SpringSource dm Server™ User Guide

Rob Harrop
Paul Kuzan
Sam Brannen
Damilola Senbanjo
Paul Harris
Christopher Frost
Ben Hale
Glyn Normington
Juliet Shackell

2.0.5.RELEASE

Table of Contents

1. Installing dm Server 1
    1.1. Prerequisites 1
    1.2. Installing from the ZIP Download 1
    1.3. Post-installation steps 1
2. Starting and Stopping dm Server 11
    2.1. Starting SpringSource dm Server 11
    2.2. Starting in Clean Mode 11
    2.3. Starting in Debug Mode 12
    2.4. Starting with JMX Access Modifications 13
    2.5. Starting With a Custom Configuration Directory 21
    2.6. Stopping SpringSource dm Server 22
    2.7. Starting SpringSource dm Server When the Operating System Starts 23
3. Overview of the dm Server Kernel and User Region 25
    3.1. The dm Server Kernel 25
    3.2. The dm Server User Region 26
4. The dm Shell 29
    4.1. Using the dm Shell 29
    4.2. dm Shell Command Reference 31
5. The Admin Console 49
    5.1. Invoking the Admin Console 49
    5.2. Typical Admin Console Use Cases 51
6. The Provisioning Repository 61
    6.1. Overview of the Provisioning Repository 61
    6.2. Finding and Downloading Bundles from the SpringSource Enterprise Bundle Repository 63
    6.3. Configuring the repository 64
7. Serviceability 65
    7.1. Event log files 65
    7.2. Trace (Logging) 65
    7.3. Service Dumps 67
8. Working with Applications 69
    8.1. Deploying Artifacts 69
    8.2. Undeploying Artifacts 70
9. Configuring dm Server 73
    9.1. Configuring the dm Kernel and User Region 73
    9.2. Configuring Serviceability 78
    9.3. Configuring the Embedded Tomcat Servlet Container 82
    9.4. Configuring the Local Provisioning Repository 85
    9.5. Configuring a Hosted Repository 91
A. Event log codes 95
    A.1. Format of the event log codes 95
B. Known Issues 97
    B.1. Timeout During Startup Due to Firewall Settings 97
    B.2. OutOfMemoryError: PermGen space running on Sun VM 97

C. Further Reading 99

1. Installing dm Server

1.1 Prerequisites

The SpringSource dm Server requires Java SE 6 or later to be installed. Java is available from Sun and elsewhere.

1.2 Installing from the ZIP Download

Downloading the ZIP file

SpringSource dm Server is distributed as a ZIP file. This can be downloaded from here. Follow the instructions to obtain a username and password.

Installing

Linux

To install the SpringSource dm Server on Linux, unzip the distribution package to the desired installation directory. For example, to install into `/opt`:

```
prompt$ unzip springsource-dm-server-2.0.5.RELEASE.zip -d /opt
```

This creates a directory called `springsource-dm-server-2.0.5.RELEASE` under `/opt`.

SpringSource dm Server requires write access to the installation directory, in this case `/opt/springsource-dm-server-2.0.5.RELEASE`. Typically this means it must be run as the user that installed it, or the installation directory's ownership must be changed.

Microsoft Windows

To install the SpringSource dm Server on Windows, unzip the distribution package to the desired installation directory. You should use a zip application such as 7zip, not the built-in folder decompression. Note that both Windows and Java 5 have some issues with long file names and file paths, so we recommend installing to the root directory of your chosen drive.

1.3 Post-installation steps

Set environment variable variables

JAVA_HOME

The SpringSource dm Server uses the `JAVA_HOME` environment variable to locate the `java` executable. Configure this environment variable to point to the home directory of the Java 5 or Java 6 installation on your computer.

SERVER_HOME

As a convenience it is recommended that you create an environment variable that points to the SpringSource dm Server installation directory. Note that the SpringSource dm Server does not require that such an environment variable has been set. This variable may have any name of your choosing. The SpringSource dm Server's documentation assumes that the variable is named `SERVER_HOME`.

Linux

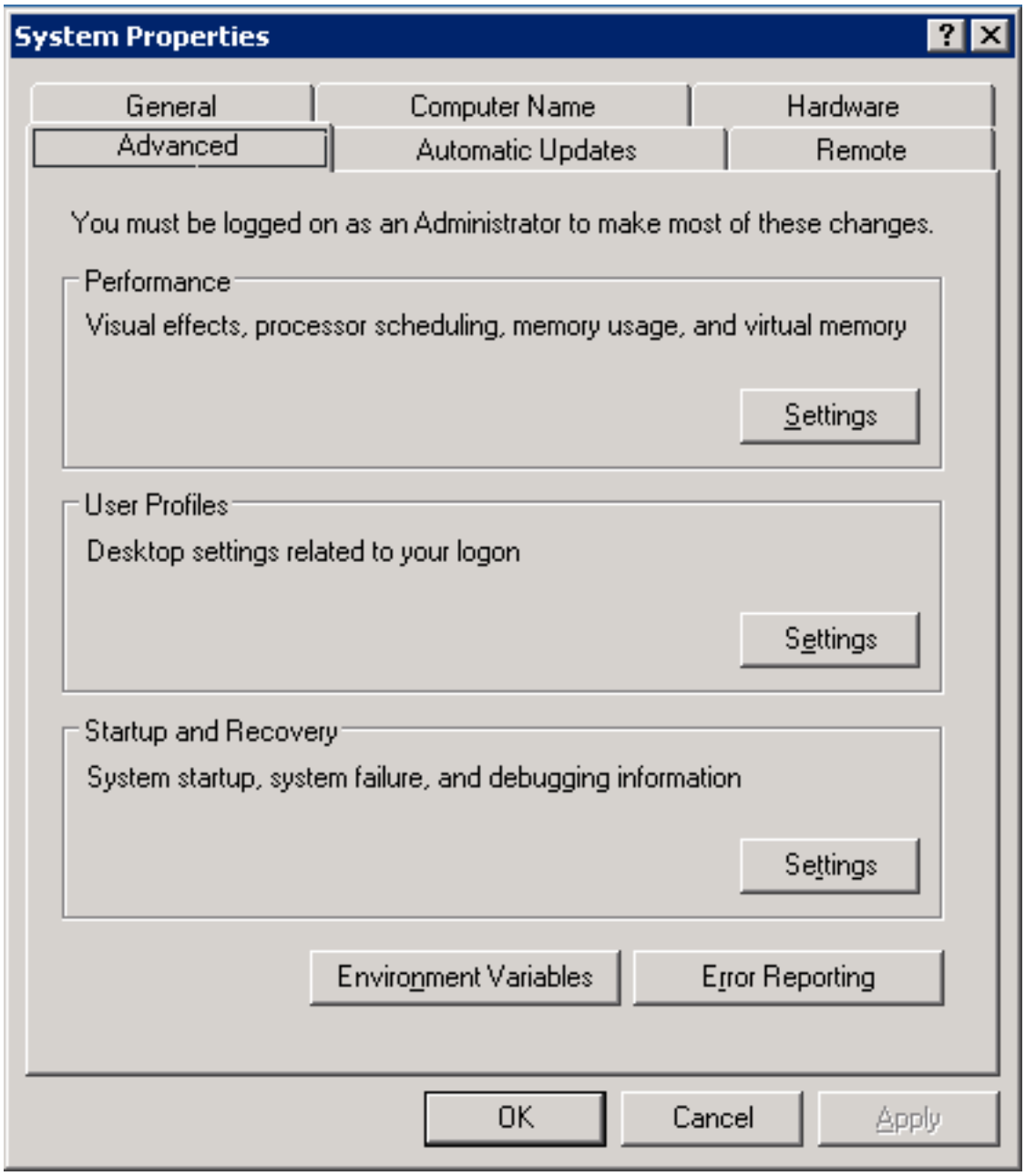
Edit the `.profile` file in your home directory to add the `SERVER_HOME` and `JAVA_HOME` environment variables. For example, if you installed into `/opt`:

```
export SERVER_HOME=/opt/springsource-dm-server-2.0.5.RELEASE/
export JAVA_HOME=/user/java/jdk1.6.0_17
export PATH=$JAVA_HOME/bin:$PATH
```

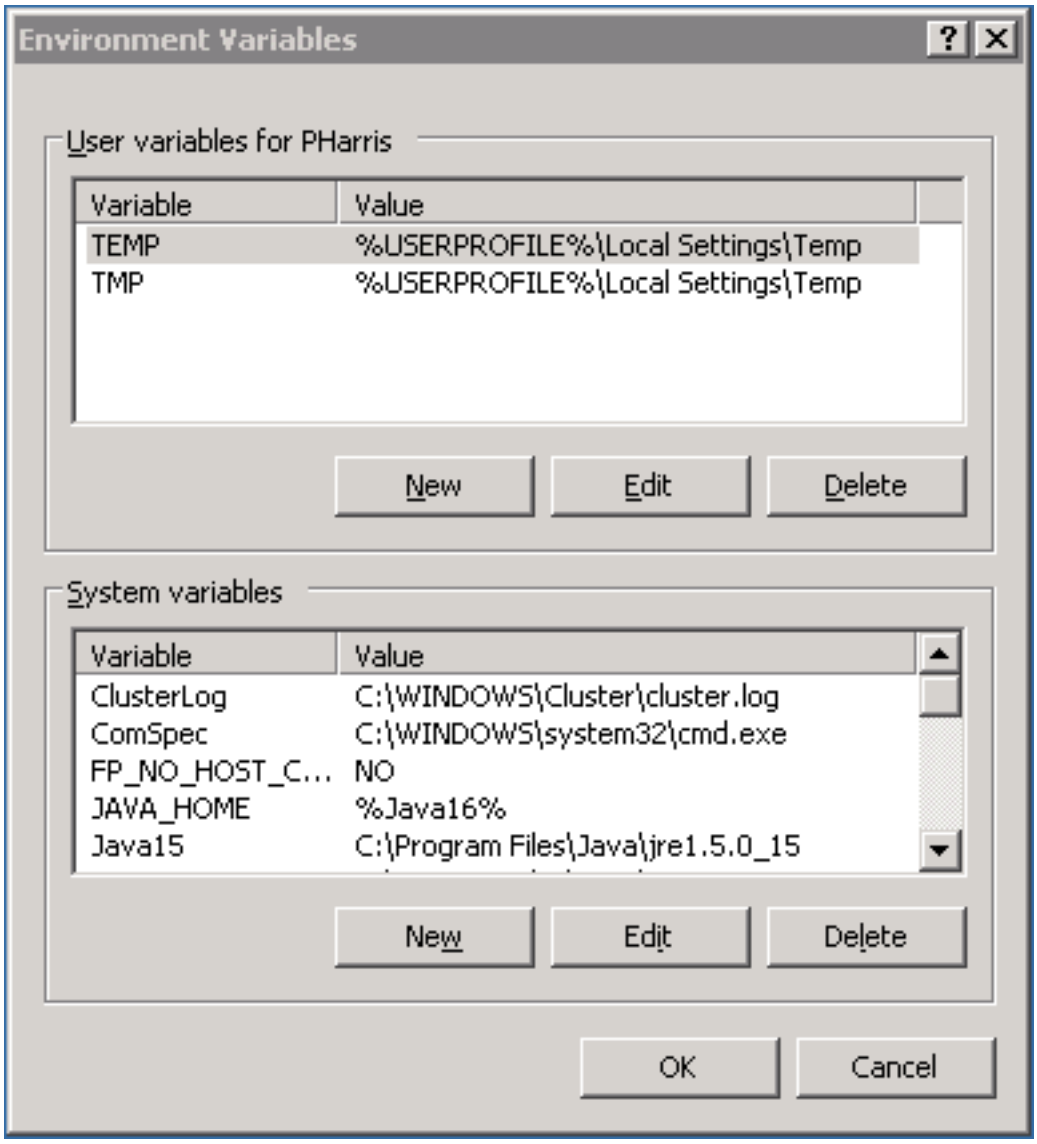
Microsoft Windows

This section shows how to add `SERVER_HOME` as a system variable on Windows. Follow the same procedure to add or update the `JAVA_HOME` environment variable.

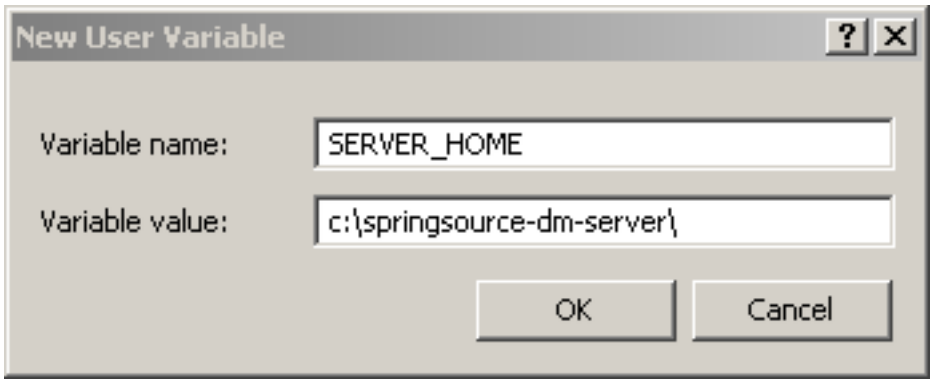
From the Start menu, open the Control Panel and double-click on 'System'.



