Using the BPEL Service Engine in a Project

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

Part No: 820–0605
July 2008

080906@20795

Contents

Using the BPEL Service Engine in a Project

This guide provides an overview of the BPEL Service Engine, and includes details that are necessary to configure and deploy the service engine in a JBI project. The BPEL Service Engine is a JSR 208-compliant JBI runtime component that provides services for executing WS-BPEL 2.0 (or simply BPEL) compliant business processes. WS-BPEL 2.0 (Web Services Business Process Execution Language) is an XML-based language used to program business processes.

For information about creating and editing BPEL processes using the BPEL Designer, see Developer Guide to the BPEL Designer.

For more information, see the Java CAPS web site at `http://goldstar.stc.com/support`.

What You Need to Know

The following topics contain introductory and conceptual information for the BPEL Service Engine.

What You Need to Do

The following topics contain instructions for configuring and using the BPEL Service Engine in a Project

More Information

The following topics contain additional information about using the BPEL Service Engine

- "BPEL 2.0 Language Constructs" on page 17

About the BPEL Service Engine

The BPEL Service Engine provides runtime services for deploying BPEL processes. The BPEL Service Engine is used to execute WS-BPEL 2.0 (or simply BPEL) compliant business processes. WS-BPEL 2.0 (Web Services Business Process Execution Language) is an XML-based language used to program business processes.

Business processes typically involve the exchange, or *orchestration*, of messages between the process and other web services known as *partner services*. The contract between a business process and partner services is described in WSDL 1.1. The message exchange between a business process and partner services is wrapped in the WSDL 1.1 message wrapper, as defined by the JBI specification, and routed via the JBI Normalized Message Router (NMR). The NMR interacts with external web services, not resident on the local JVM, via binding components. Binding components are responsible for encapsulating protocol-specific details. Transactions between the BPEL Service Engine and collocated EJBs or web components are handled through the Java EE service engine.

WS-BPEL 2.0 utilizes several XML specifications: WSDL 1.1, XML Schema 1.0, XPath 1.0, and XSLT 1.0. Note that the JBI specification is targeted toward WSDL 2.0 and accommodates WSDL 1.1 by defining the wrapper.

The BPEL Service Engine supports one-way, request-response operations (as defined in WSDL 1.1), within stateful, long-running interactions that involve two or more parties. Asynchronous request-response is accomplished using two one-way operations, one implemented by a partner, the other implemented by the business process using correlation.

The BPEL editor provides an easy-to-use interface that allows you to build or edit your BPEL Project, deploy your project to the BPEL Service Engine, and test BPEL services. The BPEL Service Engine implements most constructs; non-implemented constructs are rarely used.

The BPEL Service Engine is bundled as part of NetBeans IDE 6.1 with SOA and Project Open ESB. You can download these components from the Sun Developer Network at: `http://java.sun.com/downloads/`.

BPEL Service Engine Features

Following features are supported by the BPEL Service Engine:

- Standard JBI 1.0 engine component
- Supports BPEL 2.0, see "BPEL 2.0 Language Constructs" on page 17 for details
- Provides and consumes web services defined by using WSDL 1.1

- Exchanges messages in JBI-defined XML document format for wrapped WSDL 1.1 message parts
- Implements endpoint status monitoring
- Supports multiple-thread execution
- Supports debugging of business processes
- Supports database persistence of business process instances for reliable recovery from system failure
- Supports load balancing and failover when clustered

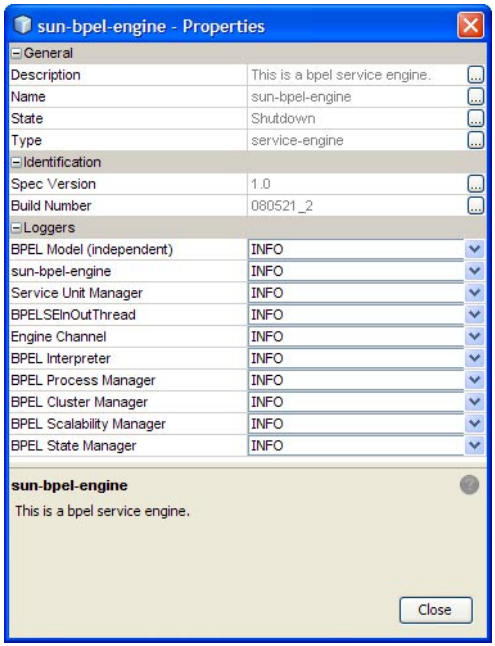
Configuring the BPEL Service Engine Runtime Properties

The BPEL Service Engine runtime properties can be configured from the NetBeans IDE, or from a command prompt (command line interface) during installation.

