Oracle® Java CAPS Java EE Service Engine User's Guide

Part No: 821–2613
January 2011

ORACLE

Copyright © 2009, 2011, Oracle and/or its affiliates. All rights reserved.

License Restrictions Warranty/Consequential Damages Disclaimer

This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited.

Warranty Disclaimer

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

Restricted Rights Notice

If this is software or related documentation that is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, the following notice is applicable:

U.S. GOVERNMENT RIGHTS

Programs, software, databases, and related documentation and technical data delivered to U.S. Government customers are "commercial computer software" or "commercial technical data" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, the use, duplication, disclosure, modification, and adaptation shall be subject to the restrictions and license terms set forth in the applicable Government contract, and, to the extent applicable by the terms of the Government contract, the additional rights set forth in FAR 52.227-19, Commercial Computer Software License (December 2007). Oracle America, Inc., 500 Oracle Parkway, Redwood City, CA 94065.

Hazardous Applications Notice

This software or hardware is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications that may create a risk of personal injury. If you use this software or hardware in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy, and other measures to ensure its safe use. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software or hardware in dangerous applications.

Trademark Notice

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Intel and Intel Xeon are trademarks or registered trademarks of Intel Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. AMD, Opteron, the AMD logo, and the AMD Opteron logo are trademarks or registered trademarks of Advanced Micro Devices. UNIX is a registered trademark of The Open Group in the United States and other countries.

Third Party Content, Products, and Services Disclaimer

This software or hardware and documentation may provide access to or information on content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services.

120126@25097

Contents

Using the Java EE Service Engine in a Project 5
About the Java EE Service Engine 5
Java EE Service Engine Features 7
Java EE Service Engine Limitations 8
Java EE Service Engine Use Case Scenarios 8
Java EE Service Engine as Service Provider and Service Consumer 9
    Java EE Service Engine as a Service Provider 9
    Java EE Service Engine as a Service Consumer 9
Java EE Service Engine Example Scenario 10
    Scenario Message Flow 11
Configuring and Starting the Java EE Service Engine 11
▼ To Start the Java EE Service Engine from the NetBeans IDE 12
▼ To Start the Java EE Service Engine from the Admin Console 12
▼ To Start the Java EE Service Engine Using Command Line Interface 13
    Installing Java EE Service Engine Using Command Line Interface 13
    Other Operations Using the Command Line Interface 13
Administering the Java EE Service Engine 14
▼ To View the General Properties 14
    Java EE Service Engine Log Management 14
Java EE Service Engine Deployment Artifacts 16

Using the Java EE Service Engine in a Project

This guide provides an overview of the Java EE Service Engine and its relationship with the application server and JBI runtime environment. It includes details that are necessary to start the Java EE Service Engine in a JBI project.

What You Need to Know

These links take you to what you need to know before you use the Java EE Service Engine.

- "About the Java EE Service Engine" on page 5
- "Java EE Service Engine Features" on page 7
- "Java EE Service Engine Limitations" on page 8
- "Java EE Service Engine Use Case Scenarios" on page 8
- "Java EE Service Engine as Service Provider and Service Consumer" on page 9
- "Java EE Service Engine Example Scenario" on page 10

Using the Java EE Service Engine

These links take you to topics containing information about using the Java EE Service Engine in a project.