Click the ‘Advanced’ tab and select ‘Environment Variables’. Next, click the ‘Edit’ button in the ‘System Variables’ section.



This will display the 'Edit System Variable' window. Enter `SERVER_HOME` as the 'Variable name' and the installation directory as the 'Variable value'. Click OK.

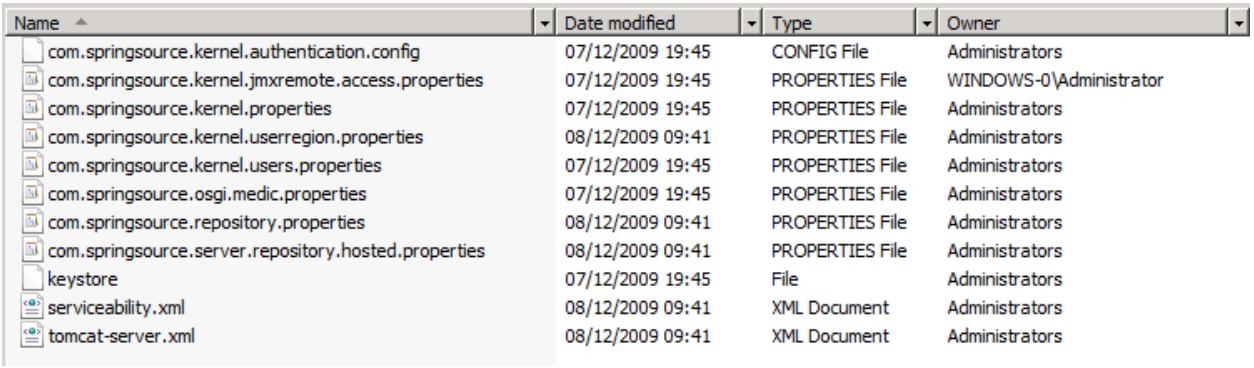


Microsoft Windows - Troubleshooting

When starting the SpringSource dm Server on some variants of Windows you might encounter a problem with file permissions. The error looks like this.

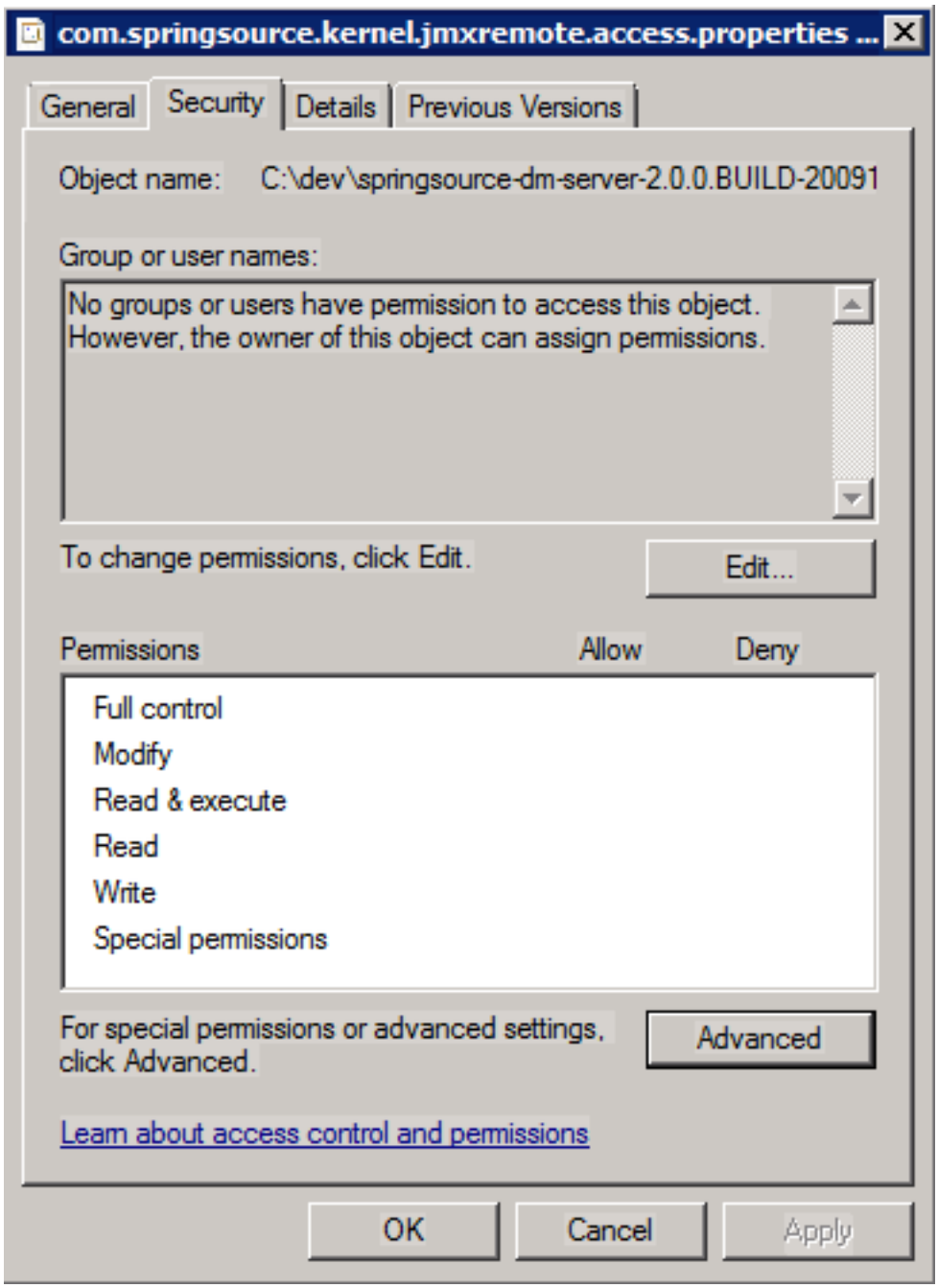
```
C:\dev\springsource-dm-server-2.0.0.BUILD-20091208094124>bin\startup.bat
Error: Exception thrown by the agent : java.io.FileNotFoundException: C:\dev\...\config\
                com.springsource.kernel.jmxremote.access.properties (Access is denied)
```

If the dm Server starts at this point you can skip this section and carry on. Otherwise, go to the 'config' directory of your install in Windows Explorer and you will be able to confirm the problem, an incorrect file ownership.

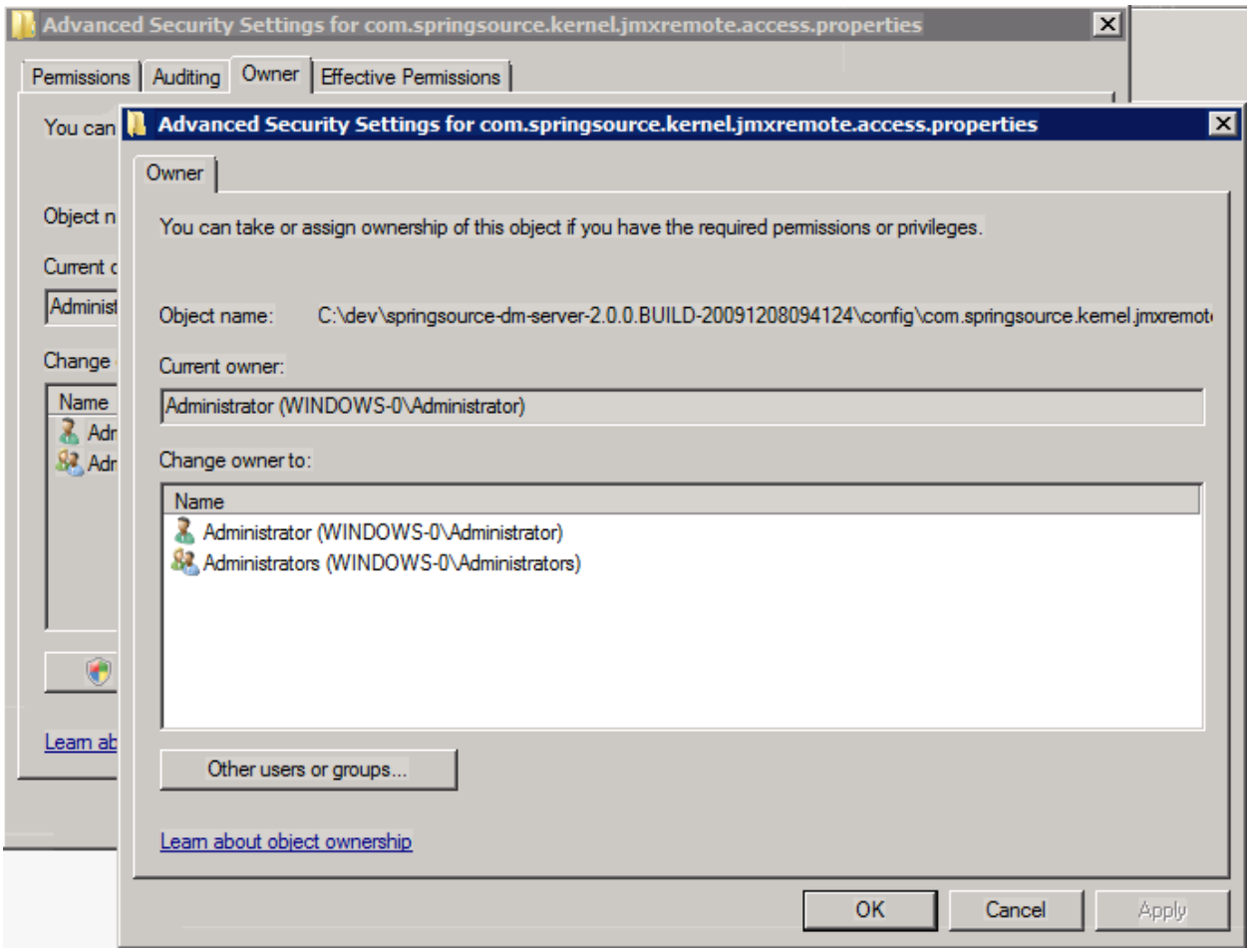


Name	Date modified	Type	Owner
com.springsource.kernel.authentication.config	07/12/2009 19:45	CONFIG File	Administrators
com.springsource.kernel.jmxremote.access.properties	07/12/2009 19:45	PROPERTIES File	WINDOWS-0\Administrator
com.springsource.kernel.properties	07/12/2009 19:45	PROPERTIES File	Administrators
com.springsource.kernel.userregion.properties	08/12/2009 09:41	PROPERTIES File	Administrators
com.springsource.kernel.users.properties	07/12/2009 19:45	PROPERTIES File	Administrators
com.springsource.osgi.medic.properties	07/12/2009 19:45	PROPERTIES File	Administrators
com.springsource.repository.properties	08/12/2009 09:41	PROPERTIES File	Administrators
com.springsource.server.repository.hosted.properties	08/12/2009 09:41	PROPERTIES File	Administrators
keystore	07/12/2009 19:45	File	Administrators
serviceability.xml	08/12/2009 09:41	XML Document	Administrators
tomcat-server.xml	08/12/2009 09:41	XML Document	Administrators

