BEA WebLogic Server ®
Using FastSwap ™ to Minimize Redeployment

Version: 10.3 Tech Preview

Document Date: October 2007

Table of Contents

Overview of Class Redefinition

Web Application Developers expect to make changes to a deployed application and see the changes immediately by refreshing the browser. On the Java EE side of the world, developers have to typically go through the following cycle to see their changes in action.

Edit → Build → Deploy → Test

These steps along with the many descriptor elements make Java EE seem complex and heavy weight to developers. Among these steps, the build and deploy cycle are necessitated by the Java language and the application server being employed. IDEs are trying to make the Edit and Build step seamless by providing incremental compilation support and application servers must try and make the deploy and test cycles seamless.

Our goal:Eliminate the deploy cycle for changes to Java Classes in development mode.

Hasn't this been attempted before?

Yes. In earlier releases, WebLogic Server shipped with a *ChangeAwareClassLoader* which detects changes to classes on disk and refreshes/reloads the entire ClassLoader. This certainly does eliminate the deploy cycle but is time consuming. Change-aware class loading involves various steps

1. Destroy all running Servlets/Filters and Listeners (*ServletContextListeners*, *HttpEventListeners* etc)
2. Create a new ClassLoader and load all the necessary classes. (or load them lazily on demand)
3. Initialize listeners, Servlets and Filters.

After this process, the request that was paused for the above steps is served. The disadvantage of this approach is that all state is lost and must be recreated before serving the request. This can take a long time depending on the number of classes loaded in the specific classloader, any static data being held, and the number of instances.

What does FastSwap provide?

Java SE 5 introduces the ability to redefine a class at runtime without dropping its ClassLoader or abandoning existing instances. This will allow containers to reload altered classes without disturbing running applications, vastly speeding up iterative development cycles and improving the overall development and testing experiences. The usefulness of Java SE 5's dynamic class redefinition is severely curtailed, however, by the restriction that the shape of the class – its declared fields and methods – cannot change. The purpose of FastSwap is to remove this restriction in WebLogic Server, allowing the dynamic redefinition of classes with new shapes to facilitate iterative development.

Key benefit: With FastSwap, Java classes are redefined in-place without reloading the ClassLoader thus having the huge advantage of fast turnaround times. This means that developers do not have to sit and wait for an application to redeploy and then navigate

back to wherever they were in the Web page flow. They can make their changes, auto compile, and then see the effects immediately.

Enabling FastSwap in your application

To enable FastSwap in your application, edit the `weblogic-application.xml` and add the following element.

```
<fast-swap>true</fast-swap>
```

Note: For Tech Preview, this element is available in the `weblogic.xml` descriptor instead of the `weblogic-application.xml` descriptor. For the final release, this element will be available in `weblogic-application.xml`, and not in `weblogic.xml`.

In weblogic.xml, adding the following snippet enables fast-swap

```
<container-descriptor>
   <fast-swap>true</fast-swap>
</container-descriptor>
```

Supported Application Configurations

- FastSwap is only supported when the server is running in development mode. It is automatically disabled in production mode.
- Only changes to class files in exploded directories are supported. Modifications to class-files in archived applications as well as archived jars appearing in the application's classpath are not supported. Examples are as follows:
  - When a web application is deployed as an archived war within an ear, modifications to any of the classes are not picked up by the FastSwap agent.
  - Within an exploded web application, modifications to Java classes are only supported in the WEB-INF/classes directory; the FastSwap agent does not pick up changes to archived jars residing in WEB-INF/lib.

Overview of the process of FastSwap with an example

Step1: Once FastSwap is enabled at the descriptor level, an appropriate ClassLoader is instantiated when the application is deployed to WebLogic Server.

Step 2: The user/developer opens up a browser to see the application at work. He/She then modifies adds/edits/deletes methods/classes (see the section on Limitations) and compiles them. (We recommend using an IDE such as Eclipse or IntelliJ and setting the compile-on-save option so that java files are compiled on saving. Please Note that the FastSwap agent does not compile Java files)

Step 3: The user then hits refresh on the browser or sends a new request to the application. The FastSwap agent tries to find all classes that have been modified since the last iteration by looking at all directories in the classpath. Considering an exploded

application with a single webapp, the following directories are examined for any class file modifications based on their timestamps:

```
ExampleApp/APP-INF/classes
ExampleApp/webapp/WEB-INF/classes
```

The FastSwap agent redefines the modified classes in the application and then serves the request.

