JBoss Web Framework Kit 1.1

Snowdrop 1.1 User Guide

Marius Bogoevici

Aleš Justin

JBoss Web Framework Kit 1.1 Snowdrop 1.1 User Guide

Author Marius Bogoevici
Author Aleš Justin
Editor Laura Bailey

This book is a user guide to Snowdrop 1.1 for use with JBoss Web Framework Kit 1.1.

Preface **v**
1. Document Conventions v
1.1. Typographic Conventions v
1.2. Pull-quote Conventions vi
1.3. Notes and Warnings vii
2. We Need Feedback! vii
1. What This Guide Covers **1**
2. Introduction **3**
2.1. Package Structure 3
3. Component usage **5**
3.1. VFS-enabled Application Contexts 5
3.2. Load-time weaving 6
3.3. The Spring Deployer 7
3.3.1. JBoss + Spring + EJB 3.0 Integration 7
3.3.2. Installation 7
3.3.3. Spring deployments 7
3.3.4. Deployment 7
3.3.5. Defining the JNDI name 8
3.3.6. Parent Bean factories 8
3.3.7. Injection into EJBs 8
A. Revision History **9**

Preface

1. Document Conventions

This manual uses several conventions to highlight certain words and phrases and draw attention to specific pieces of information.

In PDF and paper editions, this manual uses typefaces drawn from the *Liberation Fonts*[1] set. The Liberation Fonts set is also used in HTML editions if the set is installed on your system. If not, alternative but equivalent typefaces are displayed. Note: Red Hat Enterprise Linux 5 and later includes the Liberation Fonts set by default.

1.1. Typographic Conventions

Four typographic conventions are used to call attention to specific words and phrases. These conventions, and the circumstances they apply to, are as follows.

`Mono-spaced Bold`

Used to highlight system input, including shell commands, file names and paths. Also used to highlight keycaps and key combinations. For example:

> To see the contents of the file **`my_next_bestselling_novel`** in your current working directory, enter the **`cat my_next_bestselling_novel`** command at the shell prompt and press **`Enter`** to execute the command.

The above includes a file name, a shell command and a keycap, all presented in mono-spaced bold and all distinguishable thanks to context.

Key combinations can be distinguished from keycaps by the hyphen connecting each part of a key combination. For example:

> Press **`Enter`** to execute the command.
>
> Press **`Ctrl`**+**`Alt`**+**`F1`** to switch to the first virtual terminal. Press **`Ctrl`**+**`Alt`**+**`F7`** to return to your X-Windows session.

The first paragraph highlights the particular keycap to press. The second highlights two key combinations (each a set of three keycaps with each set pressed simultaneously).

If source code is discussed, class names, methods, functions, variable names and returned values mentioned within a paragraph will be presented as above, in **`mono-spaced bold`**. For example:

> File-related classes include **`filesystem`** for file systems, **`file`** for files, and **`dir`** for directories. Each class has its own associated set of permissions.

Proportional Bold

This denotes words or phrases encountered on a system, including application names; dialog box text; labeled buttons; check-box and radio button labels; menu titles and sub-menu titles. For example:

[1] https://fedorahosted.org/liberation-fonts/

Choose **System** → **Preferences** → **Mouse** from the main menu bar to launch **Mouse Preferences**. In the **Buttons** tab, click the **Left-handed mouse** check box and click **Close** to switch the primary mouse button from the left to the right (making the mouse suitable for use in the left hand).

To insert a special character into a **gedit** file, choose **Applications** → **Accessories** → **Character Map** from the main menu bar. Next, choose **Search** → **Find…** from the **Character Map** menu bar, type the name of the character in the **Search** field and click **Next**. The character you sought will be highlighted in the **Character Table**. Double-click this highlighted character to place it in the **Text to copy** field and then click the **Copy** button. Now switch back to your document and choose **Edit** → **Paste** from the **gedit** menu bar.

The above text includes application names; system-wide menu names and items; application-specific menu names; and buttons and text found within a GUI interface, all presented in proportional bold and all distinguishable by context.