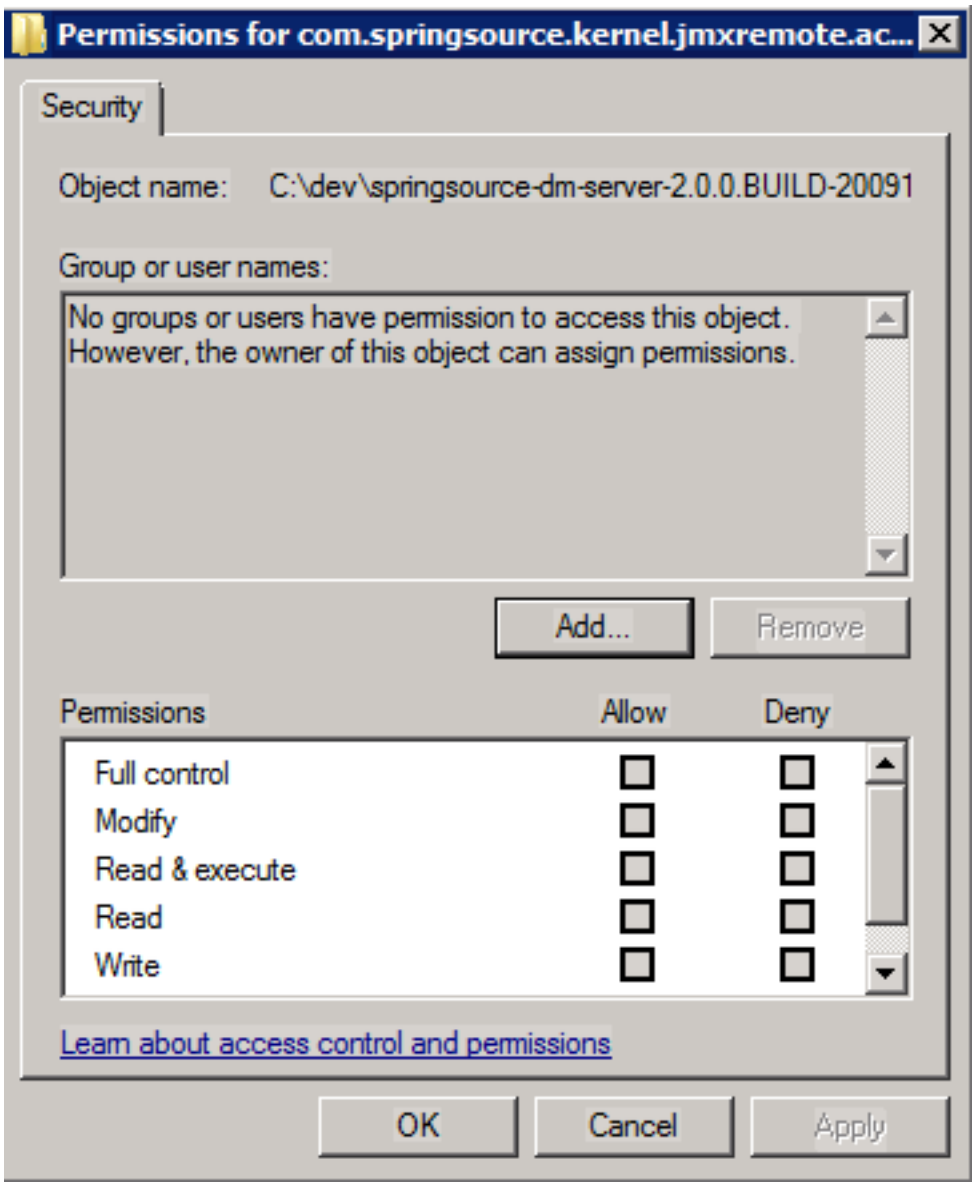
Right click on the 'com.springsource.kernel.jmxremote.access.properties' file and view its properties, then select the 'Security' tab.



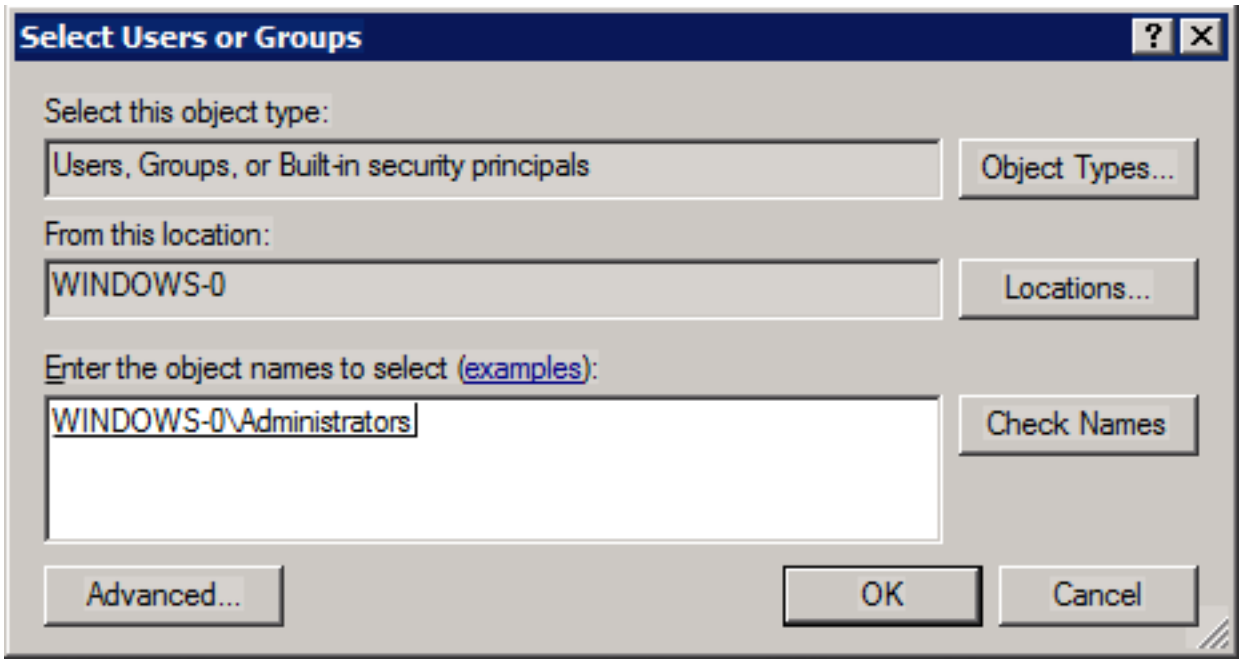
Within the security page select the ‘Advanced’ options, view the owners and then select ‘Edit’. From the list select the owner that you are trying to run the dm Server as.



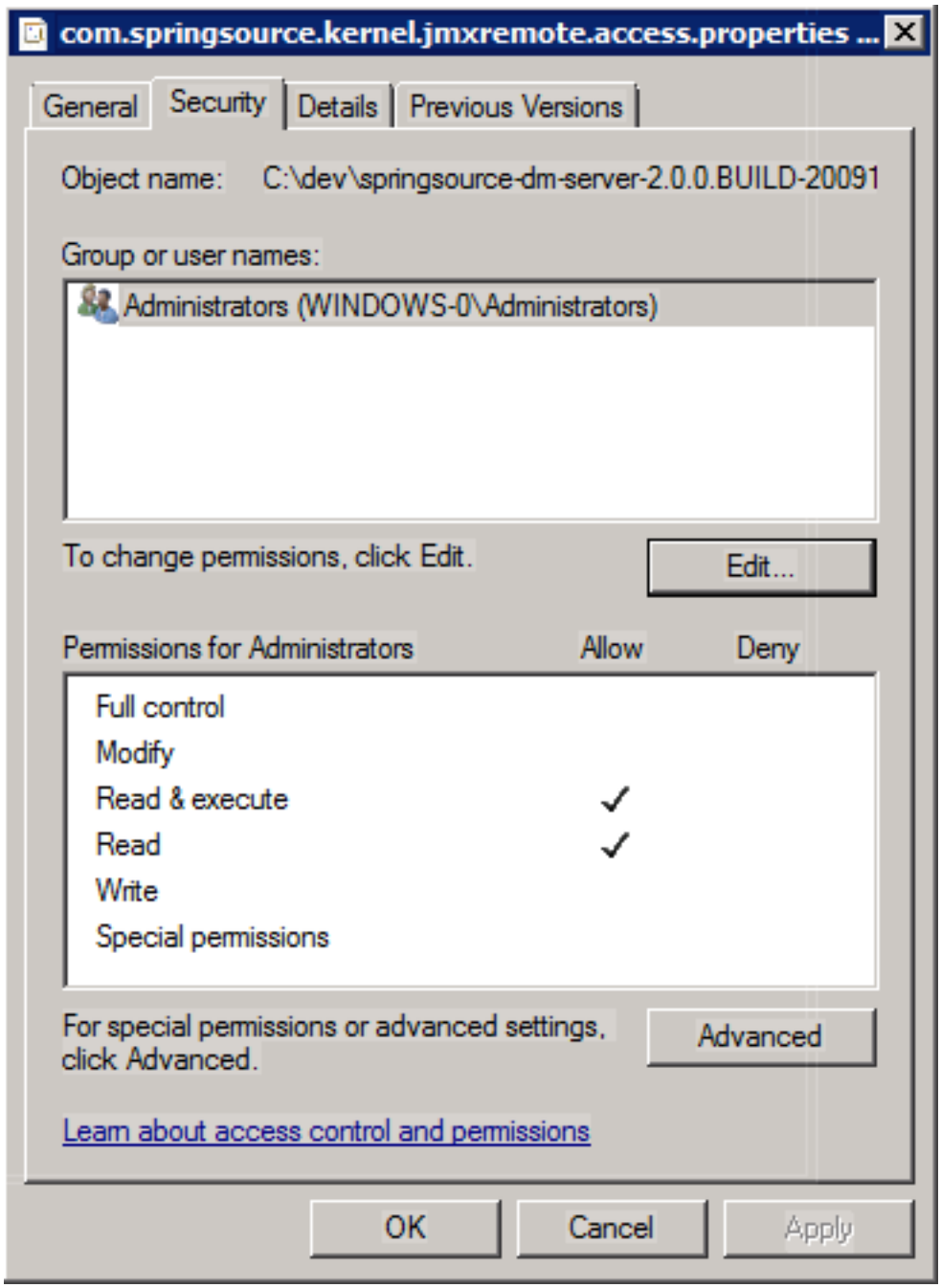
Once this is done select ‘OK’ and then ‘OK’ again to return to the ‘Security’ tab and now select ‘Edit’ on the list of groups and users that have permission to edit the file.



It is likely this list will be empty to start with. Select ‘Add’ and give it the same name you selected from the list before. Clicking ‘Check Names’ will ensure you have entered an acceptable name.



Finally select ‘OK’ and then ‘Apply’ to apply the new setting and you will be back on the security tab and be able to see the new settings.



Once all these steps are complete you can proceed to start the dm Server.

```
C:\dev\springsource-dm-server-2.0.0.BUILD-20091208094124>bin\tstartup.bat
[2009-12-08 13:09:09.545] startup-tracker              <KE0001I> Kernel starting.
```

2. Starting and Stopping SpringSource dm Server

2.1 Starting SpringSource dm Server

To start SpringSource dm Server run the `startup.sh` (Linux) or `startup.bat` (Windows) script. For both platforms, the script is located in the `SERVER_HOME/bin` directory.

Linux

To start SpringSource dm Server, open a terminal window and run `startup.sh`:

```
prompt$ cd $SERVER_HOME
prompt$ bin/startup.sh
```

Once SpringSource dm Server has started, the console will display a log message similar to the one shown below, along with other status messages:

```
[2009-11-30 12:12:12.111] Thread-2   <UR0001I> User region ready.
```

The preceding message indicates that you can start using dm Server.

Microsoft Windows

To start SpringSource dm Server, open a command-window and run `startup.bat`:

```
prompt> cd %SERVER_HOME%
prompt> bin\startup.bat
```

Once SpringSource dm Server has started console will display a log message similar to the one shown below:

```
[2009-11-30 12:12:12.111] Thread-2   <UR0001I> User region ready.
```

The preceding message indicates that you can start using dm Server.

2.2 Starting in Clean Mode

When you start dm Server in clean mode, the startup script removes the `SERVER_HOME/work` directory (and hence all running applications) as well as all trace, log and dump files. It leaves the `SERVER_HOME/repository` and `SERVER_HOME/pickup` directories untouched, which means that any applications previously hot deployed will be automatically reinstalled.

Linux

To start SpringSource dm Server in clean mode, open a terminal window and run `startup.sh -clean`:

```
prompt$ cd $SERVER_HOME
prompt$ bin/startup.sh -clean
```

Microsoft Windows

To start SpringSource dm Server in clean mode, open a command window and run `startup.bat -clean`:

```
prompt> cd %SERVER_HOME%
prompt> bin\startup.bat -clean
```

2.3 Starting in Debug Mode

Linux

To start SpringSource dm Server in debug mode, run `startup.sh` passing in the `-debug` argument:

```
prompt$ cd $SERVER_HOME
prompt$ bin/startup.sh -debug
```

This will start the debug agent listening on port `8000` which is the default remote debug port used by Eclipse. To start in debug mode with a specific port number, pass this in as the value for the `-debug` argument:

```
prompt$ cd $SERVER_HOME
prompt$ bin/startup.sh -debug 8001
```

This will start the debug agent listening on port `8001`. To start in debug mode and suspend the VM until a debugger attaches, pass in the `-suspend` argument along with the `-debug` argument:

```
prompt$ cd $SERVER_HOME
prompt$ bin/startup.sh -debug -suspend
```

This starts the debug agent, but prevents SpringSource dm Server from actually starting until a debugger attaches to the agent. This can be useful when trying to diagnose problems that occur during startup.