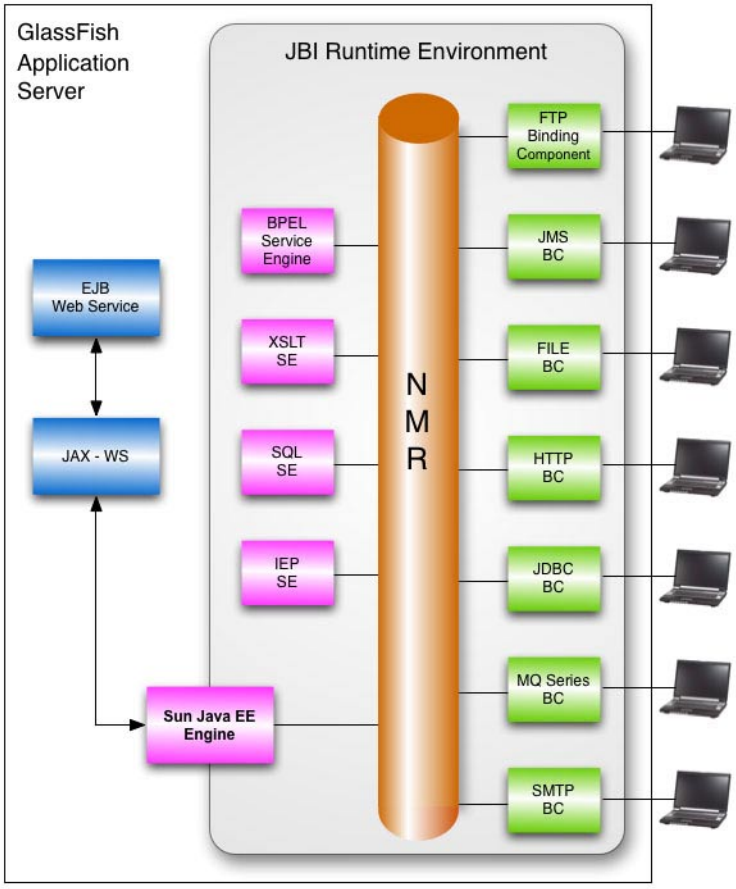
- "Configuring and Starting the Java EE Service Engine" on page 11
- "Administering the Java EE Service Engine" on page 14
- "Java EE Service Engine Deployment Artifacts" on page 16

About the Java EE Service Engine

The Java EE Service Engine is a JSR 208-compliant JBI runtime component that connects Java EE web services to JBI components. The framework of the Java EE Service Engine is defined by the inter-connectivity of GlassFish and the JBI runtime environment, which is installed as part of the application server. The Java EE Service Engine acts as a bridge between the application server and the JBI runtime environment, facilitating interaction between Java EE and JBI components.

Without the Java EE Service Engine, a Java EE component that is deployed as a servlet or Enterprise Java Beans web service can only be accessed from the HTTP SOAP transport. With the application server and JBI runtime environment working together, Java EE components can more readily access an array of binding components for transport or service engines for business logic, processing, transformation, and routing services. Enterprise beans and servlets that are packaged as web services and deployed on the application server are transparently exposed as service providers in JBI environment. Therefore the Java EE components can consume services exposed in JBI environment using the Java EE Service Engine without being aware of the underlying binding or protocol. This in-process communication between components of the application server and JBI runtime components results in increased request processing speed because all component interactions occur within the application server.

The following illustration shows the relationship between the application server, the JBI runtime environment, and the Java EE Service Engine. Acting as either a service provider or as a service consumer, the Java EE Service Engine communicates directly with the Normalized Message Router (NMR) and EJB web services using JAX-WS. The power and versatility of the Java EE web service communication continues to expand when additional JBI components are added to the runtime environment.



Java EE Service Engine Features

The features of theJava EE Service Engine are:

- Java EE service units - Using Java EE service units, end users can deploy and manage a composite application as a single entity: a JBI service assembly.
- Transactions support - Enables Java EE web services and JBI services to participate in a single transaction. For example, a BPEL process can call an EJB bean that updates a database, all in the same transaction.
- Security support - Enables separate JBI components to make use of a single authentication mechanism. In practice, a user signs on once, and the JBI system propagates the security credentials to the various JBI components as needed.
- Code upgrade to JAX-WS 2.1

- Endpoint enabling through `jbi.xml`
- Support for RPC/Literal style operations
- Direct invocation of endpoints
- Packaging Java EE applications as part of the composite application
- Cluster support - Enables you to deploy a Java EE service unit to an application server cluster. When the JBI service assembly is deployed to the cluster, the Java EE service units are also deployed to each server instance in the cluster.
- Message properties propagation — This new feature supports propagation and reception of Normalized message properties. This feature allows additional properties to be propagated between Java EE components and JBI components. Propagation and reception of these message properties is handled by the Java EE Service Engine. For example, a web service client can pass additional message properties while invoking a JBI endpoint (for example, BPEL endpoints). The endpoint can use these properties and also send back additional properties to the client.

For more information on the Java EE Service Engine Features, see the article Bridging Java EE Web Services and JBI Components.

Java EE Service Engine Limitations

The Java EE Service Engine has the following limitations:

- WSDL 2.0 is not supported.
- The JBI Message Exchange Patterns “Robust In-Only” and “Optional In/Out” are not supported.

