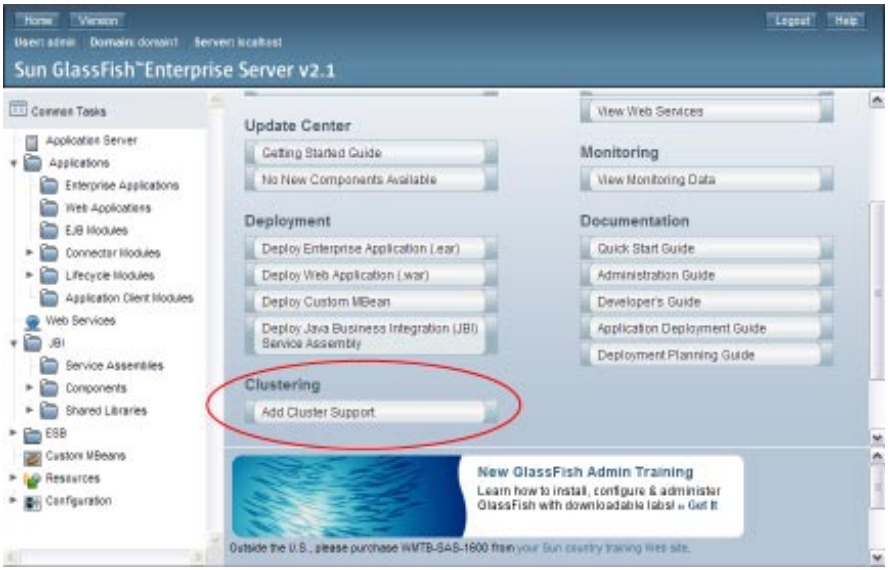
2 Once the server is started, launch the Admin Console by doing one of the following:

- **On the Services tab in NetBeans, right-click the server and select View Admin Console.**
- **Launch a web browser and enter the following URL:**

  `http://`*hostname*`:`*port*

  - *hostname* is the name of the machine where the server is located, and can be localhost.
  - *port* is the administrative port number, which is 4848 by default.

3 On the main Admin Console page, click Add Cluster Support.



4 **Review the information on the Add Cluster Support page, and then click OK.**

5 **On the Restart Required page, click Stop Instance to stop the GlassFish server.**

6 **Follow the instructions on the Admin Console to restart the server.**

7 **Run the following command to create the node agent, using a unique value for the node agent name:**

GlassFish_Home`/bin/asadmin create-node-agent` *NodeAgent_Name*

8 **Run the following command to start the node agent:**

GlassFish_Home`/bin/asadmin start-node-agent` *NodeAgent_Name*

9 **When the node agent is started, log back in to the GlassFish Admin Console and create the cluster:**

a. **In the navigation bar on the left, click Clusters.**

b. **On the Clusters page, click New.**

c. **Enter a name for the cluster, and then click OK.**

10 Add instances to the cluster:

a. In the navigation bar on the left, expand Clusters and then select the cluster you just created.

b. Click the Instances tab.

c. Under Server Instances, click New.

d. Enter a name for the instance, select the node agent you just created, and then click OK.

e. Repeat the above steps to add each instance to the cluster.

11 To start the cluster, select Clusters in the left navigation bar, select the check box next the cluster you just created, and then click Start Cluster.

12 **Click OK on the dialog box that appears.**

Tip – Log files for the cluster are located at *GlassFish_Home*`/nodeagents/`*cluster-nodeagent*`/agent/logs`.

Adding a GlassFish ESB Component to a Cluster

You add GlassFish ESB components to a cluster using the GlassFish Admin Console. The process for adding a component to a GlassFish cluster is the same for most components, though some components require additional configuration once they are added. Most components are dependant on one or more shared libraries, which need to be added to the cluster first.

Note – GlassFish ESB components can only be added to a cluster and configured for clustering using the GlassFish Admin Console. NetBeans does not currently recognize or support GlassFish clustering, so any runtime configuration changes made on NetBeans are not seen by the cluster.

▼ To Add a Shared Library to a Cluster

Before You Begin Make sure the cluster and all server instances are created and running. To verify they are running, select Nodeagent in the left navigation bar on the Admin Console and review the cluster information.

1 **Launch the GlassFish Admin Console.**

2 **In the navigation bar on the left, expand JBI and then expand Shared Libraries.**

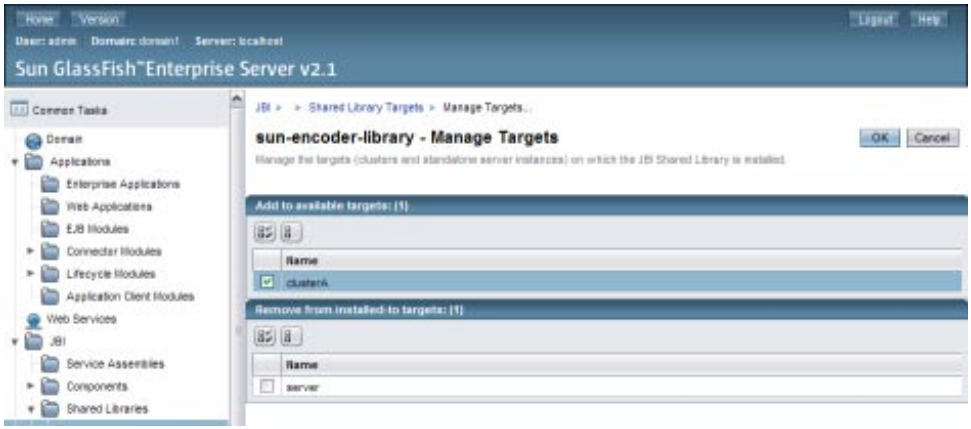
3 **Select the name of the shared library to add.**

The Properties page appears.

4 **Click the Targets tab.**

5 **On the Shared Library Targets page, click Manage Targets.**

6 **Under Add to Available Targets, select the cluster to which you want to add the shared library.**



7 **If the shared library is already added to another cluster or server to which it should not be, select that cluster or server under Remove From Installed-To Targets.**

Note – Removing a shared library from the default "server" target removes the library from the NetBeans Services window.

---

8 **Click OK.**

Note – Validations are performed to ensure that no dependencies are broken when adding or removing a shared library. In order to successfully remove a shared library from a server or cluster target, you might need to shut down service engines or binding components that are currently running on that target.

▼ To Add a GlassFish ESB Component to a Cluster

Before You Begin Make sure the cluster and all server instances are created and running. To verify they are running, select Nodeagent in the left navigation bar on the Admin Console and review the cluster information. Also be sure any libraries on which the component is dependent have been added to the cluster.