Microsoft Windows

To start SpringSource dm Server in debug mode, run `startup.bat` passing in the `-debug`

argument:

```
prompt> cd %SERVER_HOME%
prompt> bin\startup.bat -debug
```

This will start the debug agent listening on port `8000` which is the default remote debug port used by Eclipse. To start in debug mode with a specific port number, pass this in as the value for the `-debug` argument:

```
prompt> cd %SERVER_HOME%
prompt> bin\startup.bat -debug 8001
```

This will start the debug agent listening on port `8001`. To start in debug mode and suspend the VM until a debugger attaches, pass in the `-suspend` argument along with the `-debug` argument:

```
prompt> cd %SERVER_HOME%
prompt> bin\startup.bat -debug -suspend
```

This starts the debug agent, but prevents SpringSource dm Server from actually starting until a debugger attaches to the agent. This can be useful when trying to diagnose problems that occur during startup.

2.4 Starting with JMX Access Modifications

The SpringSource dm Server always starts with JMX access enabled, allowing you to use a management tool such as JConsole to attach to the dm Server instance. By default both local access and remote access over SSL with username and password authentication are provided. The default port for secure JMX access is `9875` and the default username and password are `admin` and `springsource`.

Linux

To start SpringSource dm Server with default JMX access enabled, run `startup.sh` passing in no arguments:

```
prompt$ cd $SERVER_HOME
prompt$ bin/startup.sh
```

To start JConsole, run the `jconsole.sh` script, located in the `bin` directory, as shown:

```
prompt$ cd $SERVER_HOME
prompt$ bin/jconsole.sh
```

The following image shows how to specify a local connection using JConsole.

The following image shows how to specify a remote connection in JConsole that uses SSL with the default username/password (`admin/springsource` and default secure port of `9875`).

To start with the JMX remote access on a specific port number other than the default `9875`, pass this port number in as the value of the `-jmxport` argument:

```
prompt$ cd $SERVER_HOME
prompt$ bin/startup.sh -jmxport 9090
```

This will start the SpringSource dm Server with JMX enabled for remote connections on port `9090`.

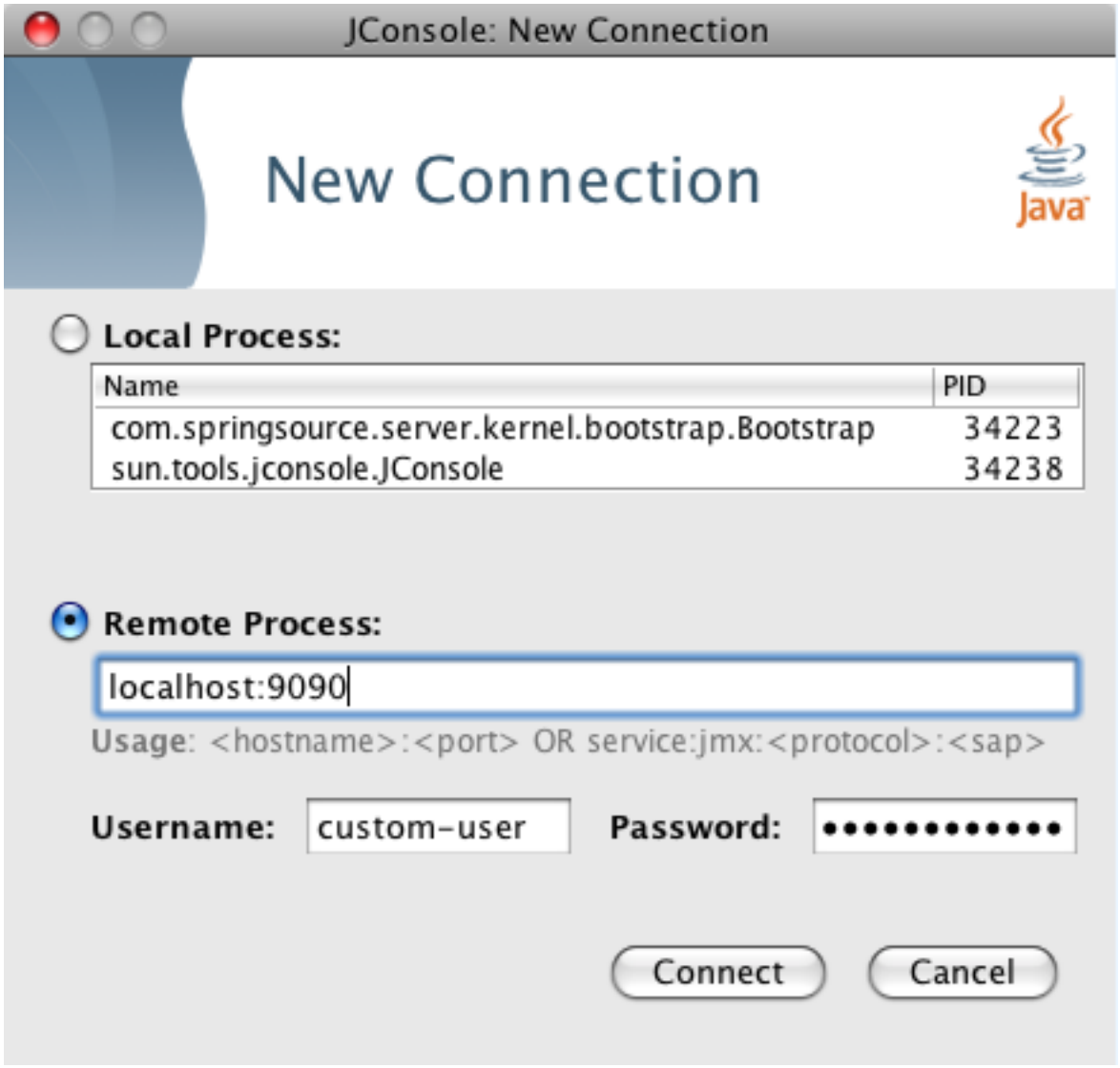
To start the JMX remote access with a custom username and password, update the `$SERVER_HOME/config/com.springsource.kernel.users.properties` file. First specify the custom username by changing the value of the `role.admin` property. Then set the password of this new user by adding a new property called `user.`*`username`*, where *`username`* refers to the actual name of the user. Finally, restart dm Server for the changes to take effect.

For example, if you want change the JMX remote access username to `custom-user` with password `springsource1`, change the file as follows:

```
##################
# User definitions
##################
user.custom-user=springsource1


##################
# Role definitions
##################
role.admin=custom-user
```

Specify the custom username in JConsole as shown.



To start the JMX remote access using a custom SSL certificate, edit the file located at `$SERVER_HOME/config/keystore`. If you wish to use a different keystore, pass this filename in as the value for the `-keystore` argument and the keystore password in as the value for the `-keystorePassword` argument:

```
prompt$ cd $SERVER_HOME
prompt$ bin/startup.sh -keystore customKeystore -keystorePassword customKeystorePassword
```

This will start the SpringSource dm Server with JMX enabled for remote connections using an SSL certificate from `customKeystore` with a password of `customKeystorePassword`.

Microsoft Windows

To start SpringSource dm Server with default JMX access enabled, run `startup.bat` passing in no arguments:

```
prompt> cd %SERVER_HOME%
prompt> bin\startup.bat
```

To start JConsole, run the `jconsole.bat` script, located in the `bin` directory, as shown:

```
prompt> cd %SERVER_HOME%
prompt> bin\jconsole.bat
```

The following image shows how to specify a local connection using JConsole.

The following image shows how to specify a remote connection in JConsole that uses SSL with the default username/password (`admin/springsource` and default secure port of `9875`).

To start with the JMX remote access on a specific port number other than the default `9875`, pass this port number in as the value of the `-jmxport` argument:

```
prompt> cd %SERVER_HOME%
prompt> bin\startup.bat -jmxport 9090
```

This will start the SpringSource dm Server with JMX enabled for remote connections on port `9090`.

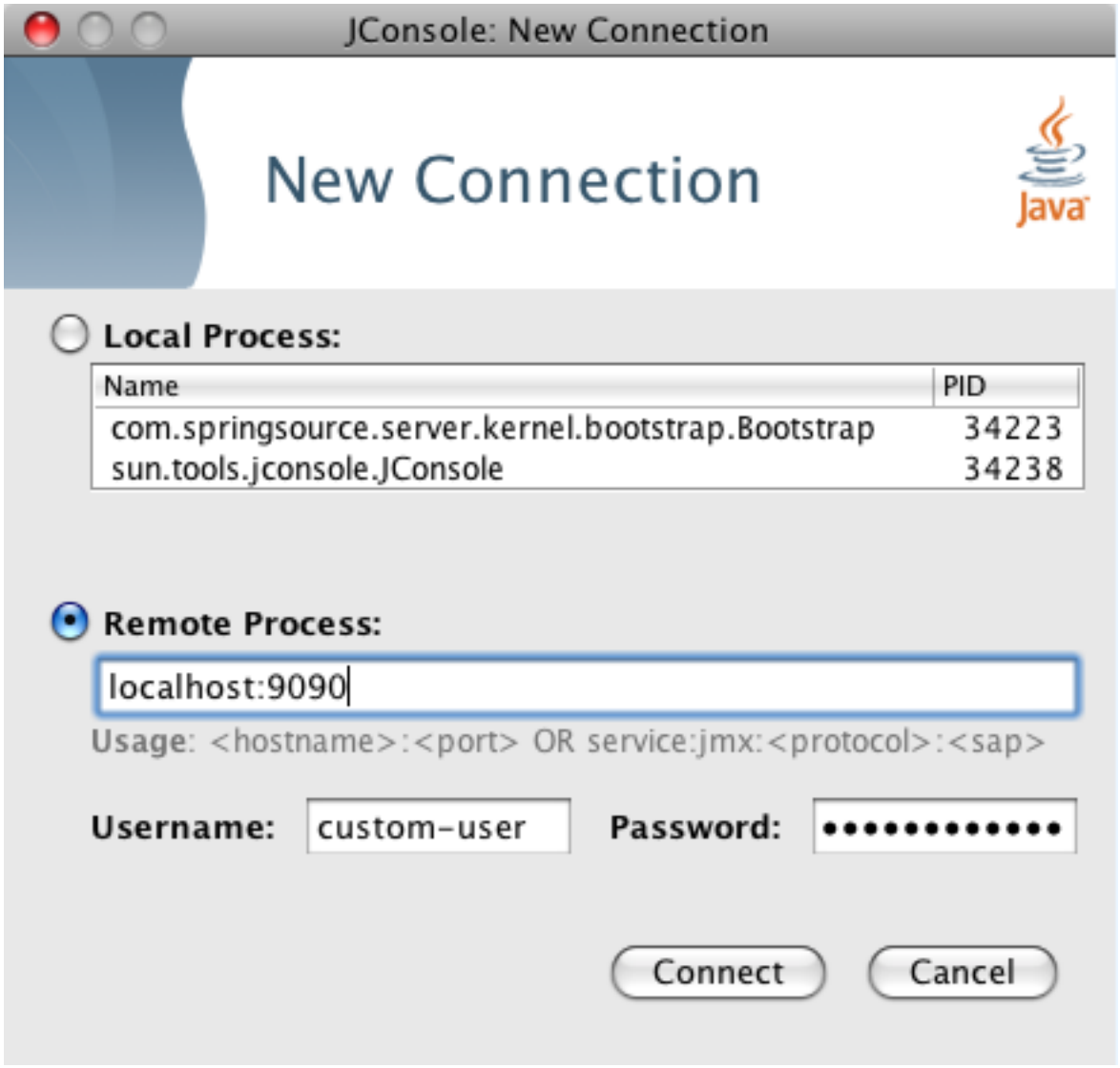
To start the JMX remote access with a custom username and password, update the `%SERVER_HOME%\config\com.springsource.kernel.users.properties` file. First specify the custom username by changing the value of the `role.admin` property. Then set the password of this new user by adding a new property called `user.`*`username`*, where *`username`* refers to the actual name of the user. Finally, restart dm Server for the changes to take effect.

For example, if you want change the JMX remote access username to `custom-user` with password `springsource1`, change the file as follows:

```
##################
# User definitions
##################
user.custom-user=springsource1


##################
# Role definitions
##################
role.admin=custom-user
```

Specify the custom username in JConsole as shown.



To start the JMX remote access using a custom SSL certificate, edit the file located at `%SERVER_HOME%\config\keystore`. If you wish to use a different keystore, pass this filename in as the value for the `-keystore` argument and the keystore password in as the value for the `-keystorePassword` argument:

```
prompt> cd %SERVER_HOME%
prompt> bin\startup.bat -keystore customKeystore -keystorePassword customKeystorePassword
```

This will start the SpringSource dm Server with JMX enabled for remote attach using an SSL certificate from `customKeystore` with a password of `customKeystorePassword`.

2.5 Starting With a Custom Configuration Directory

Use the `-configDir` option to specify an alternate `config` directory, different from the default `SERVER_HOME/config` directory. This option allows you to use the same SpringSource dm Server installation to run multiple instances of dm Server . Simply create a config directory for each instance, specify unique port numbers, logging and tracing directories, and so on. and then specify that directory when starting SpringSource dm Server.

If you specify a relative path for the `-configDir` parameter, the startup script interprets the path as relative to the root of the SpringSource dm Server installation, and not relative to the directory from which you execute the `startup` script.