Mono-spaced Bold Italic or ***Proportional Bold Italic***

Whether mono-spaced bold or proportional bold, the addition of italics indicates replaceable or variable text. Italics denotes text you do not input literally or displayed text that changes depending on circumstance. For example:

To connect to a remote machine using ssh, type **ssh *username*@*domain.name*** at a shell prompt. If the remote machine is **example.com** and your username on that machine is john, type **ssh john@example.com**.

The **mount -o remount *file-system*** command remounts the named file system. For example, to remount the **/home** file system, the command is **mount -o remount /home**.

To see the version of a currently installed package, use the **rpm -q *package*** command. It will return a result as follows: ***package-version-release***.

Note the words in bold italics above — username, domain.name, file-system, package, version and release. Each word is a placeholder, either for text you enter when issuing a command or for text displayed by the system.

Aside from standard usage for presenting the title of a work, italics denotes the first use of a new and important term. For example:

Publican is a *DocBook* publishing system.

1.2. Pull-quote Conventions

Terminal output and source code listings are set off visually from the surrounding text.

Output sent to a terminal is set in **mono-spaced roman** and presented thus:

```
books        Desktop   documentation  drafts  mss    photos   stuff  svn
books_tests  Desktop1  downloads      images  notes  scripts  svgs
```

Source-code listings are also set in **mono-spaced roman** but add syntax highlighting as follows:

```
package org.jboss.book.jca.ex1;

import javax.naming.InitialContext;

public class ExClient
{
   public static void main(String args[])
       throws Exception
   {
      InitialContext iniCtx = new InitialContext();
      Object         ref    = iniCtx.lookup("EchoBean");
      EchoHome       home   = (EchoHome) ref;
      Echo           echo   = home.create();

      System.out.println("Created Echo");

      System.out.println("Echo.echo('Hello') = " + echo.echo("Hello"));
   }
}
```

1.3. Notes and Warnings

Finally, we use three visual styles to draw attention to information that might otherwise be overlooked.

Note

Notes are tips, shortcuts or alternative approaches to the task at hand. Ignoring a note should have no negative consequences, but you might miss out on a trick that makes your life easier.

Important

Important boxes detail things that are easily missed: configuration changes that only apply to the current session, or services that need restarting before an update will apply. Ignoring a box labeled 'Important' won't cause data loss but may cause irritation and frustration.

Warning

Warnings should not be ignored. Ignoring warnings will most likely cause data loss.

2. We Need Feedback!

If you find a typographical error in this manual, or if you have thought of a way to make this manual better, we would love to hear from you! Please submit a report in JIRA: *http://jira.jboss.org/* against the product **JBoss Enterprise Application Platform** and component *Documentation*.

When submitting a bug report, be sure to mention the manual's identifier: JBoss Web Framework Kit *Snowdrop 1.1 User Guide*.

If you have a suggestion for improving the documentation, try to be as specific as possible when describing it. If you have found an error, please include the section number and some of the surrounding text so we can find it easily.

What This Guide Covers

Snowdrop is a utility package that contains JBoss-specific extensions to the Spring Framework. These extensions are either:

- extensions to Spring Framework classes that can be used wherever the generic implementations provided by the framework do not integrate correctly with JBoss Web Framework Kit.
- extensions for deploying and running Spring applications with JBoss Enterprise Application Platform, JBoss Enterprise Web Platform, and JBoss Enterprise Web Server.

This user guide aims to cover the functionality of Snowdrop, to describe its components, and to provide information on how to use it optimally for running Spring applications in JBoss Enterprise Application Platform, JBoss Enterprise Web Platform, and JBoss Enterprise Web Server.

Introduction

2.1. Package Structure

Snowdrop contains the following files:

`snowdrop-vfs.jar`
: A library that contains the support classes for resource scanning (scanning the classpath for bean definitions, or using "classpath*:"-style patterns).

`snowdrop-weaving.jar`
: A library that contains the support classes for load-time weaving.

`spring-deployer.zip`
: The Spring deployer, which bootstraps and registers the application contexts to be used by your Java EE applications.

