Snowdrop User Guide

by Marius Bogoevici and Aleš Justin

edited by Laura Bailey and Rebecca Newton

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

> 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 project **Snowdrop** and component *Documentation*.

When submitting a bug report, be sure to mention the manual's identifier: *Snowdrop 1.2 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 AS.
- extensions for deploying and running Spring applications with JBoss AS.

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 AS.

Supported Spring Version

Snowdrop 2.0 and subsequent micro-release versions (2.0.1, 2.0.2, etc.) should be used with Spring 2.5 (preferred version being Spring 2.5.6.SEC03) or Spring 3.0 (preferred version being 3.0.6.RELEASE and above). Later Spring versions (3.1, 3.2. etc.) may be compatible with Snowdrop 2.0 as long as they maintain binary compatibility with Spring 3.0. In such cases, the instructions regarding Spring 3.0 apply to Spring 3.1 and later as well.

Supported JBoss Version

Snowdrop 2.0 and subsequent micro-release versions (2.0.1, 2.0.2, etc.) should be used with JBoss AS 5.x, JBoss AS 6.x or JBoss AS 7.x.

Introduction

2.1. Package Structure

The Snowdrop distribution contains two types of artifacts:

Utility archives
: Libraries that contain utility classes and can be packaged in applications

Server-side components
: Pre-packaged components that can be installed in the application server as deployers (JBoss AS5, JBoss AS6) or subsystems (JBoss AS7).

The server-side components may include some of the utility libraries, where they are needed by the deployer or, respectively, subsystem.

Snowdrop provides the following utility libraries:

`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.

`snowdrop-deployers.jar`
: Contains utility classes used internally by the Spring deployer. It is not intended for direct use by developers.

`snowdrop-namespace.jar`
: Custom namespace for easier configuration of beans that need access to JBoss internals (JMX server locator, JCA activation spec and resource adapter for JMS endpoints)

Snowdrop provides the following pre-packaged deployers and subsystems:

`jboss-spring-deployer-as5.zip`
: This is a packaged distribution of the Spring deployer for JBoss AS5, which bootstraps and registers the application contexts to be used by your Java EE applications.

`jboss-spring-deployer-as6.zip`
: This is a packaged distribution of the Spring deployer for JBoss AS6, which bootstraps and registers the application contexts to be used by your Java EE applications.

`jboss-spring-subsystem-as7.zip`
: This is a packaged distribution of the Spring subsystem for JBoss AS7, 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). It must be included in Spring-based applications that use classpath and resource scanning.

When the Spring framework 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`. Both classes have configuration

options that allow users to replace the default application context type with a custom application context type.

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:44
```

3.2. The JBoss custom namespace

Starting with version 1.2, Snowdrop includes a custom Spring namespace for JBoss AS. The goals of this custom namespace is to simplify the development of Spring applications that run on JBoss, by reducing the amount of proprietary code and improving portability.

The amount of proprietary code is reduced because of replacing bean definitions that include references to specific JBoss classes with namespace-based constructs. All the knowledge about the proprietary classes is encapsulated in the namespace handlers.

The applications are more portable because certain proprietary classes may change when upgrading to a different version of the application server. In such cases, the runtime will be detected automatically by Snowdrop which will set up beans using the classes that are appropriate for that specific runtime.

The custom namespace can be set up as follows:

```
<beans xmlns="http://www.springframework.org/schema/beans"
      xmlns:jboss="http://www.jboss.org/schema/spring"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                        xsi:schemaLocation="http://www.springframework.org/schema/beans    http://
www.springframework.org/schema/beans/spring-beans.xsd
                    http://www.jboss.org/schema/snowdrop    http://www.jboss.org/schema/snowdrop/
snowdrop.xsd">
```

3.2.1. Accessing the default JBoss MBean Server

The default MBean server of JBoss AS can be accessed as follows:

```
<jboss:mbean-server/>
```

The bean will be installed with the default id 'mbeanServer'. If necessary, developers can specify a different bean name:

```
<jboss:mbean-server id="customName"/>
```

3.2.2. JCA/JMS support beans

Spring JMS message listeners (including message-driven POJOs) can use a JCA-based MessageListenerContainer. The configuration of a JCA-based listener container in Spring requires the setup of a number of beans based on application-server specific classes. Using the JBoss custom namespace, the ResourceAdapter and ActivationSpec configuration can be set up as follows:

```
    <jboss:activation-spec-factory id="activationSpecFactory" subscriptionName="jca-example"
useDLQ="false"/>

  <jboss:resource-adapter id="resourceAdapter"/>
```

which can be further used in a JCA message listener configuration:

```
<jms:jca-listener-container resource-adapter="resourceAdapter" acknowledge="auto"
                            activation-spec-factory="activationSpecFactory">
    <jms:listener destination="/someDestination" ref="messageDrivenPojo"
                  method="pojoHandlerMethod"/>
  </jms:jca-listener-container>
```

3.3. Load-time weaving

Note

From Spring 3.0 onward, load-time weaving on JBoss AS 5 and JBoss AS 6 is supported out of the box. However, on JBoss AS7 it is necessary to use the load-time weaver provided by Snowdrop. The component described in this section should be used with Spring 2.5 on any of the servers or when using Spring 2.5

and Spring 3.0 with JBoss AS7. In other cases, it is optional, but can be used to facilitate backwards compatibility.

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 `JBossLoadTimeWeaver` in the Spring application context as shown here:

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

3.4. 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.

3.4.1. JBoss + Spring + EJB 3.0 Integration

Snowdrop contains a JBoss deployer that supports Spring packaging in JBoss AS. This means that 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.

3.4.2. Spring Deployer Installation

3.4.2.1. Installation on JBoss AS 5 and JBoss AS 6

To install the Spring deployer, unzip the deployer archive in the `$JBOSS_HOME/server/$PROFILE/deployers` directory of your JBoss Application Server installation.

The Snowdrop Spring deployer requires the inclusion of the Spring libraries. If you are installing a version without dependencies or you want to include your own Spring version, you must ensure that you are including one of the following sets of Spring libraries in the `$JBOSS_HOME/server/$PROFILE/deployersspring.deployer` folder.

Please include at least the following jars from either the Spring 2.5.6.SEC03 or Spring 3.0.6.RELEASE distribution:

- spring-beans.jar (or org.springframework.beans.jar)

- spring-context.jar (org.springframework.context.jar)
- spring-core.jar (or org.springframework.core.jar)
- spring-web.jar (or org.springframework.web.jar)

3.4.2.2. Installation on JBoss AS 7

To install the Snowdrop Deployment subsystem, unzip the `jboss-spring-subsystem-as7.zip` file. Create the subsystem and Spring modules in JBoss AS7 by copying the contents of the `module-deployer` folder and one of the `module-spring-2.5` or `spring-3` folders in the `$JBOSS_HOME/modules` directory of your JBoss Application Server installation.

The above step will create two modules inside JBoss AS7:

`org.jboss.snowdrop:main`
: The module that contains the JBoss AS7 subsystem proper

`org.springframework.spring:snowdrop`
: A module that contains the Spring JARs that are required by Snowdrop. It can contain Spring 2.5 or Spring 3 JARs, dependending the which of the two modules has been copied in the step above. Users may add other JARs to the module, case in which they need to adjust the `module.xml` file accordingly. It is a dependency of `org.jboss.snowdrop:main`

If you are using the distribution without dependencies, or you wish to create your own version of the Spring module, then create a `$JBOSS_HOME/modules/org/springframework/spring/main` directory and copy one of the two `module.xml` files (from either the `module-spring-2.5` or the `module-spring-3` directory of the distribution and copy one of the following sets of files from the corresponding Spring distribution (either Spring 2.5.6.SEC03 or Spring 3.0.6.RELEASE).

For Spring 2.5:

- aspectjrt.jar
- aspectjweaver.jar
- aopalliance.jar
- spring-aop.jar
- spring-beans.jar
- spring-core.jar
- spring-context.jar
- spring-context-support.jar
- spring-web.jar

For Spring 3:

- aspectjrt.jar
- aspectjweaver.jar
- aopalliance.jar
- spring-aop.jar
- spring-asm.jar
- spring-beans.jar
- spring-core.jar
- spring-expression.jar
- spring-context.jar
- spring-context-support.jar
- spring-web.jar

The final step in the installation is to change `$JBOSS_HOME/standalone/configuration/standalone.xml` by including `<extension module="org.jboss.snowdrop"/>` inside the `<extensions>` element, as well as including `<subsystem xmlns="urn:jboss:domain:snowdrop:1.0"/>` inside the `<profile>` element.

3.4.3. 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 can also place 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`.

3.4.4. Deployment

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

3.4.5. 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.

3.4.6. 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>
```

3.4.7. 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 = "springBean", jndiName = "MyApp")
 private SpringBean springBean;

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

In this example, the EJB `InjectedEjbImpl` will be injected with the bean named `springBean`, which is defined in the `ApplicationContext`.

Appendix A. Revision History

Revision History

Revision 0.1-0 Tue May 18 2010 LauraBailey<lbailey@redhat.com>

Converted book to Publican format.