Linux

To start SpringSource dm Server using a config directory of `/config/node1`:

```
prompt$ cd $SERVER_HOME
prompt$ bin/startup.sh -configDir /config/node1
```

Windows

To start SpringSource dm Server using a config directory of `c:\config\node1`:

```
prompt> cd %SERVER_HOME%
prompt> bin\startup.bat -configDir c:\config\node1
```

2.6 Stopping SpringSource dm Server

Linux

To stop a running instance of SpringSource dm Server, start a new terminal window and the run `shutdown.sh` script:

```
prompt$ cd $SERVER_HOME
prompt$ bin/shutdown.sh
```

To stop a running instance of SpringSource dm Server immediately, bypassing normal shutdown processing, run `shutdown.sh` with the `-immediate` option:

```
prompt$ cd $SERVER_HOME
prompt$ bin/shutdown.sh -immediate
```

If, when you started the dm Server instance, you used the `-jmxport` option to specify a non-default JMX port number, then you must pass this port number to the `-jmxport` of the `shutdown.sh` script to gracefully shut it down. For example, if you specified `9090` as the JMX port, use the following to shut down the dm Server instance:

```
prompt$ cd $SERVER_HOME
prompt$ bin/shutdown.sh -jmxport 9090
```

Microsoft Windows

To stop a running instance of SpringSource dm Server, start a new console window and run the `shutdown.bat` script:

```
prompt> cd %SERVER_HOME%
prompt> bin\shutdown.bat
```

To stop a running instance of SpringSource dm Server immediately, bypassing normal shutdown processing, run `shutdown.bat` with the `-immediate` option:

```
prompt> cd %SERVER_HOME%
prompt> bin\shutdown.bat -immediate
```

If, when you started the dm Server instance, you used the `-jmxport` option to specify a non-default JMX port number, then you must pass this port number to the `-jmxport` of the `shutdown.bat` script to gracefully shut it down. For example, if you specified `9090` as the JMX port, use the following to shut down the dm Server instance:

```
prompt> cd %SERVER_HOME%
prompt> bin\shutdown.bat -jmxport 9090
```

2.7 Starting SpringSource dm Server When the Operating System Starts

If you need SpringSource dm Server to start automatically when the operating system starts, you should run SpringSource dm Server as a Windows service or a UNIX background process. You can do this by using a service wrapper script provided with SpringSource dm Server.

The `SERVER_HOME/bin/service` directory contains a service wrapper script for each supported operating system. Before running the appropriate script, you must either set the `SERVER_HOME` environment variable to point to the SpringSource dm Server installation directory or edit the file `SERVER_HOME/bin/service/conf/wrapper.conf`.

If you run the service wrapper script with no option, it will display the available options. The most useful options are described in the following table.

Table 2.1. Service Wrapper Options

Option	Description
`console`	Starts SpringSource dm Server in the foreground. Useful for validating that the service wrapper is configured correctly.
`install`	*Windows only.* Installs SpringSource dm Server as a Windows service.
`start`	Starts SpringSource dm Server in the

Option	Description
	background. On UNIX operating systems, you may call the wrapper script with this option during initialization, for example as part of `init.d` processing. SpringSource dm Server console output appears in `SERVER_HOME/wrapper.log`.
`stop`	Stops the SpringSource dm Server background process. On UNIX operating systems, you may call the wrapper script with this option during termination, for example as part of `init.d` processing.
`remove`	*Windows only*. Removes SpringSource dm Server as a Windows service.

3. Overview of the dm Server Kernel and User Region

Conceptually, dm Server can be divided into two separate subsystems, one of which actually encompasses the other:

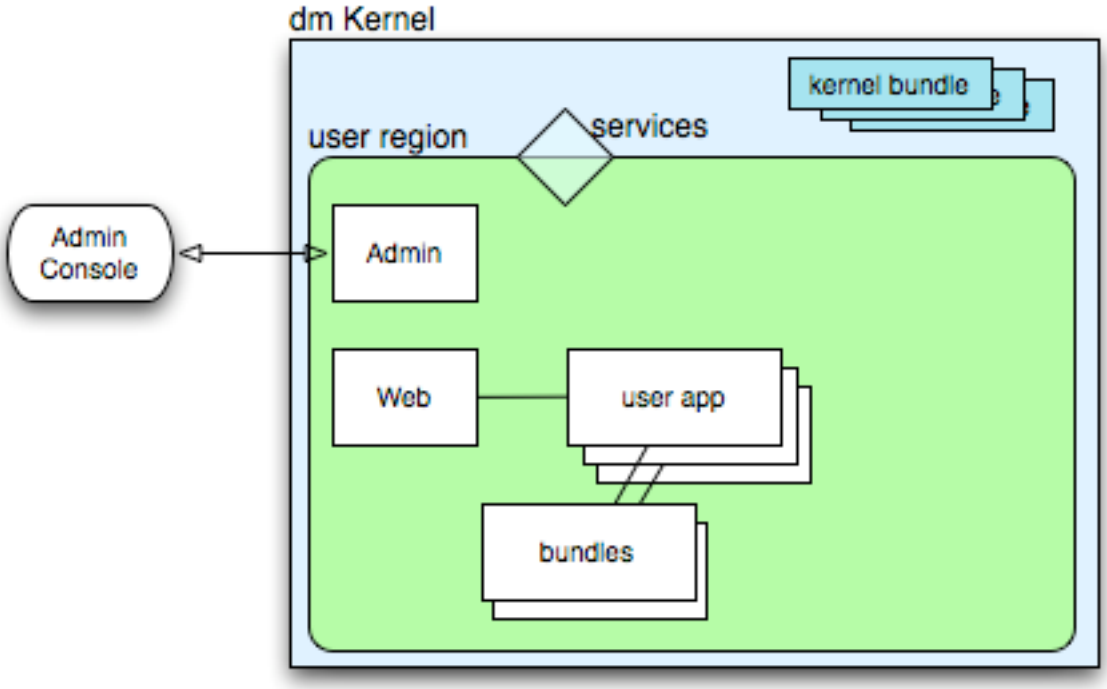
- The *kernel*, which is the heart of dm Server. It makes up most of the dm Server, except for the part that supports Web applications. In other words, the kernel provides full OSGi modular support for your applications, as long as they are not Web-based.

  See The dm Server Kernel for additional information.

- The *user region* is the subsystem that manages user applications. It deliberately isolates the kernel from both your applications and those of the dm Server itself, such as the Admin Console, making it much easier for you to administer dm Server.

  See The dm Server User Region for additional information.

The following graphic shows how the kernel and user region make up dm Server:



When you download and install SpringSource dm Server you get both the kernel and a user region. You can also download and use the kernel on its own if you do not plan on deploying Web applications.

3.1 The dm Server Kernel

The dm Kernel encapsulates almost all of dm Server except for the deployment of Web applications. In sum, the kernel provides the following dm Server features:

- Deployment of non-Web artifacts, such as OSGi bundles, PARs, plans, and configuration artifacts.
- Local and hosted repositories
- Scoping
- Hot deployment
- User region
- Auto-provisioning
- System and application tracing and dump support
- Spring beans and Spring DM support

See Configuring dm Server for details about configuring the default kernel to better suit your environment.

3.2 The dm Server User Region

The user region isolates the kernel from deployed applications, including both your own user applications and the user-oriented dm Server applications such as the Admin Console. This means that the kernel is mostly invisible to applications and to application management. This is because most of the kernel bundles are not installed in the user region (apart from a few needed for region management). The necessary function to support the kernel runs in the OSGi framework, but the user region applications cannot see it, except for the services that are normally offered.

This way of implementing dm Server greatly simplifies the administration tasks you perform. In particular, when you use the dm Shell or the Admin Console to manage dm Server, you do not see the many bundles that are internal to the kernel. The only exceptions are the kernel bundles that dm Server uses for region management, which are required to be installed in the user region.

This isolation has many other benefits. For example, it is no longer necessary for the kernel and user applications to use the same version of the Spring Framework. In fact the kernel installs only those parts of the Spring Framework that it needs. If you update the kernel, it is far less likely that you will also need to upgrade or adjust the applications to accomodate a new version of the kernel. The kernel implementation is therefore much more stable and resilient and applications are much more likely to survive kernel upgrades between releases.

When you install dm Server, the kernel creates a single user region. The configuration of the user region and the kernel is completely separate; see Configuring dm Server for details.

Finally, the isolation provided by the user region together with scoped applications and plans solve common dependency problems that occur when using OSGi.

4. The dm Shell

The dm Shell is a command line utility that allows you to examine artifacts currently installed to a particular dm Server instance, manage the lifecycle of the installed artifacts, install new artifacts, and shutdown the server. You can install, examine, and manage the lifecycle of the following artifacts:

- Bundles
- Configuration Artifacts
- Exported packages
- PARs
- Plans
- Services installed in the OSGi registry

You can run the dm Shell locally or remotely by using `ssh`. The dm Shell is similar to the Equinox shell, but with added features such as tab completion and command history. Tab completion allows you to enter a partial command, and then press the Tab key and let the dm Shell show you a list of possible commands. To get a command history, simply use the up and down arrows.

4.1 Using the dm Shell

You invoke the dm Shell locally by passing the `-shell` flag to the `startup.sh` (Unix) or `startup.bat` (Windows) command that starts up dm Server. For example, on Unix:

```
prompt$ SERVER_HOME/bin/startup.sh -shell
```

You will see status messages for the dm Kernel starting, and then the dm Shell commmand line utility takes over. You can still view the console log messages by looking at the `$SERVER_HOME/serviceability/eventlogs/event.log` file. After you get the dm Shell splash screen, you can enter commands at the `:>` prompt. Enter `help` for a list of all available commands. For example:

```
prompt$ cd $SERVER_HOME/bin
prompt$ ./startup.sh -shell

[2009-11-06 14:27:26.620] startup-tracker              <KE0001I> Kernel starting.
[2009-11-06 14:27:31.873] startup-tracker              <KE0002I> Kernel started.
[2009-11-06 14:27:32.155] system-artifacts             <DE0056I> Installing plan 'com.springsource.server.web' version '
[2009-11-06 14:27:32.659] Thread-2                     <SH0001I> dm Kernel ssh shell available on port 2401.

    @@@ ***
   @@@ *****          .__.                .__.          ____   .__.   .__.
  @@@@ ******         __|  |  _____   _____.|  |__   ____ \   \  |  |  |  |
  @@@@@@ ****       / __  | /     \  /  ___/|  |  \ /  __ \ |  |  |  |  |
   @@@@@ ***       / /_/  ||  Y Y  \ \___ \ |   Y  \\  ___/ |  |__|  |__
    @@@ ***        \_____||__|_|__/ /_____/ |___|__/ \____/ |____/|____/

Type 'help' to see the available commands.
:> help
```

```
    bundle - Management and examination of bundle artifacts
    config - Management and examination of configuration artifacts
    help
    install - Install (deploy) an artifact to the server
    package - Management and examination of exported packages
    par - Management and examination of PAR artifacts
    plan - Management and examination of plan artifacts
    service - Examination of services
    shutdown
:>
```

To invoke dm Shell remotely, use the `ssh` command with the following parameters:

- Use the `-p` *`shell.port`* option to pass the port number where the dm Shell listens; the default value is `2401`. You can change this port by editing the value of the `shell.port` property in the `$SERVER_HOME/conf/com.springsource.kernel.properties` configuration file.
- Use the *`user@host`* parameter to specify the administration user; you will be prompted for a password. The default administration user/password is `admin/springsource`. You can change the administration user by updating the `role.admin` value in the `$SERVER_HOME/conf/com.springsource.kernel.users.properties` file. Use the `user.admin` to change the administration user's password.

The following example shows how to invoke the dm Shell using `ssh` on the local computer using the default port, user, and password:

```
prompt$ ssh -p 2401 admin@localhost
admin@localhost's password:

    @@@ ***
   @@@ *****          .__.                  .__.          .__.  .__.
  @@@@ ******         |__|  ______   ______ |  |__   ____ |  |  |  |
  @@@@@@ ****        /  __  |  /  \  /  ___/ |  |  \ /  __ \|  |  |  |
   @@@@@ ***       / /_/  ||  Y Y  \ \___ \  |   Y  \\  ___/|  |__|  |__
    @@@ ***        \_____||__|_|__/ /_____/ |___|__/ \_____/|____/|____/

Type 'help' to see the available commands.
:>
```

Available dm Shell Commands

The following table lists the dm Shell commands; each command in turn has a variety of options that you can specify, depending on what you want to do, such as start a bundle or refresh a plan. The reference documentation about each command provides the full list of available options.

Table 4.1. dm Shell Commands

Command	Description
exit	Exit from the dm Shell session.
bundle	Used to manage and display information about bundle artifacts.
config	Used to manage and display information about configuration artifacts.