Java EE Service Engine Use Case Scenarios

JBI components are typically used to handle high-level business operations. The Java EE Service Engine becomes necessary in situations where complex Java EE-based business logic needs to communicate with JBI components. In these situations, the Java EE Service Engine facilitates the transfer of data between components. Some examples of this data exchange include:

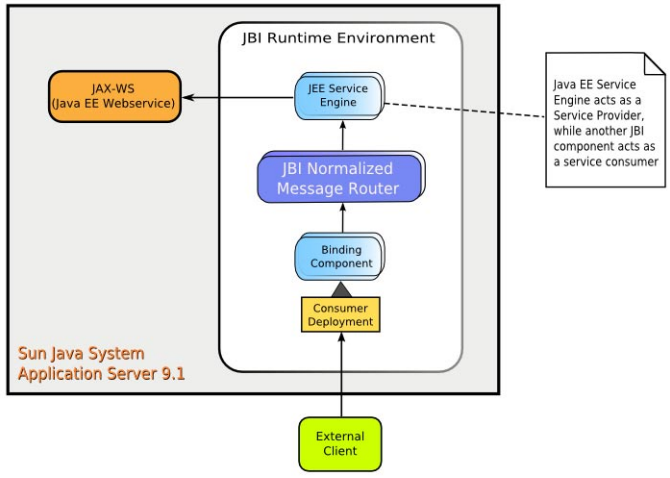
- BPEL Service Engine calling an Enterprise Java Bean (EJB) web service
- Message-driven bean (MDB) or servlet calling a BPEL process
- EJB web service called through a JMS transport using the JMS Binding Component
- Java EE components calling web services using the FTP Binding Component
- Java EE components making web service calls through SMTP transport using the SMTP Binding Component

Java EE Service Engine as Service Provider and Service Consumer

JBI uses the abstract service model as the main basis of component interactions. As with other JBI runtime components, the Java EE Service Engine can act either as a service *provider* that *performs* a service, or as a service *consumer* that *invokes* a service.

Java EE Service Engine as a Service Provider

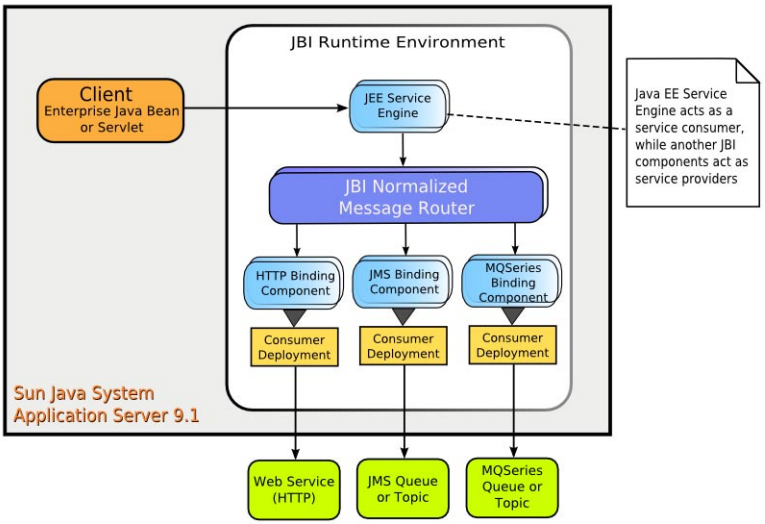
The Java EE Service Engine functions as a service provider by enabling an endpoint in the NMR. When a Java EE web service is deployed, the deployment runtime of application server notifies the Java EE Service Engine so that an endpoint is enabled in the NMR of the JBI runtime. The notification enables any component deployed in the NMR to access the Java EE web service. For example, a BPEL application running inside the BPEL service engine can access the Java EE web service by sending a normalized message to the endpoint enabled by the Java EE Service Engine. This way of accessing Java EE web services is an alternative to the normal web service client access defined by JAX-WS.