Note – The BPEL SE, IEP SE, and Database BC require additional steps before and after adding them to a cluster in order to configure and create the persistence databases. In addition, most components require some configuration steps. Be sure to review the sections for each component you add before you begin. Links to each section are provided at the end of these instructions.

---

1 **Launch the GlassFish Admin Console.**

2 **In the navigation bar on the left, expand JBI and then expand Components.**

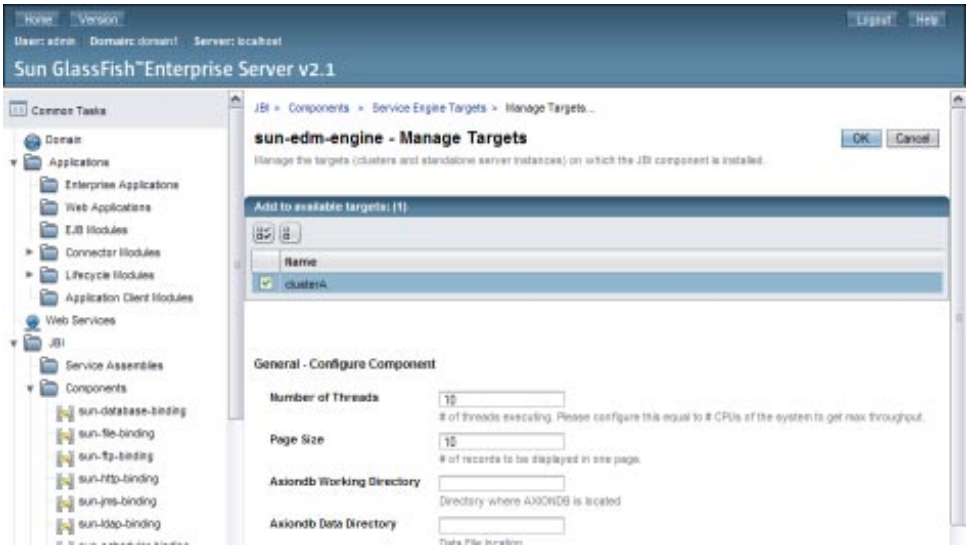
3 **Select the name of the service engine or binding component to add.**

The Properties page appears.

4 **Click the Targets tab.**

5 **On the Targets page, click Manage Targets.**

6 **Under Add to Available Targets, select the cluster to which you are adding the component.**



7 **Configure any runtime properties that appear below the target.**

8 **If the component is already added to another cluster or server that it should not be, select that cluster or server under Remove From Installed-To Targets.**

Note – Removing some components from the default "server" target removes them from the NetBeans Services window.

---

9 **Click OK.**

The Targets page reappears.

10 **In the Targets list, select the check box next to the cluster, and then click Start.**

Note – Many components require additional configuration or have certain requirements. Refer to the topic for each component for more information. Below is a link to each component:

- "Configuring the BPEL Service Engine for Clustering" on page 17
- "Configuring the IEP Service Engine for Clustering" on page 22
- "Configuring the XSLT Service Engine for Clustering" on page 25
- "Configuring the Java EE Service Engine for Clustering" on page 25
- "Configuring the Data Mashup Service Engine for Clustering" on page 26
- "Configuring the Database Binding Component for Clustering" on page 27
- "Configuring the File Binding Component for Clustering" on page 28
- "Configuring the FTP Binding Component for Clustering" on page 32
- "Configuring the HTTP Binding Component for Clustering" on page 35
- "Configuring the JMS Binding Component for Clustering" on page 37
- "Configuring the LDAP Binding Component for Clustering" on page 38
- "Configuring the Scheduler Binding Component for Clustering" on page 38

Modifying Server Properties for GlassFish ESB Components in a Cluster

Once you have added a GlassFish ESB component to a cluster, you can update the runtime properties, update the application configuration (including adding variables), view the component descriptor, set logging levels, and monitor the component. You must perform these tasks on the GlassFish Admin Console, and you need to update the configuration for each instance of that component in the cluster. Configuring runtime properties in NetBeans will not work for the cluster.

Caution – Each time you add a new instance to a cluster, you need to configure the server properties of all components on the cluster for the new instance. If you uninstall and reinstall a component or drop it from the cluster and add it back in, you need to reconfigure the component as well.

Before you perform any of these tasks, make sure the component is enabled on the cluster. Do the following to enable the cluster:

- On the GlassFish Admin Console, expand JBI and then expand Components.
- Select the name of the service engine or binding component to modify. For example, to modify the IEP SE runtime properties, select sun-iep-engine.
- Click the Targets tab.
- In the Targets list, select the check box next to the cluster and then click Start.

▼ To Modify Runtime Properties for a Component in a Cluster

1. **On the GlassFish Admin Console, expand JBI and then expand Components.**
2. **Select the name of the component to configure.**
3. **Click the Configuration tab.**
4. **In the View/Update Instance field, select the first instance to configure.**
5. **Modify any of the runtime properties.**
   The runtime properties for each component are described in that component's user guide.
6. **Click Save.**
7. **Repeat the above steps for each instance in the cluster, selecting a different instance each time until all instances are configured.**

Caution – When updating properties for a component that uses a persistence database, such as the BPEL SE, IEP SE, or Database BC, be sure that all instances of that component point to the JDBC resource for the same persistence database.

▼ To Create Application Configurations and Variables for a Component in a Cluster

Note – Not all components support application configurations or application variables. For example, the Data Mashup SE is the only service engine that supports configurations and variables.

1. **On the GlassFish Admin Console, expand JBI and then expand Components.**
2. **Select the name of the component to configure.**
3. **Click the Application tab.**
4. **To add a new configuration definition, do the following:**
   a. **In the View/Update Instance field, select the first instance to configure.**

b. **Click Add Configuration.**

c. **In the Identification section, enter a name for the configuration.**

d. **Enter values for any of the displayed properties.**

e. **Click Save.**

f. **Repeat the above steps for each instance in the cluster, selecting a different instance each time until all instances are configured.**

Caution – When updating properties for a component that uses a persistence database, such as the BPEL SE, IEP SE, or Database BC, be sure that all instances of that component point to the JDBC resource for the same persistence database.

5 **To add a variable, do the following:**

a. **Click the Variables sub-tab.**

b. **In the View/update Instance field, select the first instance to configure.**

c. **Click Add Variable.**

d. **Enter a Name, Type, and Value for the variable.**

e. **Click Save.**

f. **Repeat the above steps for each instance in the cluster, selecting a different instance each time until all instances are configured.**

▼ To View the Descriptor for a Component in a Cluster

1 **On the GlassFish Admin Console, expand JBI and then expand Components.**

2 **Select the name of the component to configure.**

3 **Click the Descriptor tab.**

4 **Review the displayed information.**

▼ To Set Logging Properties for a Component in a Cluster

1. On the GlassFish Admin Console, expand JBI and then expand Components.
2. Select the name of the component to configure.
3. Click the Loggers tab.
4. In the View/Update Instance field, select the first instance to configure.
5. Select a new logging level for any of the displayed component loggers.
6. Click Save.
7. Repeat the above steps for each instance in the cluster, selecting a different instance each time until all instances are configured.