Command	Description
package	Used to manage and display information about exported packages.
par	Used to manage and display information about PAR artifacts.
plan	Used to manage and display information about plan artifacts.
service	Displays information about services in the OSGi registry.
install	Used to install an artifact to dm Server.
shutdown	Shuts down the dm Server instance to which the dm Shell is connected.
help	Display help about the list of available commands, as well as more detailed help about individual commands.

4.2 dm Shell Command Reference

This section contains reference information about the following dm Shell commands:

- exit
- bundle
- config
- package
- par
- plan
- service
- install
- shutdown
- help

exit Command

Use the `exit` command to exit from the dm Shell session.

The `exit` command does not have any options.

bundle Command

Use the `bundle` command to manage the lifecycle of bundles deployed to @dms@ and to gather information about deployed bundles, such as diagnostic information, header information, and so on.

The following table lists the options you can specify for this command.

Table 4.2. Options of the bundle Command

Option	Descrption
list	Displays the list of bundles that are currently installed to the current dm Server instance. With the exception of a few kernel bundles and their services, which dm Server uses to administer the user region, none of the kernel is visible to user installed artifacts; rather, only the bundles installed in the user region are visible. Each bundle is identified by an internal `ID` which you can then use with the other `bundle` commands that manage a particular bundle, such as `start` *`id`*. The `list` command also displays the version of the bundle, along with its state, which is one of the following standard OSGi lifecycle states: • **Installed**: The bundle is installed but its dependencies have not yet been resolved. • **Resolved**: The bundle is resolved and you can now start it. • **Uninstalled**: The bundle is uninstalled and you can not use it. • **Starting**: The bundle is in the process of starting. • **Active**: The bundle is running and you can now use it.

Option	Descrption
	• **Stopping**: The bundle is in the process of stopping. Use one of the other `bundle` command to change the state of a bundle. For example, use the `bundle start` *`id`* command to change the state of a bundle from `Installed` to `Active`.
examine *id*	Displays detailed information about the specified bundle. Use the `bundle list` command to get the internal id of a particular bundle. In addition to the information provided by the `bundle list` command (id, full name, version, and state), the `examine` command specifies whether the bundle includes a Spring application context (or is *Spring Powered*) and the exact physical location of the bundle JAR file. The `examine` also provides the full list of packages that the bundle imports, as well as the bundles that in turn export these imported packages. Finally, the command displays the packages that the current bundle exports, and then in turn the list of other installed bundles that are currently importing these exported packages.
start *id*	Starts the specified bundle. Use the `bundle list` command to get the internal id of a particular bundle. To start a bundle, it must have already been resolved by dm Server, or in other words, be in the OSGi `Resolved` state. After dm Server successfully starts the bundle, it is listed in the `Active` state.
stop *id*	Stops the specified bundle. Use the `bundle list` command to get the internal id of a particular bundle. When you stop a bundle, it goes from the OSGi `Active` state to the `Resolved` state, and you must re-start it if you want to use the

Option	Descrption
	application that the bundle contains.
refresh *id*	Updates the contents of the specified bundle. Use the `bundle list` command to get the internal id of a particular bundle. Use this command if you have changed the contents of the bundle JAR file and you want to refresh the artifact as installed in the OSGi framework.
uninstall *id*	Uninstalls the specified bundle from dm Server. Use the `bundle list` command to get the internal id of a particular bundle. When the uninstall process is complete, the bundle does not show up in the list of bundles displayed by the `bundle list` command. If you want to use the application in the bundle, you must re-install it using the `install` command.
diag *id*	Provides diagnostic information about the specified bundle. In particular, this command displays information about the imported packages that dm Server could not resolve. Use the `bundle list` command to get the internal id of a particular bundle.
headers *id*	Displays the complete list of manifest headers of the specified bundle. Use the `bundle list` command to get the internal id of a particular bundle. The manifest headers include: `Import-Package`, `Module-Type`, `Bundle-SymbolicName`, and so on.

The following examples show how to use this command.

First, use the `bundle list` command to view all the installed bundles; note the last one that is in a `Resolved` state (and many entries have been removed for simplicity):

```
:> bundle list

Id   Name                                    Version               State

0    org.eclipse.osgi                        3.5.1.R35x_v20091005  ACTIVE
1    com.springsource.region.user            0.0.0                 ACTIVE
2    org.springframework.aop                 3.0.0.RC1             RESOLVED
3    org.springframework.asm                 3.0.0.RC1             RESOLVED

(entries removed...)
```

```
60   swf-booking-mvc.war                    0.0.0                  RESOLVED
```

Then use the `bundle start` to start the `swf-booking-mvc.war` bundle:

```
:> bundle start 60

bundle swf-booking-mvc.war:0.0.0 started successfully
```

The following example shows how to view the headers of the `swf-booking-mvc.war` bundle (only the first few lines are shown):

```
:> bundle headers 60

Manifest-Version: 1.0
Ant-Version: Apache Ant 1.7.0
Implementation-Title: swf-booking-mvc
Implementation-Version: 2.0.3.RELEASE
....
```

config Command

Use the `config` command to view and manage the configuration artifacts that have been installed to dm Server. A *configuration artifact* is simply a properties file that is associated with a user application that is contained in a bundle. Using configuration artifacts, you can manage the configuration of a user application completely separately from the bundle that contains the application.

The following table lists the options you can specify for this command.

Table 4.3. Options of the config Command

Option	**Descrption**
list	Lists the configuration artifacts that are currently installed in dm Server. The `list` command displays the full name of each installed configuration artifact, its version, and its current state. Configuration artifacts have similar lifecycles to other OSGi artifacts, such as bundles, and so the list of states in which a configuration can be in is the same as those of bundles; see the bundle command for the list of possible states.
examine *name [version]*	Displays information about the specified configuration artifact. Although you must specify the name of the configuration artifact, its version is optional unless you have multiple versions of the configuration artifact installed. Use the `config list` command to view all configuration artifacts and versions currently installed in dm Server.

Option	Descrption
	A configuration artifact must be active for you to examine it; if it is not currently active, use `config start` to start it and thus change its state to `Active`. The command first displays the factory pid of the configuration artifact as well as the complete location of the bundle to which the configuration artifact is associated. The command then lists all the properties that make up the configuration, as well as their current value.
start *name [version]*	Starts the specified configuration artifact and makes it visible to the internal configuration sub-system of dm Server. Although you must specify the name of the configuration artifact, its version is optional unless you have multiple versions of the configuration artifact installed. Use the `config list` command to view all configuration artifacts and versions currently installed in dm Server. Starting the configuration sets its state to `Active`.
stop *name [version]*	Stops the specified configuration artifact and makes it invisible to the internal configuration sub-system of dm Server. Although you must specify the name of the configuration artifact, its version is optional unless you have multiple versions of the configuration artifact installed. Use the `config list` command to view all configuration artifacts and versions currently installed in dm Server. Stopping the configuration sets its state to `Resolved`.
refresh *name [version]*	Updates the contents of the specified configuration artifact to the internal configuration sub-system of dm Server. Although you must specify the name of the configuration artifact, its version is optional unless you have multiple versions of the configuration artifact installed. Use the `config list` command to view all configuration artifacts and versions currently

Option	Descrption
	installed in dm Server. Use this command if you have changed the contents of the configuration artifact, and you want to make this information known to dm Server and the associated bundle.
uninstall *name [version]*	Uninstalls the specified configuration artifact and make it completely unavailable to dm Server. Although you must specify the name of the configuration artifact, its version is optional unless you have multiple versions of the configuration artifact installed. Use the `config list` command to view all configuration artifacts and versions currently installed in dm Server. Stopping the configuration removes it from dm Server's list of deployed artifacts and it will not show up when you perform a `config list`.

The following example shows how to use this command to list the installed configuration artifacts.

```
:> config list

Name                                    Version                         State
com.springsource.kernel                 0.0.0                          ACTIVE
com.springsource.kernel.jmxremote.access  0.0.0                        ACTIVE
com.springsource.kernel.region          0.0.0                          ACTIVE
com.springsource.kernel.users           0.0.0                          ACTIVE
com.springsource.osgi.medic             0.0.0                          ACTIVE
com.springsource.repository             0.0.0                          ACTIVE
com.springsource.server.repository.hosted 0.0.0                        ACTIVE
```

The next example shows how to refresh the configuration artifact `com.springsource.osgi.medic`:

```
:> config refresh com.springsource.osgi.medic

configuration com.springsource.osgi.medic:0.0.0 refreshed successfully
```

Finally, to view the properties of a configuration artifact, and their current values, use `config examine`:

```
:> config examine com.springsource.osgi.medic

Factory pid:
Bundle Location: file:lib/kernel/com.springsource.kernel.shell-2.0.0.D-20091107203259.jar

Properties:
    dump.root.directory:
        serviceability/dump
    log.dump.bufferSize:
        10000
    log.dump.level:
        DEBUG
    log.dump.pattern:
```

```
        [%d{yyyy-MM-dd HH:mm:ss.SSS}] %-28.28thread %-64.64logger{64} %X{medic.eventCode} %msg %ex%n
    log.wrapSysErr:
        true
    log.wrapSysOut:
        true
    service.pid:
        com.springsource.osgi.medic
```

package Command

Use the `package` command to view the complete list of packages exported by all bundles installed to dm Server, as well as examine a particular exported package in more detail.

The following table lists the options you can specify for this command.

Table 4.4. Options of the package Command

Option	Descrption
list	Displays all the exported packages for all bundles in dm Server. In addition to the package name, the command displays the version of the exported package and the `id` of the bundle that contains the exported package. You can examine the bundle by using the command `bundle examine` *`id`*.
examine *name version*	Displays details about the exported bundle. You must specify both the name of the exported package and its version; use `package list` to view the exact names and version. This command provides the following additional information about the exported package: • The name and version of the bundle that exports the package. This means that the package name is explicitly listed in the bundle's `MANIFEST.MF` file as part of the `Export-Package` header. • Any attributes that are part of the `Export-Package`, in addition to `version`. • The directives that are part of the `Export-Package` header. A typical directive is `uses`, which declares up-front constraints on a number of other packages.

Option	Descrption
	• The list of all bundles that import the package.

The following example shows how to list all the exported packages for all bundles installed on dm Server:

```
:> package list

Name                                        Version                   Providing Bundle

com.springsource.kernel.agent.dm            2.0.0.D-20091110110152    6
com.springsource.kernel.artifact.bundle     2.0.0                     1
com.springsource.kernel.artifact.library    2.0.0                     1
com.springsource.kernel.core                2.0.0.D-20091110110152    1
com.springsource.kernel.deployer.app.spring 2.0.0.D-20091110110152    7
com.springsource.kernel.deployer.core       2.0.0.D-20091110110152    1
com.springsource.kernel.deployer.core.event 2.0.0.D-20091110110152    1
com.springsource.kernel.dmfragment          2.0.0.D-20091110110152    16
com.springsource.kernel.install.artifact    2.0.0.D-20091110110152    1
com.springsource.kernel.install.environment 2.0.0.D-20091110110152    1
...
[data removed for clarity]
```

The following example shows how to examine a particular exported package:

```
:> package examine org.springframework.web.servlet 3.0.0.RC1

Providing Bundle: org.springframework.web.servlet 3.0.0.RC1 [55]

Attributes:

Directives:
    uses:
        javax.servlet, javax.servlet.http, org.springframework.beans,
        org.springframework.context, org.springframework.context.event,
        org.springframework.context.i18n, org.springframework.ui,
        org.springframework.ui.context, org.springframework.util,
        org.springframework.web.context, org.springframework.web.multipart
    x-equinox-ee:
        -1
    x-internal:
        false

Imported By:
    com.springsource.server.splash 2.0.0.D-20091110115520 [59] at [3.0.0, 3.1.0)
    com.springsource.server.repository.hosted-2.0.0.D-20091109225604-com.springsource.server.repository.hosted.web 2.0.0
    org.springframework.js 2.0.8.RELEASE [54] at [2.5.6.SEC01, 3.1.0)
    com.springsource.server.admin.web 2.0.0.D-20091110115520 [50] at [3.0.0, 3.1.0)
```

par Command

Use the `par` command to view all the PARs currently installed in dm Server, view details about a particular PAR and manage its lifecycle, such as starting, stopping, refreshing, and uninstalling it.

The following table lists the options you can specify for this command.

Table 4.5. Options of the par Command

Option	Descrption
list	Displays all the PARs that are currently installed in dm Server.