Component usage

This chapter details how to use each of the components included in Snowdrop.

3.1. VFS-enabled Application Contexts

Note

From Spring 3.0 onward, the **`ApplicationContext`** implementations shipped with the Spring framework are VFS-compatible. The components described in this section are included with Snowdrop to provide backwards compatibility, but are not necessarily required.

The **`snowdrop-vfs.jar`** library supports resource scanning in the JBoss Virtual File System (VFS).

When Spring 2.5 performs resource scanning, it assumes that resources are either from a directory or a packaged JAR, and treats any URLs it encounters accordingly.

This assumption is not correct for the JBoss VFS, so Snowdrop provides a different underlying resource resolution mechanism by amending the functionality of the **`PathMatchingResourcePatternResolver`**.

This is done by using one of two **`ApplicationContext`** implementations provided by **`snowdrop-vfs.jar`**:

`org.jboss.spring.vfs.context.VFSClassPathXmlApplicationContext`
: Replaces the Spring **`org.springframework.context.support.ClassPathXmlApplicationContext`**.

`org.jboss.spring.vfs.context.VFSXmlWebApplicationContext`
: Replaces the Spring **`org.springframework.web.context.support.XmlWebApplicationContext`**.

In many cases, the **`VFSClassPathXmlApplicationContext`** is instantiated on its own, using something like:

```
ApplicationContext context =
new VFSClassPathXmlApplicationContext("classpath:/context-definition-file.xml");
```

The **`XmlWebApplicationContext`** is not instantiated directly. Instead, it is bootstrapped by either the **`ContextLoaderListener`** or the **`DispatcherServlet`**. In this case, the class used for bootstrapping must be used to trigger an instantiation of the VFS-enabled context.

To change the type of application context created by the **`ContextLoaderListener`**, add the `contextClass` parameter as shown in the following example code:

```
<context-param>
  <param-name>contextConfigLocation</param-name>
  <param-value>classpath*:spring-contexts/*.xml</param-value>
</context-param>
<context-param> <param-name>contextClass</param-name> <param-value>
 org.jboss.spring.vfs.context.VFSXmlWebApplicationContext </param-value> </context-param>
```

```
<listener>
  <listener-class>
    org.springframework.web.context.ContextLoaderListener
  </listener-class>
</listener>
```

To change the type of application context created by the **`DispatcherServlet`**, use the same **`contextClass`** parameter on the **`DispatcherServlet`** definition as shown:

```
<servlet>
  <servlet-name>spring-mvc-servlet</servlet-name>
  <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
  <init-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>/WEB-INF/mvc-config.xml</param-value>
  </init-param> <init-param> <param-name>contextClass</param-name> <param-value>
 org.jboss.spring.vfs.context.VFSXmlWebApplicationContext </param-value> </init-param> </
servlet>
```

Important: `ZipException`

If you encounter the `ZipException` when attempting to start the application, you need to replace the default **`ApplicationContext`** with one of the VFS-enabled implementations.

```
Caused by: java.util.zip.ZipException: error in opening zip file
...
at org.springframework.core.io.support.PathMatchingResourcePatternResolver
.doFindPathMatchingJarResources(PathMatchingResourcePatternResolver.java:448)
```

3.2. Load-time weaving

Note

From Spring 3.0 onward, load-time weaving on JBoss Enterprise Application Platform and JBoss Enterprise Web Platform is supported out of the box. The component described in this section can be used to facilitate backwards compatibility, but configuring a custom load-time weaver is not required when using Spring 3.0.

Load-time weaving support is provided by the **`snowdrop-weaving.jar`** library.

To perform load-time weaving for the application classes in Spring (either for using load-time support for AspectJ or for JPA support), the Spring framework needs to install its own transformers in the classloader. For JBoss Enterprise Application Platform, JBoss Enterprise Web Platform and JBoss Enterprise Web Server, a classloader-specific **`LoadTimeWeaver`** is necessary.