Accessing the BPEL Service Engine Runtime Properrties

To display or edit the properties in the NetBeans IDE, do the following:

1. From the Services tab of the NetBeans IDE, expand the Servers node.
2. Start your application server, for example GlassFish v2. To do this, right-click your application server and select `Start` from the shortcut menu.
3. Under the application server, expand the JBI → Service Engines nodes and select the BPEL Service Engine. The current BPEL Service Engine properties are displayed at the right side of the NetBeans IDE. You can also double-click the BPEL Service Engine to open a properties window.
4. Edit the properties as needed. To apply any changes you make to the runtime BPEL Service Engine properties, stop and restart the BPEL Service Engine.



Runtime Property Descriptions

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

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

Property Name	Description	Default Value
Type	Type of the JBI Component (service-engine or binding-component)	service-engine
Identification Properties		
Build Number	Date and time stamp for the current build.	<build_number>
Spec Version	BPEL specification fully supported by this build.	<spec_version>
Loggers		
BPEL Model (independent)	Specifies the user-designated level of logging for each event. Each logger can be set to record information at any of the following levels: ■ FINEST: messages provide highly detailed tracing ■ FINER: messages provide more detailed tracing ■ FINE: messages provide basic tracing ■ CONFIG: provides static configuration messages ■ INFO: provides informative messages ■ WARNING: messages indicate a warning ■ SEVERE: messages indicate a severe failure ■ OFF: no logging messages	INFO
sun-bpel-engine		INFO
Service Unit Manager		INFO
BPELSEInOutThread		INFO
EngineChannel		INFO
BPELInterpreter		INFO
BPEL Process Manager		INFO
BPEL Cluster Manager		INFO
BPEL Scalability Manager		INFO
BPEL State Manager		INFO

BPEL Service Engine Deployment Artifacts

The BPEL Service Engine requires the following artifacts to execute a business process:

- BPEL Documents: define the activity sequence to execute.
- WSDL Documents: define the contract between the business process and partner services.
- Optional XSDs: define exchanged XML documents; XSDs can be defined inline within WSDL documents.

These artifacts are packaged into a service unit, and in turn packaged into the service assembly, along with other JBI component's service units, based on the internal/external partner services requirements.

Service assemblies and service units can be deployed to JBI runtime and corresponding components using Ant scripts provided with the NetBeans IDE. The JBI `DeploymentServiceMBean` interprets the deployment descriptor, jbi.xml, and deploys the service unit to the associated component.

Configuring Persistence for the BPEL Service Engine

In order to ensure the integrity of your business process data in the case of a system failure, you can configure the BPEL Service Engine to persist business process data to a database. With BPEL Persistence enabled, the BPEL Service Engine recovers otherwise lost business process data and continues processing from the point of system failure.

The BPEL Service Engine uses the JDBC resource created in the Sun Java System Application Server (GlassFish) to make the necessary database connection for persistence. NetBeans is bundled with the JavaDB database, Sun's supported distribution of the open source Apache Derby database. The BPEL Service Engine is configured to connect to the JavaDB database by default.

The BPEL Service Engine also supports connecting to an Oracle database to persist data. To create and connect to an Oracle database, see Creating a JDBC Connection Pool for Oracle and Creating a JDBC Resource for Oracle.

This section includes the following topics:

Note – For more information on BPEL Persistence, see Using BPEL Persistence (`http://wiki.open-esb.java.net/Wiki.jsp?page=UsingBPELPersistence`)

Creating a JDBC Connection Pool for the JavaDB Database

The following procedure provides the steps for creating a JavaDB JDBC connection pool.

▼ To create a JDBC Connection Pool

1. **In your web browser, log into the Sun Java System Application Server Administrator Console. For example:**

   `http://localhost:4848, username:admin, password: adminadmin`

2. **In the navigation tree, expand the following nodes:** `Resources` → `JDBC`**.**

3. **Select** `Connection Pools`**, and in the right panel, click the** `New` **button.**

4. **Under General Settings, specify a name (such as bpelseDB).**

5. **Set the Resource Type to** `javax.sql.XADatasource`**.**

6. **Set the database vendor to** `Derby` **and click** `Next`**.**

7. **Under Connection Validation, enable Allow Non Component Callers by selecting the** `Enabled` **check box.**

8. **Under Additional Properties, specify the ServerName, User, Password and DatabaseName.**

   You can use the following values, or choose your own for user, password and database name (keep create=true flag).