Option	Descrption
	The `list` command displays the full name of each installed PAR, its version, and its current state. PARs have similar lifecycles to other OSGi artifacts, such as bundles, and so the list of states in which a PAR can be in is the same as those of bundles; see the bundle command for the list of possible states.
examine *name version*	Displays information about the specified PAR; you are required to identify the PAR with both its name and its version. Use the `par list` command to view all installed PAR files and their versions. The command displays the following information: • The current state of the PAR (see the bundle command for the full list of possible states). • Whether the PAR is *scoped*. Scoping specifies whether dm Server should deploy the members of the PAR in their own scope; when scoping is disabled, dm Server deploys the artifacts into the global scope and they are accessible for access by all other artifacts. • Whether the PAR is *atomic*. When a PAR is atomic, dm Server manages the lifecycle of all its member artifacts as a single entity, which means if one artifact member is started, then dm Server starts *all* the PAR artifacts. If one artifact fails to start, then dm Server stops all other artifacts in the PAR. • The individual members, or children, of the PAR. These could be other PARs, plans, bundles, configuration artifacts, and so on.
start *name version*	Starts the specified PAR. You must specify both the full name of the PAR as well as the version you want to start. Use the `par list` command to get the list of PARs currently installed in dm Server. To start a PAR, it must have already been resolved by dm Server, or in other words, be

Option	Descrption
	in the OSGi `Resolved` state. After dm Server successfully starts the PAR, it is listed in the `Active` state.
stop *name version*	Stops the specified PAR. You must specify both the full name of the PAR as well as the version you want to stop. Use the `par list` command to get the list of PARs currently installed in dm Server. When you stop a PAR, it goes from the OSGi `Active` state to the `Resolved` state, and you must re-start it if you want to use the application that the PAR contains.
refresh *name version*	Updates the contents of the specified PAR. You must specify both the name and version of the PAR you want to refresh. Use the `par list` command to this information. Use this command if you have changed the contents of the PAR file and you want to refresh the artifact as installed in the OSGi framework.
uninstall *name version*	Uninstalls the specified PAR. You must specify both the name and version of the PAR you want to refresh. Use the `par list` command to this information. When the uninstall process is complete, the PAR will not show up in the list of PARs displayed by the `par list` command. If you want to use the application in the PAR, you must re-install it using the `install` command.

The following example shows how to list the PARs that have been installed in dm Server:

```
:> par list
Name                                        Version                    State
com.springsource.server.repository.hosted    2.0.0.D-20091109225604          ACTIVE
```

The following example shows how to examine a particular PAR file:

```
:> par examine com.springsource.server.repository.hosted 2.0.0.D-20091109225604

State:  ACTIVE
Scoped: true
```

```
Atomic: true

Children:
    bundle com.springsource.server.repository.hosted.core 2.0.0.D-20091109225604
    bundle com.springsource.server.repository.hosted.web 2.0.0.D-20091109225604
    bundle com.springsource.server.repository.hosted-synthetic.context 2.0.0.D-20091109225604
```

Finally, the following example shows how to refresh an installed PAR file:

```
:> par refresh my.exciting.par 1.2.0

par my.exciting.par 1.2.0 refreshed successfully
```

plan Command

Use the `plan` command to view all the plans currently installed in dm Server, view details about a particular plan and manage its lifecycle, such as starting, stopping, refreshing, and uninstalling it.

The following table lists the options you can specify for this command.

Table 4.6. Options of the plan Command

Option	Descrption
list	Displays all the plans that are currently installed in dm Server. The `list` command displays the full name of each installed plan, its version, and its current state. Plans have similar lifecycles to other OSGi artifacts, such as bundles, and so the list of states in which a plan can be in is the same as those of bundles; see the bundle command for the list of possible states.
examine *name version*	Displays information about the specified plan; you are required to identify the plan with both its name and its version. Use the `plan list` command to view all installed plans and their versions. The command displays the following information: • The current state of the plan (see the bundle command for the full list of possible states). • Whether the plan is *scoped*. Scoping specifies whether dm Server should deploy the members of the plan in their own scope; when scoping is disabled, dm Server deploys the artifacts into the global scope and they are accessible for access by all

Option	Descrption
	other artifacts. • Whether the plan is *atomic*. When a plan is atomic, dm Server manages the lifecycle of all its member artifacts as a single entity, which means if one artifact member is started, then dm Server starts *all* the plan artifacts. If one artifact fails to start, then dm Server stops all other artifacts in the plan. • The individual members, or children, of the plan. These could be other plans, PARs, bundles, configuration artifacts, and so on.
start *name version*	Starts the specified plan. You must specify both the full name of the plan as well as the version you want to start. Use the `plan list` command to get the list of plans currently installed in dm Server. To start a plan, it must have already been resolved by dm Server, or in other words, be in the OSGi `Resolved` state. After dm Server successfully starts the plan, it is listed in the `Active` state.
stop *name version*	Stops the specified plan. You must specify both the full name of the plan as well as the version you want to stop. Use the `plan list` command to get the list of plans currently installed in dm Server. When you stop a plan, it goes from the OSGi `Active` state to the `Resolved` state, and you must re-start it if you want to use the application that the plan contains.
refresh *name version*	Updates the contents of the specified plan. You must specify both the name and version of the plan you want to refresh. Use the `plan list` command to this information. Use this command if you have changed the contents of the plan file and you want to refresh the artifact as installed in the OSGi framework.

Option	Descrption
uninstall *name version*	Uninstalls the specified plan. You must specify both the name and version of the plan you want to refresh. Use the `plan list` command to this information. When the uninstall process is complete, the plan will not show up in the list of plans displayed by the `plan list` command. If you want to use the application in the plan, you must re-install it using the `install` command.

The following example shows how to list the plans that have been installed in dm Server:

```
:> plan list

Name                                  Version                          State
com.springsource.kernel.region.springdm 2.0.0                         ACTIVE
com.springsource.server.admin.plan      2.0.0                         ACTIVE
com.springsource.server.web             2.0.0                         ACTIVE
```

The following example shows how to examine a particular plan:

```
:> plan examine com.springsource.server.web 2.0.0

State:  ACTIVE
Scoped: false
Atomic: false

Children:
    bundle com.springsource.osgi.webcontainer.tomcat 1.0.0.CI-85
    bundle com.springsource.osgi.webcontainer.core 1.0.0.CI-85
    bundle com.springsource.server.web.tomcat 2.0.0.D-20091109223734
    bundle com.springsource.server.web.core 2.0.0.D-20091109223734
    bundle com.springsource.server.web.dm 2.0.0.D-20091109223734
```

The following example shows how to stop a currently Active plan:

```
:> plan stop com.springsource.server.web 2.0.0

plan com.springsource.server.web:2.0.0 stopped successfully
```

The following example shows how to start a plan:

```
:> plan start com.springsource.server.web 2.0.0

plan com.springsource.server.web:2.0.0 started successfully
```

service Command

Use the `service` command to view all the services that have been registered in the OSGi service registry of dm Server. You can also examine specific services to discover its properties, the bundle that publishes the service, and any bundles that might consume the service.

The following table lists the options you can specify for this command.

Table 4.7. Options of the service Command

Option	Descrption
list	Displays the list of services that are currently registered in the OSGi service registry of dm Server. Each service is identified by an internal `ID` which you can then use with the `service examine` command to view the details about a particular service. The `list` command also displays the object class that implements the service and the internal `id` of the bundle that provides the service.
examine *id*	Displays detailed information about the specified service. Use the `service list` command to get the internal id of a particular service. This command displays the properties of the service, such as the object class that implements the service, the name of the bundle that publishes the service and any bundles that consume the service.

The following example shows how to use list the services currently registered in the OSGi service registry:

```
:> service list

Id  Object Class(es)                                          Providing Bundle

1   org.osgi.service.packageadmin.PackageAdmin                              0
2   org.osgi.service.permissionadmin.PermissionAdmin, ...                   0
3   org.osgi.service.startlevel.StartLevel                                  0
4   org.eclipse.osgi.service.debug.DebugOptions                             0
5   java.lang.ClassLoader                                                   0
6   org.eclipse.osgi.framework.log.FrameworkLog                             0
7   org.eclipse.osgi.framework.log.FrameworkLog                             0

... (entries removed)

65 com.springsource.osgi.webcontainer.core.spi.ServletContainer            35
66 com.springsource.osgi.webcontainer.core.WebContainer                    34
67 com.springsource.server.web.core.WebApplicationRegistry                 36
... (entries removed)
```

The following example shows how to examine a particular service:

```
:> service examine 38

Properties:
    Bundle-SymbolicName:
        com.springsource.kernel.services
    Bundle-Version:
        2.0.0.D-20091110110152
    objectClass:
        com.springsource.repository.Repository
    org.springframework.osgi.bean.name:
```

```
        repository
    service.id:
        38

Publisher: com.springsource.region.user 0.0.0 [1]

Consumer(s):
    com.springsource.kernel.userregion 2.0.0.D-20091110110152 [2]
```

install Command

Use the `install` command to deploy an artifact to dm Server. The artifact can be a bundle, PAR, plan, or configuration artifact.

The `install` command takes a single parameter: the URI of the artifact you want to deploy. For example, to deploy a bundle on the local computer, use the `file` scheme:

```
file://full-pathname-to-artifact
```

After you execute the `install` command, dm Server attempts to resolve the artifact's dependencies, and if it is successful, puts it in the `Resolved` state. At that point, you must start the artifact to be able to actually use it.

The following example shows how to install a bundle called `swf-booking-mvc.war` located in the `/home/apps` directory of the computer on which the dm Shell is being run:

```
:> install file://home/apps/swf-booking-mvc.war

Artifact bundle swf-booking-mvc.war 0.0.0 installed
```

The following example shows how to use the `bundle list` command to ensure that the bundle was indeed installed to dm Server; if you had installed a different kind of artifact, for example a plan, then you would use the appropriate command (such as `plan list`):

```
:> bundle list

Id   Name                              Version               State

0    org.eclipse.osgi                  3.5.1.R35x_v20091005  ACTIVE
1    com.springsource.region.user      0.0.0                 ACTIVE

... (entries removed for clarity)

59   com.springsource.server.splash    2.0.0.D-20091110115520 ACTIVE
60   swf-booking-mvc.war               0.0.0                 RESOLVED
```

Note that the `swf-booking-mvc.war` file is in the `Resolved` state. The following examples start the bundle, and then examine it to ensure that it is in the `Active` state:

```
:> bundle start 60

bundle swf-booking-mvc.war:0.0.0 started successfully


:> bundle examine 60

Id:              60
Name:            swf-booking-mvc.war
Version          0.0.0
State:           ACTIVE
Spring Powered:  true
Bundle Location: file:/home/juliet/dmServer2.0/springsource-dm-server-2.0.0.CI-468/work/com.springsource.kernel.deployer_2.0.0.D

Imported Packages:
    javax.crypto.interfaces [0.0.0, 0.0.0]
        exported by org.eclipse.osgi 3.5.1.R35x_v20091005 [0]
```

```
    org.omg.CosNaming.NamingContextPackage [0.0.0, 0.0.0]
        exported by org.eclipse.osgi 3.5.1.R35x_v20091005 [0]
    org.omg.DynamicAny.DynAnyFactoryPackage [0.0.0, 0.0.0]
        exported by org.eclipse.osgi 3.5.1.R35x_v20091005 [0]
    ...
```

shutdown Command

Use the `shutdown` command to shut down the dm Server instance to which you are connected. When dm Server is shutdown, the shell returns you to the operating system prompt.

The `shutdown` command does not have any options.

The following example shows how to use this command.

```
:> shutdown

Shutdown MBean called
prompt$
```

help Command

Use the `help` command on its own to get a list of all available dm Shell commands. If you specify a particular command to the `help` command, then you will get the list of options that you can pass to the command.

For example:

```
:> help


    bundle - Management and examination of bundle artifacts
    config - Management and examination of configuration artifacts
    help
    install - Install (deploy) an artifact to the server
    package - Management and examination of exported packages
    par - Management and examination of PAR artifacts
    plan - Management and examination of plan artifacts
    service - Examination of services
    shutdown


:> help bundle

bundle list                           - List all bundle artifacts that are
                                        currently installed
bundle examine   [ id | name version ] - Examine a bundle artifact
bundle start     [ id | name version ] - Start a bundle artifact. Starting this
                                        artifact starts it in the OSGi
                                        framework.
bundle stop      [ id | name version ] - Stop a bundle artifact. Stopping this
                                        artifact stops it in the OSGi
                                        framework.
bundle refresh   [ id | name version ] - Refresh a bundle artifact. Refreshing
                                        this artifact updates its contents in
                                        the OSGi framework.
bundle uninstall [ id | name version ] - Uninstall a bundle artifact
bundle diag      [ id | name version ] - Provide diagnostics for a bundle
                                        artifact
bundle headers   [ id | name version ] - Show the headers for a bundle artifact
```

5. The Web Admin Console

The dm Server Admin Console is a Web application for managing a single instance of dm Server. Using the Admin Console, you can:

- View an overview of the dm Server properties.
- View and manage the lifecycle of artifacts already deployed to the dm Server instance. Artifacts include bundles, configuration files, PARs, and plans. Lifecycle management tasks include starting, stopping, refreshing, and uninstalling the artifacts.
- Install new artifacts to dm Server. .
- View the properties of the configuration artifacts deployed to dm Server.
- View details of dump files that dm Server might have generated after encountering a problem. This feature is particularly valuable if dm Server fails to install a new artifact due to resolution failures; the dump inspector can help you discover the exact artifact causing the resolution failure.
- View an overview and details of the OSGi State of dm Server, or in other words, a list of all bundles currently installed in dm Server and their state. You can then then drill down into the details of each bundle, such as its symbolic name, packages it imports and exports, services it provides and consumes, and so on. You can also view the bundles that were deployed when an exception that generated a dump occurred.