Java EE Service Engine as a Service Consumer

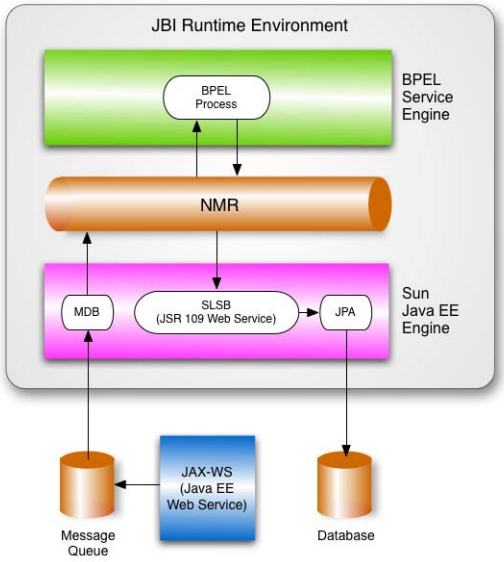
When a Java EE application needs to access an endpoint of a service provider deployed in the JBI runtime environment, the Java EE Service Engine communicates with the application server and the NMR in the JBI environment. In this case, the Java EE Service Engine normalizes the SOAP message that otherwise would have been used in the JAX-WS communication and sends it to the NMR. This normalized message is handled by a service that has been enabled by a

service provider deployed in the JBI runtime environment. The following illustration depicts three JBI binding components acting as service providers, offering application-specific content to external clients.



Java EE Service Engine Example Scenario

The following example shows how the Java EE Service Engine can be used in a business integration.



Scenario Message Flow

The example scenario message flow works as follows:

- The client application, acting as a web service, sends a message to a message queue.
- The message is picked up by a message driven bean (MDB).
- Upon receiving the message, the MDB contacts the NMR using the Java EE Service Engine, for the service endpoint that is exposed by the BPEL process.
- When the MDB executes, the BPEL application that is hosted by the BPEL service engine, contacts the NMR to find its partner services.
- In the example, only one partner service, the stateless session bean (JSR 109 web service) is hosted in the same Application Server JVM instance. The JSR 109 web service's is enabled in the NMR by the Java EE Service Engine when the stateless session bean is deployed.
- The stateless session bean then uses Java EE persistence APIs to access the database.

Configuring and Starting the Java EE Service Engine

The Java EE Service Engine requires no configuration. The service engine starts automatically whenever a JBI service assembly that contains a Java EE Service Unit is deployed.

You can start the Service Engine in the following three ways:

- "To Start the Java EE Service Engine from the NetBeans IDE" on page 12
- "To Start the Java EE Service Engine from the Admin Console" on page 12
- "To Start the Java EE Service Engine Using Command Line Interface" on page 13

Note – When working with the service engine in the NetBeans IDE, you need to start the component from NetBeans, as described in the first procedure below.

▼ To Start the Java EE Service Engine from the NetBeans IDE

1. **From the NetBeans IDE's main page, in the Services window, expand the `GlassFish V2` → `JBI` → `Service Engines` nodes.**

2. **Right-click sun-javaee-engine and choose `Start` from the pop-up menu.**

   The following message appears in the Output window:

   ```
   Engine sun-javaee-engine has been started.
   ```

▼ To Start the Java EE Service Engine from the Admin Console

1. **In the NetBeans Services window, log in to the GlassFish Administrator Console by right-clicking your application server and choosing View Admin Console.**

   Ensure that the GlassFish application server is running. A green arrow icon next to the server node indicates that the server is running. If the server is not running, right click the server name and choose Start from the pop—up menu.

2. **You can also open the Admin Console from your web browser using the correct URL, for example: `http://localhost:4848`.**

   The default log in username is`admin`; the password was defined during installation.

   The GlassFish Admin Console opens in a new browser window.

3. **On the left pane under the `JBI` node expand `Components` and choose `sun-javaee-engine`.**

   The Java EE Service Engine properties page opens.

4. **Click on the `Start` button on the General tab page to start the Java EE Service Engine.**

▼ To Start the Java EE Service Engine Using Command Line Interface

1. **Navigate to the location where you have installed Java CAPS and open the folders `appserver`→ `bin`.**
2. **Double click `asadmin.bat` file.**
3. **At the asadmin command prompt type `start-jbi-component sun-javaee-engine`.**

Installing Java EE Service Engine Using Command Line Interface

The Java EE Service Engine comes installed and configured when the GlassFish application server is installed. If you uninstall the Java EE Service Engine accidentally, you can use the CLI (Command Line Interface) to install it again manually.

Type the following command:

```
asadmin install-jbi-component
[ApplicationServer-InstallDir]/jbi/components/sun-javaee-engine/appserv-jbise.jar
```

Other Operations Using the Command Line Interface

Using the CLI we can perform many operations.