▼ To Monitor a Component in a Cluster

1. On the GlassFish Admin Console, expand JBI and then expand Components.
2. Select the name of the component to Monitor.
3. Click the Monitoring tab.
4. In the View Instance field, select the instance to monitor.
5. View the statistics displayed for the component.
6. To view statistics for a different instance, select the instance from the View Instance field.

Configuring the BPEL Service Engine for Clustering

You can configure the BPEL SE for GlassFish clustering on the same or different machines. You can also configure the SE for high availability and failover at the component level on a standalone GlassFish server. This topic describes configuring the BPEL SE on a GlassFish cluster. For information on standalone configuration for high availability and failover, see "Configuring Components for Standalone High Availability and Failover" on page 40.

When the BPEL SE is installed and configured in a clustered environment and one engine fails, any in-process business process instances are taken over by one of the remaining engines and the process is completed. When the failed engine recovers, it continues to process new requests. Clustering for the BPEL SE leverages the persistence and recovery features of the SE, so persistence must be enabled for the BPEL SE on each cluster instance.

Failover is also supported for business processes configured for correlation. When correlated messages are processed in a clustered environment, the load balancer or binding component routes the correlating message to any BPEL SE in the cluster. If the BPEL SE to which the message was routed does not own the correlating business process instance, the instance is routed to the engine that received the correlated message (regardless of which engine began processing the initial message). Processing is then completed on the engine that received the correlated message.

The BPEL Service Engine (SE) is dependent on the following shared library, which must be added to the cluster before the BPEL SE is added to the cluster.

- Sun WSDL Extension Library

Note – The following restrictions apply to BPEL SEs installed in a cluster.

- All BPEL constructs work in a clustered environment with the exception of event handlers.
- Reply activities are only supported in a clustered environment if the service is consumed by another business process (as in a sub-process scenario).
- The servers in a cluster need to be on the same time zone in order to detect whether an instance has failed.

Setting Up the BPEL Database

In order to use the BPEL SE in a clustered environment, you need to create a database, and then create and configure two connection pools and two data sources (to handle both XA and non-XA activities). All engines in the cluster must be configured to use this database. The database should be highly available because a database failure constitutes a single point of failure in the cluster. Automatic database, connection pool, and data source creation is not available from the clustered instances.

Tip – If you already created the database tables, connection pools, and JDBC resources, you can re-use them for the cluster instances by just changing the target for the JDBC resources from the default server to your cluster (described in step 3 below).

For more information about creating connection pools and JDBC resources, see Chapter 3, "JDBC Resources," in *Sun GlassFish Enterprise Server 2.1 Administration Guide*.

▼ To Set Up the BPEL Database

Before You Begin If you are using an Oracle database, copy the driver file (`ojdbc14.jar`) from your Oracle installation to *GlassFish_Home*`/glassfish/lib`, and then stop and restart the GlassFish server.

1. **Configure the user and database as described in "Configuring the User and Database for Persistence" in *Using the BPEL Designer and Service Engine*.**

2. **Create one XA connection pool and JDBC resource and one non-XA connection pool and JDBC resource.**

   This is described in " Creating an XA Connection Pool and a JDBC Resource" in *Using the BPEL Designer and Service Engine* and " Creating a Non-XA Connection Pool and JDBC Resource" in *Using the BPEL Designer and Service Engine*.

3. **For each JDBC resource you created, return to the JDBC Resources page on the GlassFish Admin Console and do the following:**

   a. **Select the JDBC resource from the list, and then click the Target tab.**

   b. **Click Manage Targets.**

   c. **Under Available Targets, select the cluster and click Add.**

   d. **Click OK.**

   e. **In the Targets table, select the cluster and then click Enable.**

4. **To automatically create the database tables, do the following:**

   a. **In the NetBeans Services window, expand Servers > GlassFish V2 > JBI > Service Engines.**

   b. **Right-click sun-bpel-engine, and then select Start.**

   c. **On the Properties window, enable persistence and update the names of the XA and non-XA data source fields to match the JDBC resources you created above.**

   d. **Stop and start sun-bpel-engine.**

   e. **Shutdown sun-bpel-engine.**

Adding the BPEL Service Engine to the Cluster

Before you can add the BPEL SE to a cluster, you need to add a shared library. When you add the BPEL SE to a cluster, you need to configure certain properties. The SE must be configured for persistence and pointing to the correct data sources. All instances of the BPEL SE should point to a single persistence database (which you created earlier under "To Set Up the BPEL Database" on page 19).

▼ To Add the BPEL Service Engine to the Cluster

1. **Add the following shared library to the cluster as described in "To Add a Shared Library to a Cluster" on page 11:**

   - sun-wsdl-ext-library

2. **Add the BPEL SE (sun-bpel-engine) to the cluster as described in "To Add a GlassFish ESB Component to a Cluster" on page 12. Configure the following properties on the Manage Targets page before you click OK:**

   a. **Under General – Configure Component, specify a value for the Lease Renewal Interval.**

   Note – The Lease Renewal Interval is the time period in seconds that BPEL engines wait before renewing their lease to let the cluster know it is still running (also known as the *heartbeat*). If an engine does not update within the specified time period, it is considered to be unavailable.

   ---

   b. **Under Persistence – Manage Persistence/Recovery, set Persistence Enabled to `true`.**

   c. **In the Non XA Data Source Name property, enter the JNDI name of the non-XA JDBC resource that you created earlier.**

   d. **In the XA Data Source Name property, enter the JNDI name of the XA JDBC resource that you created earlier.**

      This is the database that persists state data for business process instances for recovery.

   e. **Configure the remaining runtime properties as you typically would for the BPEL SE.**

      For more information, see "Configuring the BPEL Service Engine Runtime Properties" in *Using the BPEL Designer and Service Engine*.

3. **Verify that the engines table in the persistence database contains a row for each instance in the cluster.**

Tip – If the rows do not appear, check the following:

- Both JDBC Resources are enabled on the target cluster.
- The engine is enabled on the target cluster.

If both of the above are enabled, you might need to stop and restart the BPEL SE in order for the rows to appear.

You can also verify that the engines are renewing the lease within the specified interval by querying the engines table again after the interval has passed and verifying that the LASTUPDATETIME column is updated accordingly.

Debugging a Business Process Deployed in a Cluster

When configuring debugging for a business process in a clustered environment, you need to configure each instance in the cluster for debugging and you need to set the breakpoints and attach the port number using a separate NetBeans IDE instance for each cluster instance.