Define the **`JBoss5LoadTimeWeaver`** in the **`www`** Spring application context as shown here:

```
<context:load-time-weaver
 weaver-class="org.jboss.instrument.classloading.JBoss5LoadTimeWeaver"/>
```

3.3. The Spring Deployer

The Spring deployer allows you to bootstrap a Spring application context, bind it in JNDI, and use it to provide Spring-configured business object instances.

3.3.1. JBoss + Spring + EJB 3.0 Integration

Snowdrop contains a JBoss deployer that supports Spring packaging in JBoss Enterprise Application Platform, JBoss Enterprise Web Platform and JBoss Enterprise Web Server. This means it is possible to create JAR archives with a **`META-INF/jboss-spring.xml`** file to have your Spring bean factories deploy automatically.

EJB 3.0 integration is also supported. You can deploy Spring archives and inject beans created in these deployments directly into an EJB by using the `@Spring` annotation.

3.3.2. Installation

To install the Snowdrop JBoss deployer, unzip the **`jboss-spring-deployer.zip`** in the **`$JBOSS_HOME/server/$PROFILE/deployers`** directory of your JBoss Enterprise Application Platform or JBoss Enterprise Web Platform installation.

3.3.3. Spring deployments

You can create Spring deployments that work similarly to JARs, EARs, and WARs with the JBoss Spring deployer. Spring JARs are created with the following structure:

```
my-app.spring/
  org/
    acme/
      MyBean.class
      MyBean2.class
  META-INF/
    jboss-spring.xml
```

`my-app.spring` is a JAR that contains classes. A **`jboss-spring.xml`** file exists in the **`META-INF`** directory of the JAR. By default, the JBoss Spring deployer registers the bean factory defined in **`jboss-spring.xml`** into JNDI in a non-serialized form. The default JNDI name is the short name of the deployment file — in this case, **`my-app`**.

You do not have to create an archive. Instead, you can place your JAR libraries under **`$JBOSS_HOME/server/$PROFILE/lib`** and add an XML file of the form **`<name>-spring.xml`**, for example, **`my-app-spring.xml`**, into the **`deploy`** directory of your JBoss Enterprise Application Platform or JBoss Enterprise Web Platform installation. The default JNDI name will be the short name of the XML file; in this case, **`my-app`**.

3.3.4. Deployment

Once you have created a **`.spring`** or a *-spring.xml* file, copy it into the **`deploy`** directory of your JBoss Enterprise Application Platform or JBoss Enterprise Web Platform installation to deploy it into the JBoss runtime. You can also embed these deployments in an EAR, EJB-SAR, SAR, etc. since JBoss Enterprise Application Platform and JBoss Enterprise Web Platform support nested archives.

3.3.5. Defining the JNDI name

You can specify the JNDI name explicitly by putting it in the description element of the Spring XML.

```
<beans>
  <description>BeanFactory=(MyApp)</description>
  ...
  <bean id="springBean" class="example.SpringBean"/>
</beans>
```

`MyApp` will be used as the JNDI name in this example.

3.3.6. Parent Bean factories

Sometimes you want your deployed Spring bean factory to be able to reference beans deployed in another Spring deployment. You can do this by declaring a parent bean factory in the description element in the Spring XML, like so:

```
<beans>
<description>BeanFactory=(AnotherApp) ParentBeanFactory=(MyApp)</description>
...
</beans>
```

3.3.7. Injection into EJBs

Once an **`ApplicationContext`** has been successfully bootstrapped, the Spring beans defined in it can be used for injection into EJBs. To do this, the EJBs must be intercepted with the **`SpringLifecycleInterceptor`**, as in the following example:

```
@Stateless
@Interceptors(SpringLifecycleInterceptor.class)
public class InjectedEjbImpl implements InjectedEjb
{
 @Spring(bean = "springBeanName", jndiName = "SpringDao")
 private SpringBean springBean;

 /* rest of the class definition ommitted */
}
```

Appendix A. Revision History

Revision 0.1 Tue May 18 2010 Laura Bailey *lbailey@redhat.com*

Converted book to Publican format.