- To stop the Java EE Service Engine, type:

  ```
  asadmin stop-jbi-component sun-javaee-engine
  ```

- To shut down the Java EE Service Engine, type:

  ```
  asadmin shut-down-jbi-component sun-javaee-engine
  ```

- (Optional) To uninstall the Java EE Service Engine, type:

```
asadmin uninstall-jbi-component sun-javaee-engine
```

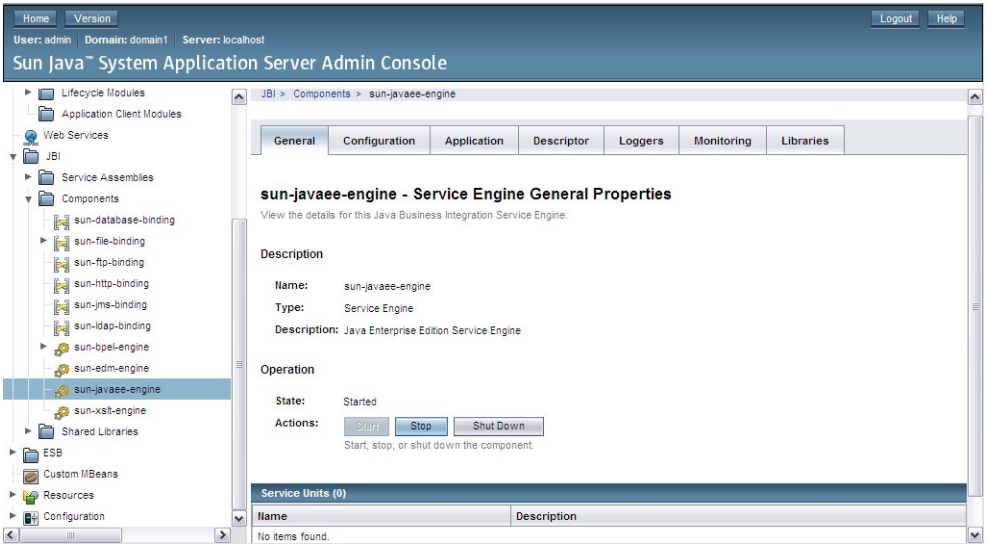
Administering the Java EE Service Engine

You can administer the Java EE Service Engine instances using the GlassFish Admin Console tools. These tools enable you to monitor and change the Java EE Service Engine's variable values, and suspend, resume, or terminate instances.

▼ To View the General Properties

- **Log in to the GlassFish Administrator Console and on the left pane under the `JBI` node expand `Components` and choose `sun-javaee-engine`.**

  The Java EE Service Engine properties page opens. Here you can view the details of the Java EE Service Engine, Service Units, and perform Start, stop and shut down operations.



Java EE Service Engine Log Management

The NetBeans IDE provides the ability to define logging for process activities.

Logging is used to write specified expression values or partner links endpoint reference information to the server log. The log level for the Java EE Service Engine is specified through the GlassFish Admin Console.

▼ To Set the Log Level for the Java EE Service Engine

1 **Access the Java EE Service Engine General Properties page.**

For more information see"To View the General Properties" on page 14.

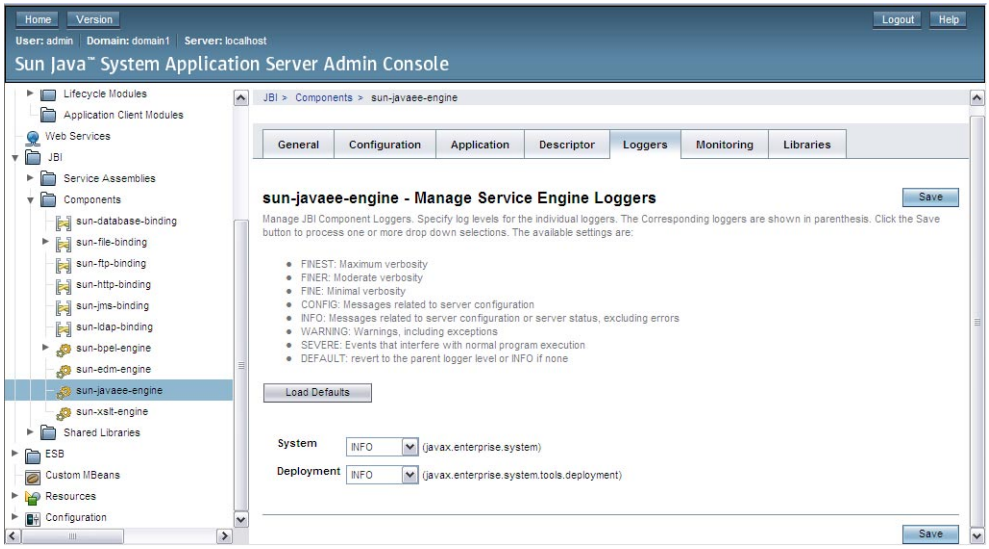
2 **Once on the General Properties page, click the `Loggers` tab.**