▼ To Debug a Business Process Deployed in a Cluster

1. **Launch the GlassFish Admin Console.**
2. **In the navigation bar on the left, expand JBI, expand Components, and then select sun-bpel-engine.**
3. **In the View/Update Instance field, select an instance you want to monitor.**
4. **Set the Debug Enabled property to `true`.**
5. **Set the Debug Port property to a unique valid port number.**
6. **Click Save.**
7. **Repeat the above steps for each instance you want to monitor.**
8. **If you set breakpoints in the business process and attach a port number, you need to launch a new NetBeans IDE instance, set a break point, and then attach the port numbers for each instance in the cluster.**

Configuring the IEP Service Engine for Clustering

You can configure the IEP SE to run in a GlassFish cluster on the same or different servers, You can also configure the SE for high availability and failover on a standalone GlassFish server. This topic describes configuring the IEP SE on a GlassFish cluster. For information on standalone configuration for high availability and failover, see "Configuring Components for Standalone High Availability and Failover" on page 40). The IEP SE can be implemented in a cluster on the same or different servers.

The IEP SE uses a database to maintain instance information in a cluster. In a clustered environment, an event processor belongs to any one of the live cluster instances at a given time. All instances can receive the incoming messages, and once a message is received it is inserted into the IEP database for further processing. The instance that owns the event processor picks up the event to complete processing. The output is only written by the instance that owns the event processor. If an instance fails, any in-process transactions are taken over by one of the remaining instances and the process is completed. When the failed engine recovers, it continues to process new requests.

The IEP SE is not dependent on any shared libraries.

Note – The servers in a cluster need to be on the same time zone in order to detect whether an instance has failed.

Setting Up the IEP Database

Default connection pools and JDBC resources are automatically created for an IEP Derby database, as are the Derby database and persistence tables once you enable the engine on the cluster. The tables are automatically created for an Oracle database, but you need to manually create the connection pools and JDBC resources for Oracle.

For Derby, you can customize the default connection pools and JDBC resources if necessary, or create new ones for the cluster. IEP uses both XA and non-XA transactions so two connection pools and JDBC resources are required, one for each type.

For more information about creating connection pools and JDBC resources, see Chapter 3, "JDBC Resources," in *Sun GlassFish Enterprise Server 2.1 Administration Guide*.

▼ To Set Up the IEP Database

1 To setup an Oracle database, do the following:

a. Copy the Oracle driver (`ojdbc14.jar`) from your Oracle installation to *GlassFish_Home*`/lib`. Restart the GlassFish server.

b. Create an Oracle database instance using the administrative tools provided by Oracle.

c. Customize and run the following script to create the IEP user (or use the information in the script as a guide).

`http://wiki.open-esb.java.net/attach/HowToRunIEPOnOracle/create_iepse_user.sql`

2 Launch the GlassFish Admin Console.

3 Create and configure two connection pools for the database, one for XA transactions and one for non-XA transactions. Point both connection pools to the same database using the same user name and password.

Note – For Derby, you can use the connection pools that are automatically created when IEP is installed.

- **Select Allow Non Component Callers.**
- **For a Derby database, enter the following properties:**
  - `Host` (can be **`localhost`**)
  - `PortNumber` (1527 is the default Derby port)
  - `DatabaseName`
  - `User`
  - `Password`
  - `connectionAttributes` (set this to **`;create=true`** to automatically create the Derby database)
- **For an Oracle database, enter the following properties:**
  - `URL` (the format for the URL is `jdbc:oracle:thin:@`*hostName*`:`*portNumber*`:`*databaseName*)
  - `user`
  - `password`

4 Create and configure two JDBC resources for the database, one for XA transactions and one for non-XA transactions. After you enter the name and connection pool, do the following:

a. Under Available Targets, select the cluster and click Add.

b. Click OK.

This adds the resource to the cluster and enables the resource on the cluster.

5 **To create the database tables, complete the steps under "Adding the IEP Service Engine to the Cluster" on page 24.**

Adding the IEP Service Engine to the Cluster

The IEP SE is not dependent on any shared libraries; you only need to add the service engine to the cluster.

▼ To Add the IEP Service Engine to the Cluster

1 **Add the IEP SE (sun-bpel-engine) to the cluster as described in "To Add a GlassFish ESB Component to a Cluster" on page 12.**

2 **Before you click OK on the Manage Targets page, do the following:**

a. **Under General – Configure Component, specify a value for the Lease Renewal Interval.**

Note – The Lease Renewal Interval is the time period in seconds that BPEL engines wait before renewing their lease to let the cluster know it is still running (also known as the *heartbeat*). If an engine does not update within the specified time period, it is considered to be unavailable.

---

b. **In the Non XA Data Source Name property, enter the JNDI name of the non-XA JDBC resource that you created earlier.**

c. **In the XA Data Source Name property, enter the JNDI name of the XA JDBC resource that you created earlier.**

This is the database that persists state data for business process instances for recovery.

d. **Update the value of the Database Schema Name property if you created the database using a username other than `iepseDB`.**

e. **Configure the remaining runtime properties as you typically would for the IEP SE.**

For more information, see Developer Guide to the Intelligent Event Processor.

3 **Verify that the EMS_ENGINES table in the persistence database contains a row for each instance in the cluster.**

Tip – If the rows do not appear, check the following:

- Both JDBC Resources are enabled on the target cluster.
- The engine is enabled on the target cluster.

If both of the above are enabled, you might need to stop and restart the IEP SE in order for the rows to appear.

Configuring the XSLT Service Engine for Clustering

There are no configuration requirements to run the XSLT SE in a clustered environment. Because it is a stateless SE, the state does not need to be persisted. The XSLT SE is dependent on the following shared library:

- Sun WSDL Extension Library

▼ To Add the XSLT Service Engine to the Cluster

1. **Add the following shared library to the cluster as described in "To Add a Shared Library to a Cluster" on page 11:**
   - sun-wsdl-ext-library
2. **Add the XSLT SE (sun-xslt-engine) to the cluster as described in "To Add a GlassFish ESB Component to a Cluster" on page 12. Before you click OK on the Manage Targets page, you can configure the following runtime properties:**
   - Thread Count
   - Transformation Engine

Configuring the Java EE Service Engine for Clustering

There are no configuration requirements to run the Java EE SE in a clustered environment. The Java EE SE is automatically added to any clusters you create, so you only need to start the service engine to enable it on the cluster.

