
# BEA WebLogic Server R
# Spring Integration Enhancements

Version: 10.3 Tech Preview

Document Date: October 2007

# Table of Contents

## Overview of Spring Enhancements

This document covers Spring integration improvements for the WebLogic Server 10.3 release.

## Spring extension DI and AOP for WLS

Java EE specification added Dependency Injection (DI) to Web and EJB container, and interceptors (a form of AOP) to EJB container. The Spring container is the leader in DI and AOP. WebLogic Server (WLS) is using Pitchfork to bridge with Spring to provide DI and interceptor to the WLS Java EE container. The integration not only satisfies the standard specification requirement, it also creates possibility for extension to the JavaEE5 specification as the Spring framework provides a richer set of features in terms of DI and AOP.

WLS takes advantage of these integration features, provided the extension to the standard JavaEE5 DI and AOP. This extension provides DI and AOP to EJB instance and Web components that include the servlet listener and filter. In order to maintain server compliance, the standard out of box WLS installation only provides the standard JavaEE5 DI and AOP.

To enable the Spring extension with WLS, do the following:

1. Download a version of Spring and its dependencies. At minimum, you will need the following jars: spring.jar, aspectjweaver.jar, commons-logging.jar, log4j-1.2.14.jar and pitchfork.jar. You can add other jars if necessary.
2. Add these jars to the Weblogic Server classpath.

   NOTE:

   a. The WLS Web and EJB containers use these jars to provide container service (DI and interceptor). Download the version of Spring that is certified by BEA.

   b. Your applications (ear, war, jar) use these jars since they are in the server classpath. You can configure your application to use the version of jars packaged with the application if you enable certain of the deployment descriptors.
3. Enable the spring extension by setting “component-factory-class-name” tag to “weblogic.application.SpringComponentFactory”. The tag exists in both EJB, Web and application descriptor with the module level descriptor overwriting the application level descriptor. If the tag is set to null (default), the Spring extension is disabled.
4. Provide the common Spring bean definition file with name spring-ejb-jar.xml or spring-web.xml, and place it in the META-INF directory of your application. These are the standard Spring bean definition files with specific names that the Weblogic container searches for. In order for Spring container to be aware of the EJB or servlet instance, the ?id? tag of the Spring Bean needs to set to the ejb-name for EJB or class-name of the web components.

## Spring Console

The Spring Console for Essex is based on RuntimeMBeans registered using WLS infrastructure and provide a more useful management functionality for Spring Beans.

Currently, the Spring console works with a WebApp. A future enhancement is planned to include a Spring console for EJBs.

To enable the Spring Console with WLS, do the following:

1. Deploy weblogic-spring.jar. The weblogic-spring.jar is a JavaEE optional package that is used by an application (ear, war). It creates the MBeans for your application during the deployment of your application. Deploy the weblogic-spring.jar by using the following command:

   java weblogic.Deployer -library -deploy –source
   YOUR_WL_HOME/server/lib/weblogic-spring.jar –targets
   YOUR_ServerName -adminurl YOUR_serverURL -user
   YOUR_username -password YOUR_password

2. Change the manifest of your applcation (ear, war) so it includes weblogic-spring.jar as JavaEE optional package. Do this by add the following lines to your MEAT-INF/Manifest.mf:

   Extension-List: WeblogicSpring

   WeblogicSpring-Extension-Name: weblogic-spring

   WeblogicSpring-Specification-Version: 10.0.0.0

   WeblogicSpring-Implementation-Version: 10.0.0.0

3. Link the ServletContextAttributeListener in the weblogic-spring.jar to your WebApp by adding the following line to your web.xml:

   ```
   <listener>
     <listener-class>
       weblogic.spring.monitoring.SpringServletContextAttributeListener
     </listener-class>
   </listener>
   ```

4. Copy spring-console-extension.jar from \server\lib\console-ext to domain-dir/console-ext.

## WebLogic Spring Security Integration

The WebLogic Server security system supports and extends Java EE security while providing a rich set of security providers that you can be customize to integrate with different security databases or security policies. Besides using standard Java EE security, application programmers can use a wide array of proprietary extensions that allow an application to tightly integrate with the security system. WLS packages several security provider offerings. For example, a choice of authentication databases that includes most of the popular LDAP servers, Active Directory, native Windows, and a built-in

authentication solutions. The built-in providers can be augmented with custom providers to integrate with nearly any authentication database, authorization mechanism, and credential mapping service.

The Spring security (acegi) provides security to a Spring application while providing a rich set of security providers.

For a blended J2EE and Spring application, rather than require authentication with both security frameworks, WLS and Spring security with work together. WLS security handles the authentication, and converts WLS principals to Spring GrantedAuthority through a mapper class. Once authenticated by WLS security, a user is authenticated for Spring security. You can then decide how to secure the objects in the application. One of the common practice is to secure Java EE resource with Weblogic security and secure Spring resource with Spring security.

The following is added to the web.xml to pluggin the Spring security:

```
  <filter>
    <filter-name>Acegi Filter Chain Proxy</filter-name>
    <filter-class>org.acegisecurity.util.FilterToBeanProxy</filter-class>
    <init-param>
      <param-name>targetClass</param-name>
      <param-value>org.acegisecurity.util.FilterChainProxy</param-value>
    </init-param>
  </filter>
  <filter-mapping>
   <filter-name>Acegi Filter Chain Proxy</filter-name>
   <url-pattern>/main/secure/*</url-pattern>
  </filter-mapping>
 <listener>
  <listener-class>org.springframework.web.context.ContextLoaderListener</listener-
class>
 </listener>
 <context-param>
  <param-name>contextConfigLocation</param-name>
  <param-value>
   /WEB-INF/applicationContext-acegi-security.xml
  </param-value>
 </context-param>
```

The applicationContext-acegi-security.xml is the configuration file for Spring security. The most important change to note is that the WeblogicAuthenticationFilter is added to the list of filters. This filter is responsible for converting the Weblogic principals to Spring GrantedAuthority based on the mapper. The mapper is configured as a property for the WeblogicAuthenticationFilter and it is injected at creation time.

The following is an example of the mapper class.

```
public class MyAuthorityGranter implements AuthorityGranter {
   public Set grant(Principal principal) {
      Set rtnSet = new HashSet();

      if (principal.getName().equals("fred@golf.com")) {
         rtnSet.add("ROLE_SUPERVISOR");
         rtnSet.add("IS_AUTHENTICATED_ANONYMOUSLY");
      }
      return rtnSet;
   }
}
```

**Note**: fred@golf.com in the weblogic domain is mapped to the ROLE_SUPERVISOR and IS_AUTHENTICATED_ANONYMOUSLY.