   - **ServerName:** machine-name
   - **DatabaseName:** DatabaseName: bpelseDB;create=true
   - **User:** bpelse_user
   - **Password:** bpelse_user

9. **Click** `Finish`**, click the connection pool name and click the** `Ping` **button.**

   This verifies your database connection.

10. **Click Finish to prepare for creating a new JDBC resource.**

Creating a JDBC Connection Pool for Oracle

The following procedure provides the steps for creating an Oracle JDBC connection pool. Before you create your JDBC connection for Oracle, you must create an Oracle user (with required permissions) and tablespace for BPEL Service Engine persistence.

▼ To create an Oracle user

1 **Log into Oracle as** `sysdba` **(from SQLPlus, connect using** `connect sys/manager@machine-name as sysdba`**).**

2 **Execute the following script with default values:**

Click here (`http://www.glassfishwiki.org/jbiwiki/attach/BPELSEConfiguration/bpelse-oracle-create-user-1.sql`) to download the annotated scripts.

```
CREATE TABLESPACE bpelsedb

DATAFILE 'bpelsedb.dat' SIZE 512M REUSE

AUTOEXTEND ON NEXT 2048M MAXSIZE UNLIMITED;

CREATE USER bpelse_user

IDENTIFIED BY bpelse_user

DEFAULT TABLESPACE bpelsedb

QUOTA UNLIMITED ON bpelsedb

TEMPORARY TABLESPACE temp

QUOTA 0M ON system;

GRANT CREATE session to bpelse_user;

GRANT CREATE table to bpelse_user;

GRANT CREATE procedure to bpelse_user;

GRANT select on sys.dba_pending_transactions to bpelse_user;

GRANT select on sys.pending_trans$ to bpelse_user;

GRANT select on sys.dba_2pc_pending to bpelse_user;

GRANT execute on sys.dbms_system to bpelse_user;

GRANT select on SYS.dba_2pc_neighbors to bpelse_user;

GRANT force any transaction to bpelse_user;
```

Note – You can also connect using NetBeans IDE or a number of SQL clients by entering the username `sys as sysdba`. The password should be the same as that of the system user. You can also change the user, tablespace, datafile name, and size/quota according to your requirements.

▼ To create a JDBC Connection Pool

1. **In your web browser, log into the Sun Java System Application Server Administrator Console. For example:**

   ```
   http://localhost:4848, username:admin, password: adminadmin
   ```

2. **Add Oracle JDBC driver classes to the application server classpath.**

   See the Sun Java System Application Server documentation for details on how to add the classes to the application server's classpath. One method is to navigate to `Application Server` > `JVM Settings` > `Path Settings` and specify the path to the jar file (including the jar file name) in the Classpath Suffix box. You must restart the application server before you continue.

3. **In the navigation tree, expand the** `Resources` → `JDBC` **nodes, and select** `Connection Pools`**.**

4. **In the right panel, click the** `New` **button.**

5. **Under General Settings, specify a name (such as** `bpelseDB`**).**

6. **Set the Resource Type to** `javax.sql.XADatasource.`

7. **Set the database vendor to** `Oracle` **and click** `Next`**.**

8. **Under Connection Validation, enable Allow Non Component Callers by selecting the** `Enabled` **check box.**

9. **Under Additional Properties, specify the URL, User, and Password, as follows.**

   You can use the following values, or choose your own for user, password and URL.

   - **URL:** jdbc:oracle:thin:@machine-name:port:sid
   - **User:** bpelse_user
   - **Password:** bpelse_use

10. **Click** `Finish`**, click the connection pool name and click the** `Ping` **button.**

    This verifies your database connection.

11. **Click** `Finish` **to prepare for creating a new JDBC resource.**

Creating a new JDBC Resource

The following procedure provides the steps for creating an application server database JDBC resource.