The Java EE SE is dependent on the following shared library:

- Sun WSDL Shared Library

▼ To Enable the Java EE Service Engine on the Cluster

1 **Add the following shared library to the cluster as described in "To Add a Shared Library to a Cluster" on page 11:**

- sun-wsdl-library

2 **In the left navigation panel of the GlassFish Admin Console, expand JBI, expand Components, and then select sun-javaee-engine.**

3 **Click the Targets tab.**

4 **Select the check box next to the cluster, and then click Start.**

Configuring the Data Mashup Service Engine for Clustering

The Data Mashup SE is a stateless engine that supports read queries. No special configuration is required to run the Data Mashup SE in a cluster because it is stateless. It works in high-availability (HA) and failover systems. In failover systems, the entity that invoked the Data Mashup SE either recovers or resends the data after a failed connection or a failure to invoke the engine. The Data Mashup SE can then service the request.

▼ To Add the Data Mashup Service Engine to the Cluster

The Data Mashup SE is not dependent on any shared libraries, so you only need to add the SE to the cluster.

1 **Add the Data Mashup SE to the cluster as described in "To Add a GlassFish ESB Component to a Cluster" on page 12.**

The name of the Data Mashup SE node is sun-edm-engine.

2 **Before you click OK on the Manage Targets page, configure any runtime properties as you typically would for the Data Mashup SE.**

Configuring the Database Binding Component for Clustering

The Database BC works differently in inbound mode than outbound mode in a cluster. In inbound mode, the database BC distributes records evenly across the instances based on the number of records configured for each poll. For inbound mode, you need to create database tables to manage the instances. In outbound mode, the Database BC works the same in a clustered environment as in a standalone environment. For important information on failover, see the *Release Notes for Sun GlassFish ESB v2.1*.

The Database BC is not dependent on any shared libraries.

Creating the Clustering Database for the Database Binding Component

When using the Database BC in inbound mode in a clustered environment, you need to create a clustering database to manage the load. This is not required for outbound mode.

▼ To Create the Clustering Database for the Database Binding Component

Before You Begin Create a database instance in which to create the clustering database tables, and make sure it is running.

1. **Run the following SQL command against the database for each polling table, updating the variables for each table.**

   ```
   create table owner_polling_table (pkname datatype(size) primary key, instance_name varchar(50), status varchar(30))
   ```

   where:

   - *polling_table* is the name of the polling table
   - *pkname* is the name of the polling table's primary key
   - *datatype* is the data type for the primary key
   - *size* is the length of the primary key column

2. **Run the following command against the database:**

   ```
   create table instancestate (instanceid varchar(50), lastupdatetime timestamp, tablename varchar(50))
   ```

3. **Create and configure a connection pool for the database (for more information, see Chapter 3, "JDBC Resources," in *Sun GlassFish Enterprise Server 2.1 Administration Guide*).**

4 **Create and configure a JDBC resource for the database. When you create the JDBC Resource, select the cluster from the Available Targets list and click Add.**

This adds the resource to the cluster and also enables the resource on the cluster.

Adding the Database Binding Component to the Cluster

The Database BC does not depend on any shared libraries, so you only need to add the BC to the cluster.

▼ To Add the Database Binding Component to the Cluster

1 **Add the Database BC to the cluster as described in "To Add a GlassFish ESB Component to a Cluster" on page 12.**

The name of the Database BC node is sun-database-binding.

2 **Before you click OK on the Manage Targets page, configure the runtime properties:**

- **Enter the number of outbound threads.**

  Note – When using a Derby database, be sure to set this to a relatively high number (for example, 100 per 1000 records) or create all cluster-related tables in the same database. Otherwise, a SQL exception may be thrown and the database will become unusable.

- **In the Cluster Database JNDI Name property, enter the name of the JDBC resource you created earlier (in "To Create the Clustering Database for the Database Binding Component" on page 27.**

Configuring the File Binding Component for Clustering

When the File BC is implemented in a cluster, files are distributed evenly across instances in inbound mode and each file is delivered to only one instance if a file locking mechanism is in place. In on-demand read mode, each file is available to read either from all the instances (configuration mode) or from only one instance (payload mode). In outbound mode, all instances write to either one file (append mode) or to independent files, depending on how the BC is configured.

You do need to configure the File WSDL file to ensure that outbound files are not overwritten by simultaneous threads and that inbound files are picked up only once by only one inbound thread. Name the output files using a pattern that includes either a UUID or sequence number

in the filename to prevent the files from being overwritten. For inbound files, you need to configure the locking mechanism of the File BC to prevent the files from being picked up by more than one instance. For on-demand read files, there is a new property named `deleteFileOnRead` that allows you to specify that a file be removed from the directory when it is read. You should also configure the BC to archive files if you have them removed. These steps are described under "Configuring the File BC WSDL File for Clustering" on page 29.

Adding the File Binding Component to the Cluster

The File BC is dependent on the following shared library:

- Sun Encoder Library

▼ To Add the File Binding Component to the Cluster

1. **Add the following shared library to the cluster as described in "To Add a Shared Library to a Cluster" on page 11:**
   - sun-encoder-library
2. **Add the File BC to the cluster as described in "To Add a GlassFish ESB Component to a Cluster" on page 12.**

   The name of the File BC node is sun-file-binding.
3. **Before you click OK on the Manage Targets page, enter the number of threads for the outbound processor.**

Note – If you need to modify the number of outbound threads once the File BC is deployed to a cluster, you need to make the changes using the GlassFish Admin Console and you need to update the property for each instance in the cluster.

Configuring the File BC WSDL File for Clustering

The File BC has no runtime configuration requirements for clustering, but there are a few requirements and options for configuring the properties in any File BC WSDL documents deployed to the cluster. These options assist with concurrency issues, load balancing, and failover.

In a clustered environment, you need to ensure that inbound files are picked up only once by one instance and that outbound files are not overwritten by simultaneous threads. For outbound files, you can specify a pattern for the output message name using either `%u` or `%seq_name` to keep all file names unique (note that `%d` is not sufficient here). To ensure that

inbound messages are not processed by more than one instance, use the File BC's file locking mechanism. This mechanism uses a thread lock (T_LOCK) and a file lock (F_LOCK) to protect the messages from concurrent polling. The File BC processes messages as follows using the locking mechanism:

1. Acquire the file lock.
2. If the file lock is acquired, continue processing; otherwise release the thread lock.
3. List the files under the directory specified by `file:message/@fileDirectory` that match the file name or pattern.
4. Move each of the files to a working directory with the specified pattern appended to the file name.
5. Place the file names in the working directory into a queue.
6. Release the file lock.

For more information about the properties discussed below, see the following topics in *Using the File Binding Component*:

- Configuring File BC WSDL Attributes
- Inbound Message Processing
- Persisted Sequencing

▼ To Configure the File BC WSDL File for Clustering

1 For outbound messages, do the following:

a. In the File BC WSDL document, scroll to the input message definition under Bindings (named file:message, by default).

b. Select file:message.