3 **Choose the appropriate log level for the sun-javaee-engine from the drop down list.**

If logging is defined for a process activity, and the log level specified for it corresponds to the log level set for the Java EE Service Engine, after you perform a test run of the process, the selected variable value will be written to the server log file.

The following levels of logging are available:

- FINEST
- FINER
- FINE
- CONFIG
- INFO
- WARNING
- SEVERE
- OFF



▼ To View a Log File

1. **From the NetBeans IDE's main page, in the Services window, under the Servers node, right-click the GlassFish application server node and choose `View Server Log` from the pop—up menu.**

   The GlassFish server log opens in the Output window. The activity message value is included in the log. You can use Search to find the log information. Note that some overhead information is hidden.

2. **Another alternate method is to view the server log file is in a text editor and see the full information. Navigate to `application server installation` directory → `domains/domain1` and open the `server.log` file with the text editor.**

   The information provided in the log includes the following points, divided with the vertical bar:

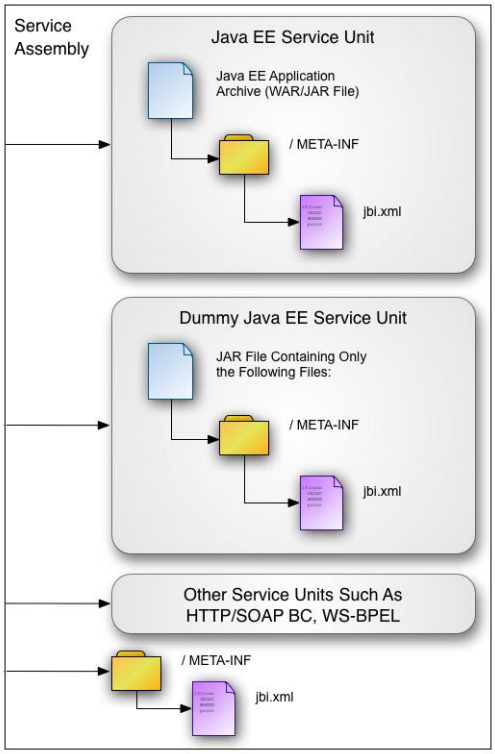
   - Date and time of the entry
   - Log level
   - Manager type
   - Thread

Java EE Service Engine Deployment Artifacts

A composite business application can contain Java EE applications. Users prefer to deploy and manage composite applications as a single entity and would prefer not to deploy the Java EE applications separately from the JBI service assembly. Java EE service units can make Java EE applications part of the JBI service assembly, enabling them to be deployed and managed as one entity. By definition, a Java EE service unit is a Java EE application that can be bundled as part of the JBI service assembly. So, when the JBI service assembly is deployed, the Java EE Service Engine takes care of deploying any Java EE applications that are bundled in the service assembly.

In addition, any life cycle operation that is applied to the JBI service assembly (start, stop, undeploy, and so on) results in the same life cycle operation being applied to the bundled Java EE applications. The difference between a normal Java EE application and the Java EE Service unit is that the latter contains an additional descriptor file named `jbi.xml`. This JBI runtime descriptor describes the services provided and consumed by a Java EE application in the JBI environment.

As shown in the following illustration, a composite application can contain various service units, where each unit needs to be deployed into the proper component of the JBI environment. For example, if the application needs to read input from a file and execute a BPEL process, then your composite business application contains at least two units: one unit is deployed into the File Binding Component , and another unit is deployed into the BPEL Service Engine.


Service Assembly
Java EE Service Unit
Java EE Application Archive (WAR/JAR File)
/ META-INF
jbi.xml
Dummy Java EE Service Unit
JAR File Containing Only the Following Files:
/ META-INF
jbi.xml
Other Service Units Such As HTTP/SOAP BC, WS-BPEL
/ META-INF
jbi.xml