▼ To create a JDBC resource

1. **In the Sun Java System Application Server (GlassFish) navigation tree, expand the** `Resources` → `JDBC` **nodes, and select** `JDBC Resources`**.**
2. **In the right panel, click the** `New` **button.**
3. **Provide a JNDIName (such as** `jdbc/bpelseDB`**) and specify the JDBC Connection Pool (bplseDB) you created previously.**
   You use this JNDIName when you later enable persistence in the BPEL Service Engine properties.
4. **Expand the** `Configuration` **node and select** `Transaction Service`**.**
5. **For the On Restart parameter, enable Automatic Recovery by selecting the** `Enabled` **check box.**

Enabling Persistence for the BPEL Service Engine

The following procedure provides the steps for configuring the BPEL Service Engine for persistence.

▼ To enable persistence for the BPEL Service Engine

1. **In the NetBeans IDE Services window, expand the** `Sun Java System Application Server (GlassFish)` → `JBI` → `Service Engines` **Nodes.**
2. **Right-click** `sun-bpel-engine` **and select Properties.**
   The `sun-bpel-engine` Properties window appears.
3. **Set the PersistenceEnabled property value to** `true`**.**
4. **Set the JNDIName property value to specify the JNDIName of the JDBC resource that you created when you configured the database.**
5. **Click** `Close` **to save your settings.**
   See "Configuring the BPEL Service Engine Runtime Properties" on page 7 for property descriptions.

6 **Stop, shut down, and start the BPEL Service Engine to enable your new settings.**

Notes on Configuring Persistence

The following notes provide Drop and Truncate Scripts as well as additional information about configuring persistence. Some of the instructions mentioned here may change, so check back for updates or contact the BPEL Service Engine team if you have questions.

- When the BPEL Service Engine is started, it queries the database for the existence of the tables required for persistence. If they are not there then BPEL Service Engine will create the required tables.
- Drop and Truncating tables scripts for derby and oracle. Click here (`http://www.glassfishwiki.org/jbiwiki/attach/BPELSEConfiguration/bpelse-drop-truncate-scripts.zip`) to download the scripts.
- To drop an Oracle user, use the following script (You can download this script from here (`http://www.glassfishwiki.org/jbiwiki/attach/BPELSEConfiguration/bpelse-oracle-drop-user.sql`):

  ```
  DROP TABLESPACE BPELSE_USER_DB INCLUDING CONTENTS AND DATAFILES CASCADE
  CONSTRAINTS;
  ```

Configuring Clustering and Failover for the BPEL Service Engine

In order to optimize and ensure business process throughput on highly scalable systems, the BPEL Service Engine supports clustering and failover . Clustering distributes processing over multiple BPEL Service 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.

Clustering

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

For details about setting up a cluster of application servers with BPEL Service Engines, see the documentation for Sun Java System (GlassFish) 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 a business process instance encounters an engine failure, any suspended instances are picked up by the next available BPEL Service Engine in the cluster.

To configure failover, set the BPEL Service Engine property, `EngineExpiryInterval` to register 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, as a sort of domino effect.
- All BPEL Service Engines in a cluster must reside in the same time zone.

For more information on Clustering and Failover support for the BPEL Service Engine, see Practical Guide for Testing Clustering Support for the BPEL Service Engine (`http://wiki.open-esb.java.net/Wiki.jsp?page=ClusteringSupportExample`)

BPEL BluePrints

BPEL BluePrints are developed to teach and promote good practices in developing business processes. Together they present solutions for developing business processes that logically combine, orchestrate, and consume web services.

The BPEL BluePrints are located at:

https://blueprints.dev.java.net/bpcatalog/ee5/soa/index.html (`https://blueprints.dev.java.net/bpcatalog/ee5/soa/index.html`)

The following BPEL BluePrints are available for download:

- BPEL BluePrint 1: Synchronous Web Service Interactions Using BPEL
- BPEL BluePrint 2: Asynchronous Web Service Interactions Using BPEL
- BPEL BluePrint 3: Fault Handling Using BPEL
- BPEL BluePrint 4: Message-Based Coordination of Events Using BPEL
- BPEL BluePrint 5: Concurrent Asynchronous Coordination of Events Using BPEL

BPEL 2.0 Language Constructs

The following table provides a list of WS-BPEL 2.0 language constructs and whether they are implemented by the BPEL Service Engine.