The Properties panel displays the properties for the message.

c. In the FileName property, enter a name for the output file using either `%u` to assign a unique UUID number to each output file or `%{seq_name}` for persisted sequence number creation. For example, `output%u.txt`**.**

Note – Do not use **`%d`** in the file name. It does not ensure a unique file name and files might be overwritten.

---

d. In the `fileNameIsPattern` **property, enter `true`.**

e. Save and close the WSDL file.

2. **For on-demand, read-only messages, you can configure any of the following options:**

   a. **In the File BC WSDL document, scroll to the input message under Bindings (named file:message, by default).**

   b. **Select file:message.**

      The Properties panel displays the properties for the message.

   c. **Do one of the following:**

      - **To allow multiple instances and processes to access the input file, set the value of the `deleteFileOnRead` property to `false`.**
      - **To ensure only one thread and one process accesses the file, set the value of the `deleteFileOnRead` property to `true`.**

        This deletes inbound on-demand messages once they are read.

   d. **If `deleteFileOnRead` is set to `true`, do one of the following:**

      - **To retain a copy of the deleted files, set the value of the `archive` property to `true` and specify an archive directory.**

        The files are removed from the input directory and are copied to the archive directory with a UUID appended to the name to ensure it is unique. This retains a history of the files that have been processed.
      - **To simply delete the files once they are read, set the value of the `archive` property to `false`.**

3. **For inbound messages, use the file locking mechanism of the File BC to ensure that each inbound file is picked up only once by one instance in the cluster.**

   Use the following attributes to implement the locking mechanism:

   - file:address/@lockName - The file name used for the F_LOCK.
   - file:address/@workArea - The name of the working directory.
   - file:address/@seqName - The name of the directory where all sequence numbers are saved.
   - file:address/@persistenceBaseLoc - The name of the directory where the lock files are stored.

   For more information, see Configuring File BC WSDL Attributes (under File Address Element) and Inbound Message Processing.

Configuring the FTP Binding Component for Clustering

When the FTP BC is implemented in a cluster, all service provider instances poll requests simultaneously, business logic is applied to those requests simultaneously, and responses are posted simultaneously. Similarly, all service consumer instances post requests simultaneously. The FTP BC is highly available, and a failure of any instance does not stop message transfer. Messages are processed evenly among the pollers to balance the workload.

In on-demand read mode, each file is available to read from either all the instances (configuration mode) or only one instance (payload mode). In outbound mode, configure all instances to write to independent files rather than appending to one file. Using a UUID or persisted sequence number pattern in outbound file names ensures that files are not overwritten.

In inbound mode, the FTP BC distributes files evenly across the instances in a cluster, and each file is delivered to only one instance. You do need to configure a file locking mechanism to ensure files are not overwritten and to prevent redundant processing. This is configured by defining a persistence directory in the FTP BC's JVM options (described in the instructions below).

The FTP BC processes messages as follows using the locking mechanism:

1. Acquire the file lock.
2. If the file lock is acquired, continue processing; otherwise release the thread lock.
3. Poll the target.
4.
   - If the poller is retrieving a request, the target is `ftp:message/@messageRepository/inbox/messageName` or `ftp:transfer/@receiveFrom`.
   - If the poller is retrieving a response, the target is `ftp:message/@messageRepository/outbox/messageName` or `ftp:transfer/@sendTo`.
5. If the message is retrieved successfully, archive the file so it will not be polled again.
6. Release the file lock.

The FTP BC is dependent on the following shared library:

- Sun Encoder Library

Adding the FTP Binding Component to the Cluster

The FTP BC is dependent on the following shared library:

- Sun Encoder Library

▼ To Add the FTP Binding Component to the Cluster

1. **Add the following shared library to the cluster as described in "To Add a Shared Library to a Cluster" on page 11:**

   - sun-encoder-library

2. **Add the FTP BC to the cluster as described in "To Add a GlassFish ESB Component to a Cluster" on page 12.**

   The name of the FTP BC node is sun-ftp-binding.

3. **Before you click OK on the Manage Targets page, configure the runtime properties as you typically would for the FTP BC.**

4. **Configure the JVM properties for the FTP BC by doing the following:**

   a. **In the left navigation panel of the Admin Console, expand Configurations, expand *cluster_name*-config, and then select JVM Settings.**

   b. **Click the JVM Options tab.**

   c. **To enable clustering, click Add JVM Option and enter the following in the empty row that appears:**

      ```
      -Dcom.sun.jbi.ftpbc.isClustered=true
      ```

   d. **To define a file-based persistence directory, click Add JVM Option and enter the following in the empty row that appears:**

      ```
      -Dcom.sun.jbi.ftpbc.token.persistence.url=persistence_directory
      ```

      where *persistence_directory* is the location where the FTP BC lock files are stored.

   e. **Click Save.**

   f. **Restart the cluster to pick up the changes to the JVM options.**

Configuring the FTP BC WSDL for Clustering

The FTP BC has no runtime configuration requirements for clustering, but there are a few requirements and options for configuring the properties in any FTP BC WSDL documents deployed to the cluster. These options assist with concurrency issues and load balancing

To ensure that output files are not overwritten by concurrent threads, you can specify a pattern for the output message name using either `%u` or `%`*seq_name* to keep all file names unique.

▼ To Configure the FTP BC WSDL for Clustering

1 For outbound messages, do the following:

a. In the FTP BC WSDL document, scroll to the input message definition under Bindings (named ftp:message or ftp:transfer, by default).

b. Select ftp:message or ftp:transfer.

The Properties panel displays the properties for the message.

c. Do one of the following:

- **For ftp:message, modify the `messageName` property to use a unique pattern.**

  Use either `%u` to assign a unique UUID number to each output file, or `%`*seq_name* for persisted sequence number creation.

  For example, `output%u.txt`.

- **For ftp:transfer, modify the `sendTo` property to use a unique pattern.**

  Use the same patterns as described above.

Note – Do not use **`%d`** in the file name because files might be overwritten.

---

d. Save and close the WSDL file.

2 To define a directory where recovery logs, malformed messages, and so on are stored, set the `ftp:address/@baseLocation` WSDL attribute to the full path.

Configuring the HTTP Binding Component for Clustering

The HTTP BC has no load balancing and failover mechanisms of its own, but Sun provides the HTTP Load Balancer to perform these functions. The Load Balancer is a web server plug-in that accepts HTTP and HTTPS requests and distributes them to application server instances in a cluster. This allows the HTTP BC to be scaled horizontally, running on multiple instances in a cluster. The Load Balancer gives you several advantages by managing the workload across cluster instances. If you use the HTTP Load Balancer, you need to configure the load balancer for each instance in the cluster.

You can read more information about clustering for the HTTP BC and using the HTTP Load Balancer at Clustering Support for HTTP Binding Component. For detailed information about the HTTP Load Balancer, see Chapter 5, "Configuring HTTP Load Balancing," in *Sun GlassFish Enterprise Server 2.1 High Availability Administration Guide*.