Application Types and Changes Supported with FastSwap

For Tech Preview, only Web applications (WARs) deployed in an exploded format are supported with FastSwap. For the final release, FastSwap is supported with POJOs (JARs), Web applications (WARs) and enterprise applications (EARs) deployed in an exploded format. FastSwap is not supported with resource adapters (RARs).

The following types of changes are supported with FastSwap:

- Addition of static methods.
- Removal of static methods.
- Addition of instance methods.
- Removal of instance methods.
- Changes to static method bodies.
- Changes to instance method bodies.
- Addition of static fields.
- Removal of static fields.
- Addition of instance fields.
- Removal of instance fields.

The following table lists detailed change types supported with FastSwap:

Scope	Java Change Type	Supported	Notes
Java Class	Add method	Yes	Addition of the `finalize` method not supported.
Instance (non-abstract)	Remove Method	Yes	Removal of the `finalize` method not supported.
	(a) Add field	Yes	
	(b) Remove field	Yes	
	(c) Change Method Body	Yes	

	(d) Add Constructor	Yes	
	(e) Remove Constructor	Yes	
	(f) Change Field modifiers	Yes	
	(g) Change Method modifiers	Yes	
Class level (static)	Add Method	Yes	
	Remove Method	Yes	
	Change Method Body	Yes	
Class Hierarchy Changes	Change list of implemented Interfaces	No	
	Change extends “SuperClass”	No	
Abstract Java Class	Add Abstract method	Yes	
	Delete abstract method	Yes	
	All other types of changes listed above (a through g)	Yes	
“final” Java Class	Same supported changes listed above	Yes	
“final” Java method	Add/Remove	Yes	
“final” Java field	Add/Remove	Yes	
Enum	Add constants	No	
	Remove constants	No	
	Add/Remove methods	No	
Anonymous Inner Class	Add/Remove fields	NA	Not supported by the Java Language

Inner Class			Specification
	Add/Remove methods	No	
Static Inner Class	Same supported changes listed above	Yes	
Member Inner classes (non-static inner classes)	Same supported changes listed above	Yes	
Local Inner Classes	Same supported changes listed above	Yes	
Java Interface	Add method	Yes	
Java Reflection	Access Existing fields/methods	Yes	
	Access New Methods	No	New Methods are not seen via reflection and some synthetic methods are exposed
	Access New fields	No	New Fields are not seen via reflection
Annotations on Classes	Add or remove method/field annotations	No	
Annotation Type	Add/Remove methods/attributes	No	
Exception Classes	Same supported changes listed above	Yes	
EJB Interface	Add/Remove method	No	Changes to EJB interfaces involve reflection, which is not fully supported.
EJB 3.0 Session/MDB EJB Implementation Class	Add/Remove method	Not for Tech Preview	
	Add/Remove fields	Not for Tech Preview	

		Preview	
EJB 3.0 EntityBean	Add/Remove method	Not for Tech Preview	
	Add/Remove fields	Not for Tech Preview	
EJB Interceptors	Add/Remove methods	Not for Tech Preview	
	Add/Remove fields	Not for Tech Preview	

Limitations

- Java Reflection results do not include newly added fields and methods and include removed fields and methods. As a result of this, use of the reflection API on the modified classes can result in undesired behavior.
- Changing the hierarchy of an already existing class is not supported by FastSwap. Example: a) Changing the list of implemented interfaces of a class. b) Changing the superclass of a class is not supported.
- Addition or Removal of Java Annotations is not supported by FastSwap, since this is tied to reflection changes mentioned above.
- Addition or Removal of methods on EJB Interfaces is not supported by FastSwap since an EJB Compilation step is required to reflect the changes at runtime.
- Addition or Removal of constants from Enums not supported in this release.
- Addition or Removal of the `finalize` method is not supported.

How to Handle Unsupported Changes

When FastSwap is enabled, after you recompile a class, FastSwap attempts to redefine classes in existing classloaders. If redefinition fails because your changes fall outside the scope of supported FastSwap changes, the JVM throws an UnsupportedOperationException in the server window and in the server log. Your application will not reflect the changes, but will continue to run.

To implement your changes, you can redeploy the application or affected modules (partial redeploy), depending on the application type and the extent of your changes.