Features	Description
<variables>	For messages or intermediate data. Supports: ■ Variables of WSDL message and XML schema ■ messageType ■ type ■ element Does not support: ■ from-spec
<invoke>	Invokes operations on web services provided by partners Supports: ■ asynchronous one-way invoke ■ synchronous request-reply invoke ■ fault handling ■ correlation ■ virtual assign: fromPart and toPart Does not support: ■ compensation handler ■ suppressJoinFailure ■ targets/sources

Features	Description
<receive>/<reply>	Business processes use receive activities and corresponding reply activities to provide web services to partners Supports: ■ association between Receive and Reply activities done by using MessageExchange attribute ■ start activity ■ variable ■ correlation ■ indicating fault in Reply ■ virtual assign: fromPart and toPart Does not support: ■ targets/sources
<assign>	Supports: ■ <from variable=”ncname” part=”ncname”?/> ■ <from> <expression>general-expr</expression> </from> ■ <from> <literal> ... literal value ... </literal> </from> ■ <to variable=”ncname” part=”ncname”?/> ■ <to queryLanguage=”anyURI”?>query</to> Does not support: ■ <from partnerLink=”ncname” endpointReference=”myRole|partnerRole”/> ■ <from variable=”ncname” property=”qname”/> ■ <to partnerLink=”ncname”/> ■ <to variable=”ncname” property=”qname”/> ■ <copy keepSrcElementName=”yes|no”?>: keepSrcElementName attribute is not supported. ■ <assign validate=”yes|no”?>: validate is not supported. ■ atomic assign ■ bpel standard faults ■ bpel:doXslTransform ■ bpel:getVariableProperty ■ expressionLanguage is limited to XPath 1.0

Features	Description
Signaling faults: <throw>	Supports: ■ Throw activity Does not supports: ■ targets/sources
<faultHandlers>	Supports: ■ Handling faults at Process and Scope level ■ Handling faults caused by invoke and throw ■ Handling faults with associated data defined using WSDL message types ■ Handling faults generated within fault handlers ■ Default fault handling behavior ¨C rethrowing unhandled faults to an enclosing scope Does not support, or has limited support: ■ Handling faults at Invoke level ■ Generating and handling standard faults ■ Handling and sending server faults - faults not defined on the WSDL operation Server faults caused due to invoke can be caught using CatchAll. Faults not handled within the business process are sent to the caller with an XML message. A standardized way of communicating and catching these faults is not supported.
<wait>	Supports: ■ Wait activity Does not support: ■ targets/sources ■ expressionLanguage
<empty>	Supports: ■ Empty activity Does not support: ■ targets/sources
<exit>	Supports: ■ Exit activity Does not support: ■ targets/sources
<sequence>	Supports: ■ Sequence activity Does not support: ■ targets/sources
<if>	Supports: ■ If activity Does not support: ■ targets/sources ■ expressionLanguage

Features	Description
<while>	Supports: ■ While activity Does not support: ■ targets/sources ■ expressionLanguage
<pick>	Supports: ■ Pick used as start activity ■ association of onMessage by using messageExchange ■ onMessage variable ■ onMessage correlation ■ onMessage virtual assign: fromPart ■ onAlarm Does not support: ■ targets/sources
<flow>	Flow activity provides synchronization and concurrency. Supports: ■ concurrency Does not support: ■ links ■ targets/sources
<sequence>	Supports: ■ variable ■ fault handler ■ event handler ■ partnerLink ■ correlation Does not support: ■ compensation handler ■ termination handler ■ targets/sources

Features	Description
<correlationSets>/<correlations>	Supports: ■ Receive as initiating activity and/or correlating activity ■ correlated Receive within Flow ■ correlated Invoke within Flow ■ correlated Invoke within While ■ Invoke correlation attribute pattern: in, out, out-in ■ correlated Reply within Flow ■ onMessage as initiating activity and/or correlating activity ■ correlated onMessage within Flow ■ recycling correlation at process level ■ In flow, with multiple initiating activities with createInstance ”yes”, all those multiple initiating activities should use only ”join” initiate flag Does not support: ■ correlations defined on Scope and their usage ■ correlated Receive within While ■ correlated Reply within While ■ correlated onMessage within While ■ request time out ■ duplicating operations
<forEach>	Supports: ■ ForEach parallel=”no” ■ CompleteCondition ■ countCompletedBranchesOnly=”yes|no” Does not support: ■ ForEach parallel=”yes” ■ targets/sources ■ Throw standard faults
<repeatUntil>	Supports: ■ RepeatUntil activity Does not support: ■ targets/sources ■ expressionLanguage

Features	Description
<eventHandlers>	Supports: ■ On Process level ■ On Scope level ■ OnEvent ■ OnAlarm ■ OnAlarm with RepeatedEvery Does not support: ■ Fault handling ■ Throw standard faults