Enabling the HTTP Binding Component on the Cluster

The HTTP BC is not dependent on any shared libraries, and is already added to the cluster. You only need to start the binding component to enable it.

▼ To Enable the HTTP Binding Component on the Cluster

1. **In the left navigation panel of the GlassFish Admin Console, expand JBI, expand Components, and then select sun-http-binding.**
2. **Click the Targets tab.**
3. **Select the check box next to the cluster, and then click Start.**
4. **To modify any HTTP BC properties on the cluster, perform any of the steps under "Modifying Server Properties for GlassFish ESB Components in a Cluster" on page 14.**

   For more information, see HTTP Binding Component Runtime Properties.

Configuring the HTTP BC Port Numbers for Clustering

Each component instance in the cluster must have exclusive access to the resource, so a unique port number must be assigned to each instance. When you define service ports for the URL, use variables instead of actual port numbers in the `soap:address` element. This allows the client to direct HTTP requests to the default port, which is defined in the HTTP BC runtime properties. The value of the variable is resolved by the HTTP BC based on the configured default values used when the application was deployed.

Note – If you reinstall the HTTP BC or drop it from the cluster and add it back in, you need to reconfigure the default ports for each component instance. The BC also needs to be configured for each server instance in the cluster.

▼ To Configure the HTTP BC Port Numbers for Clustering

1 **In the WSDL files for the HTTP BC, change any hard-coded HTTP and HTTPS port numbers in the `soap:address` elements to variables.**

Use the following variables:

- `${HttpDefaultPort}` for the HTTP port
- `${HttpsDefaultPosrt}` for the HTTPS port

For example, instead of using this URL:

```
<soap:address location="http://localhost:18181/Synchronous"/>
```

Use the following URL:

```
<soap:address location="http://localhost:${HttpDefaultPort}/Synchronous"/>
```

2 **If it is not already enabled, enable the HTTP BC on the target cluster. Do the following to enable the BC:**

a. **On the GlassFish Admin Console, expand JBI, expand Components, and then select sun-http-binding.**

b. **Click the Targets tab.**

c. **In the Targets list, select the check box next to the cluster and then click Start.**

3 **To configure the BC, click the Configuration tab and then do the following:**

a. **In the View/Update Instance field, select the first instance to configure.**

b. **Verify the default values for the HTTP Port Number and the Default HTTPS Port Number properties.**

Note – These values are chosen automatically and must be different for each instance. If there is a port conflict, you can change the default port numbers but it is recommended that you use the default values.

c. **Click Save.**

d. Repeat the above steps for each instance in the cluster, verifying that the port numbers are unique for each instance.

Configuring the JMS Binding Component for Clustering

The JMS BC can be configured to use queues or durable topics on either clustered JMS brokers or an independent JMS server. In a GlassFish cluster, JMS queues and durable topics are supported on independent JMS servers, but non-durable topics are not supported. With non-durable topics, the BC on each instance is treated as an independent subscriber, which results in duplicate messages in GlassFish ESB. However, non-durable topics do work on a JMS cluster with a standalone GlassFish instance.

More information about JMS BC clustering is provided at "JMS Binding Component Clustering" in *Using the JMS Binding Component*. For more information about JMS clustering, see Chapter 8, "Broker Clusters," in *Sun Java System Message Queue 4.1 Administration Guide* and Chapter 10, "Java Message Service Load Balancing and Failover," in *Sun GlassFish Enterprise Server 2.1 High Availability Administration Guide*.

The JMS BC is dependent on the following shared library:

- Sun Encoder Library

▼ To Add the JMS Binding Component to the Cluster

1. **Add the following shared library to the cluster as described in "To Add a Shared Library to a Cluster" on page 11:**
   - sun-encoder-library
2. **Add the JMS BC to the cluster as described in "To Add a GlassFish ESB Component to a Cluster" on page 12.**

   The name of the JMS BC node is sun-jms-binding.
3. **Before you click OK on the Manage Targets page, modify the number of threads if necessary.**

Configuring the LDAP Binding Component for Clustering

The LDAP BC is stateless, which means it scales in a clustered environment because the instances do not need to coordinate state and each maintains its own connections. In a failover system, the original caller or last persistence point re-delivers the message if configured for at least once delivery (the default) or once and only once delivery (XA transactions). When the message is re-delivered to the LDAP BC, it performs the same operation again.

In case of failure, messages are processed as follows:

- For operations in which multiple processes do not affect the result (such as read or search operations), the request processing continues as before the failure.
- For operations that modify, delete, and add, it is not guaranteed that each of these operations is only executed once per message because LDAP does not support transactional behavior on its own. It behaves in the same manner in a non-clustered environment.

The LDAP BC is not dependent on any shared libraries.

▼ To Add the LDAP Binding Component to the Cluster

1 **Add the LDAP BC to the cluster as described in "To Add a GlassFish ESB Component to a Cluster" on page 12.**

The name of the LDAP BC node is sun-ldap-binding.

2 **Before you click OK on the Manage Targets page, modify the values of the runtime properties as you typically would for the LDAP BC.**

Configuring the Scheduler Binding Component for Clustering

The Scheduler BC is not aware of multiple instances in a cluster. When implemented in a clustered environment, the BC works in the same way as in a standard environment, which means that all instances of the BC in a cluster function independently of one another instead of sharing the work load. Each time the Scheduler BC is triggered in a cluster, each server instance in the cluster processes the request, which results in multiple responses to each request instead of just one.

To use the Scheduler BC in a clustered environment, you need to configure a way for the receiving provider to treat the received trigger message such that multiple instances running at the same time do not change the result. For example, when using the Scheduler BC in conjunction with the Database BC to insert rows into a table, using a field from the message as a primary key would prevent multiple rows being inserted for one request. The Scheduler BC exposes an intended trigger date (accessed through the normalized message property

`org.glassfish.openesb.scheduler.inbound.date`) that could be used as a primary key. Triggers are described under "Creating Simple Triggers" in *Using the Scheduler Binding Component*. The date format is described under "Using the Scheduler Control and Triggers Wizard" in *Using the Scheduler Binding Component*. You can use the `substring-before` XPath function in a business process to extract the timestamp, along with logic to catch any unique key violations. The `substring-before` function is documented under BPEL Mapper String Functions (`http://www.netbeans.org/kb/60/soa/bpel-mapper.html#StringFunc`).

The Scheduler BC is dependent on the following shared libraries:

- Sun WSDL Extension Library

▼ To Add the Scheduler Binding Component to the Cluster

1. **Add the following shared library to the cluster as described in "To Add a Shared Library to a Cluster" on page 11:**
   - sun-wsdl–ext–library
2. **Add the Scheduler BC to the cluster as described in "To Add a GlassFish ESB Component to a Cluster" on page 12.**

   The name of the Scheduler BC node is sun-scheduler-binding.
3. **Before you click OK on the Manage Target page, modify the values of the runtime properties as you typically would for the Scheduler BC.**

Deploying a Service Assembly to a Cluster

In order for an application to run on a cluster, you need to deploy the Service Assembly (SA) to the cluster. Because NetBeans does not support GlassFish clustering, any Service Assemblies to be run on a cluster need to be deployed using the GlassFish Admin Console or command line. The following instructions are for deploying an SA using the GlassFish Admin Console.

▼ To Deploy a Service Assembly to a Cluster

Before You Begin Build the SA project to generate the package file that will be deployed to the cluster. Make sure that all of the GlassFish ESB components included in the Service Assembly have been added to the target cluster and are in the started state. If any components have not been added to the cluster, the deployment will fail. You should also know the location of the Service Assembly package file.

1. **In the left navigation bar on the GlassFish Admin Console, expand JBI and select Service Assemblies.**
2. **Click Deploy.**
3. **Select the SA package file in either the File to Upload or File to Copy field, depending on the location of the file.**
4. **Click Next.**
5. **By the Status field, select Enabled.**
6. **In the Add to Available Targets List, select the cluster.**
7. **Click Finish.**

Note – You can also deploy the SA using the following commands, run from the *GlassFish_home*`\bin` directory:

`asadmin deploy-jbi-service-assembly --target` *cluster_name SA_file*

`asadmin start --target` *cluster_name SA_file*

where *cluster_name* is the name of the GlassFish cluster and *SA_file* is the path and file name for the SA package file.

Caution – When an SA is deployed to a cluster, make sure to only deploy and undeploy the SE from the Admin Console or the command line. You cannot perform these tasks from NetBeans. Disable the SA before you undeploy it.

Configuring Components for Standalone High Availability and Failover

All GlassFish ESB components provide high availability and failover features without using GlassFish clustering. Implementing the BPEL SE or IEP SE in this type of environment requires special configuration. The service engines are installed across multiple standalone GlassFish domains installed on multiple machines. In this implementation, if one engine fails, any running BPEL or IEP processes failover to the running engines.

Note – You can install the GlassFish domains on the same or different machines, but installing on different machines provides high availability and failover.

Both the BPEL SE and IEP SE use persistence databases to manage the state of each instance. They depend on the database server for high availability and failover features, so the persistence database needs to be configured for high availability and failover.

Because of the new support for GlassFish clustering for GlassFish ESB components, this feature is provided primarily for compatibility with previous versions.

Configuring the BPEL Service Engine for Multiple Standalone Instances

You can deploy the same application across multiple BPEL SEs that connect to the same database. For both clustering and failover, BPEL SE persistence must be enabled.

▼ To Configure the BPEL Service Engine for Multiple Standalone Instances

Before You Begin If you are using a Derby database, start the database independent of GlassFish. If you are using Oracle, create the database instance for the persistence database.

1. **Install two or more GlassFish v2.1 domains.**
2. **Configure the first GlassFish domain by doing the following:**
   a. **Launch the Admin Console.**
   b. **Create the database connection pools and JDBC resources as described in "Setting Up the BPEL Database" on page 18.**
   c. **In the left navigation bar, select Application Server.**
   d. **Click the JVM Settings tab, and then click the JVM Options tab.**
   e. **Click Add JVM Option, and enter the following in the blank row that appears:**
      `-Dcom.sun.jbi.bpelse.isClustered=true`
   f. **Click Add JVM Option, and enter the following in the blank row that appears:**
      `-Dcom.sun.jbi.bpelse.engineId=`*Name*
      Where *Name* is a unique name for the BPEL SE instance.

g. **Restart the application server.**

h. **Make sure all necessary components are installed.**

i. **Repeat these steps on the subsequent GlassFish domains. Use a unique name for the instance, and point all domains to the same persistence database.**

3 **When all domains are configured, start the BPEL SE (sun-bpel-engine) on all domains.**

4 **Configure the following runtime properties for each IEP SE instance, and then stop and restart the BPEL SE:**

- `Persistence Enabled` – Set this to **`true`**.
- `Non XA Data Source Name` – The non-XA JDBC resource you created earlier.
- `XA Data Source Name` – The XA JDBC resource you created earlier.

5 **Check the BPEL SE database table named ENGINE. It should list each engine ID you defined.**

6 **When you deploy the Service Assembly, deploy it to all standalone GlassFish domains.**

Configuring the IEP Service Engine for Multiple Standalone Instances

You can deploy the same application across multiple IEP SEs that connect to the same database. In a standalone environment, an event processor is owned by the IEP instance on which it is deployed first. The rest of the processing is similar to that in a clustered environment. All instances can receive the incoming messages, and once a message is received it is inserted into the IEP database. The instance that owns the event processor picks up the event to complete processing and writes the output. If an IEP instance fails, any in-process transactions are taken over by one of the remaining IEP instances and the process is completed. When the failed engine recovers, it continues to process new requests.

▼ To Configure the IEP Service Engine for Multiple Standalone Instances

Before You Begin If you are using a Derby database, start the database independent of GlassFish. If you are using Oracle, create the database instance for the persistence database.

1 **Install two or more GlassFish v2.1 domains.**

2 **Configure the first GlassFish domain by doing the following:**

a. **Launch the Admin Console.**

b. **Create the database connection pools and JDBC resources as described in "Setting Up the IEP Database" on page 22.**

c. **In the left navigation bar, select Application Server.**

d. **Click the JVM Settings tab, and then click the JVM Options tab.**

e. **Click Add JVM Option, and enter the following in the blank row that appears:**

```
-Dcom.sun.jbi.iepse.isClustered=true
```

f. **Click Add JVM Option, and enter the following in the blank row that appears:**

```
-Dcom.sun.jbi.iepse.instanceName=Name
```

Where *Name* is a unique name for the IEP SE instance.

g. **Restart the application server.**

h. **Make sure all necessary components are installed.**

i. **Repeat these steps on the subsequent GlassFish domains. Use a unique name for the instance, and point all domains to the same persistence database.**

3 **When all domains are configured, start the IEP SE (sun-iep-engine) on all domains.**

4 **Configure the following runtime properties for each IEP SE instance, and then stop and restart the IEP SE:**

- `Non XA Data Source Name` – The non-XA JDBC resource you created earlier.
- `XA Data Source Name` – The XA JDBC resource you created earlier.

  lmu
- `Database Schema Name` – This matches the username specified in the connection pools.

5 **Check the IEP SE database table named EMS_ENGINE. The ID column should list each instance name you defined.**

6 **When you deploy the Service Assembly, deploy it to all standalone GlassFish domains.**