5.1 Invoking the Admin Console

To use the SpringSource Admin Console, start the SpringSource dm Server and then enter the following URL in your browser of choice.

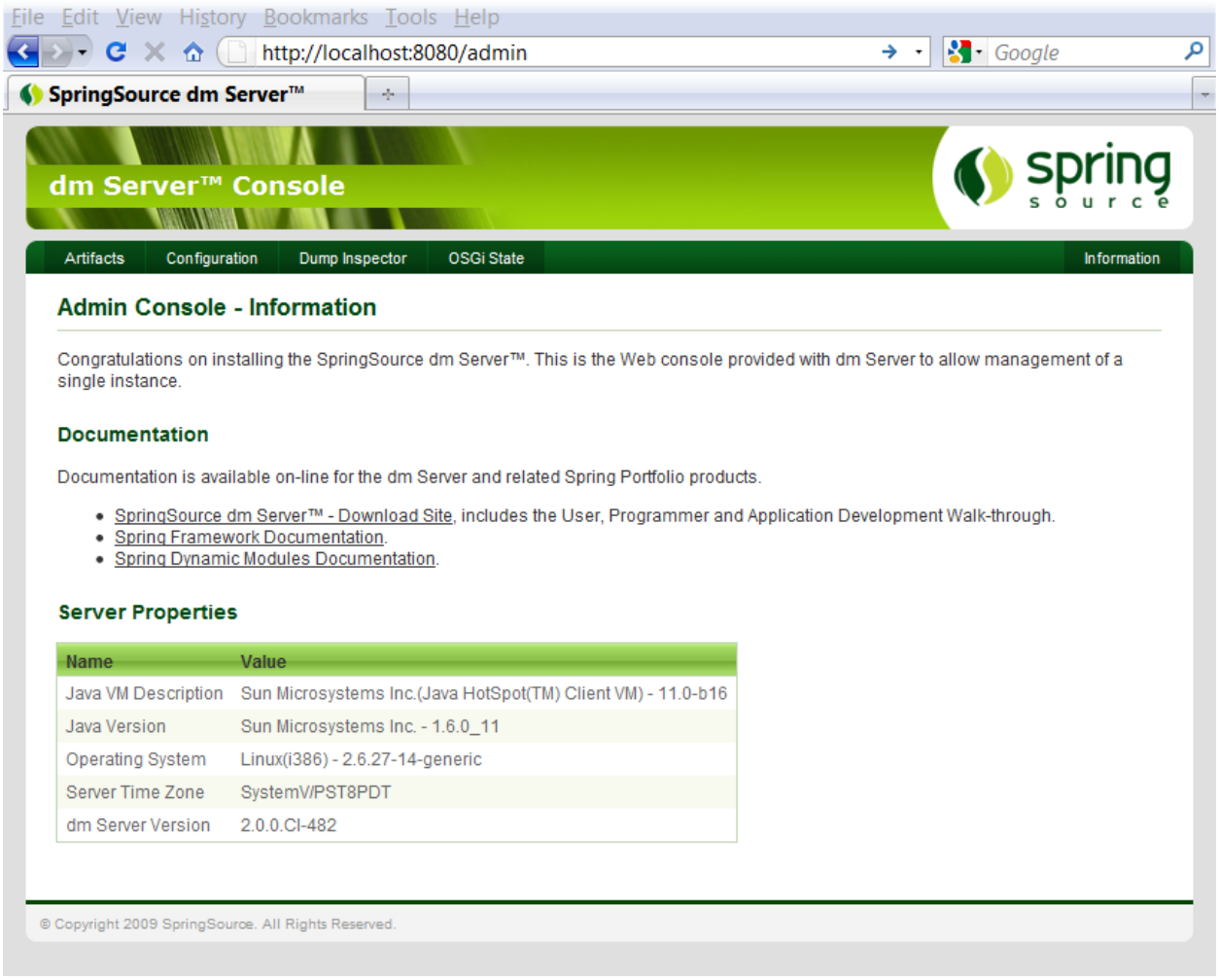
```
http://localhost:8080/admin
```

Replace `localhost` with the hostname of the computer on which the SpringSource dm Server is running if it is not the same as the computer on which you are running your browser.

The Admin Console uses basic authentication, therefore you will need to enter the default administration ID and password.

```
ID: admin
Password: springsource
```

The following graphic shows the main page of the Admin Console.



Use the links at the top of the console to perform various tasks, such as viewing and managing artifacts (**Artifacts**), viewing the properties of deployed configuration artifacts (**Configuration**), viewing details of dumps (**Dump Inspector**), and viewing the OSGi state of the dm Server instance (**OSGi State**).

You can always return to the main Admin Console page by clicking **Information** in the top right-hand corner.

The `Server Properties` section provides information about dm Server itself, such as details about the Java Virtual Machine (JVM), the operating system on which dm Server is installed, the time zone configured for the computer, and the complete version of dm Server.

Changing the Admin User

To change the ID and password for the Admin Console, update the `SERVER_HOME/config/com.springsource.kernel.users.properties` file. First specify the administration username by changing the value of the `role.admin` property. Then set the password of this new user by adding a new property called `user.`*`username`*,

where *username* refers to the actual name of the user. Finally, restart dm Server for the changes to take effect.

For example, if you want change the administration username to `juliet` with password `capulet`, change the file as follows:

```
##################
# User definitions
##################
user.juliet=capulet


##################
# Role definitions
##################
role.admin=juliet
```

The Admin Console always runs against the `admin` role.

5.2 Typical Admin Console Use Cases

The following use cases describe the typical tasks that can perform with the dm Server Admin Console:

- View and Manage the Lifecycle of Deployed Artifacts
- Install a New Artifact
- View the Properties of Deployed Configuration Artifacts
- View Details of Dump Files
- View Overview and Details of the OSGi State

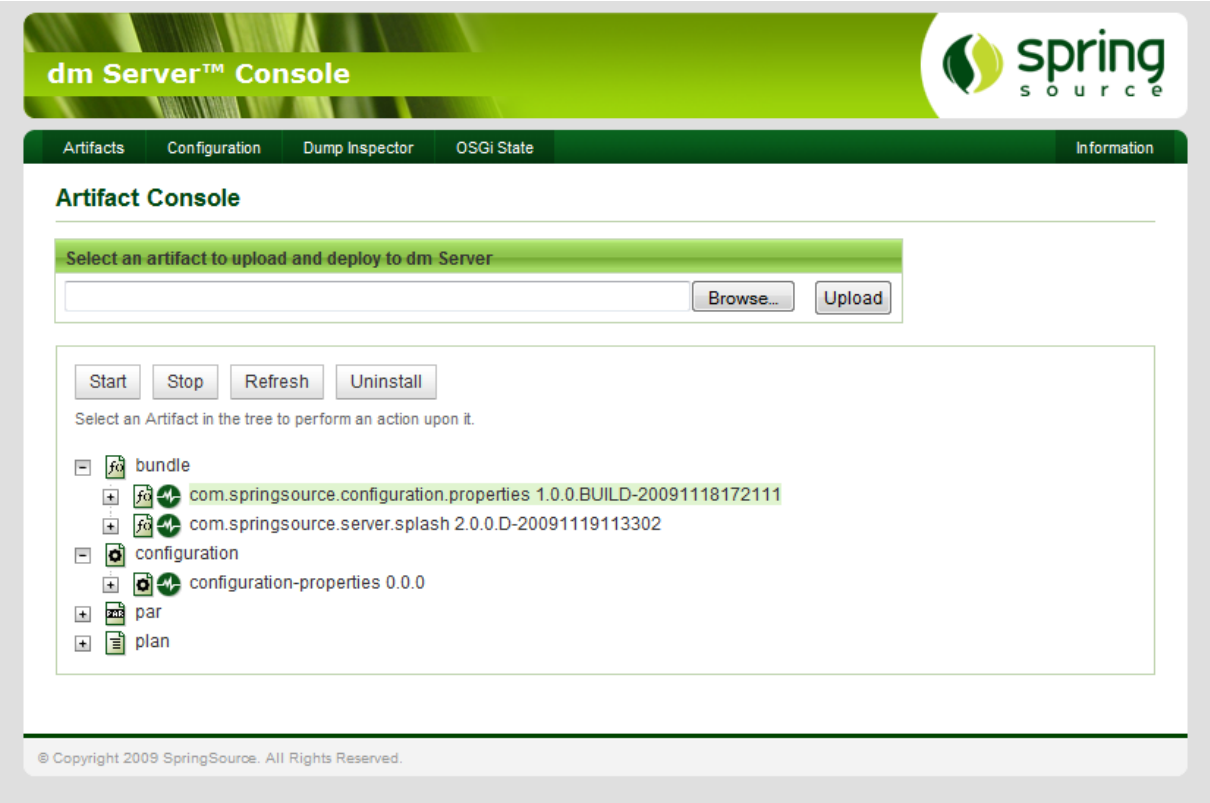
Viewing and Managing the Lifecycle of Deployed Artifacts

The following procedure describes how to view the list of artifacts that are currently deployed in the user region of dm Server. It then describes how to stop, start, refresh, and uninstall the deployed artifacts.

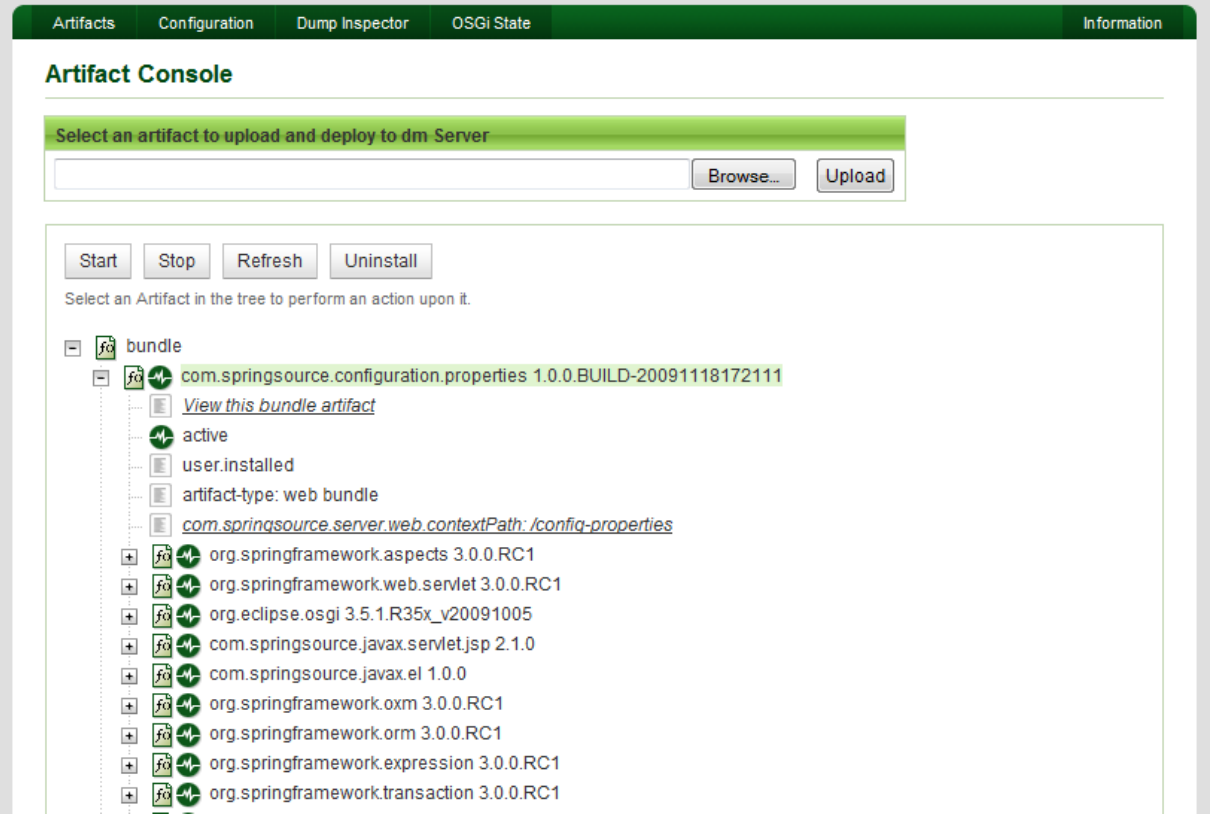
1. From the main Admin Console page, click the **Artifacts** link at the top.

   In the lower part of the page, the console displays a tree structure that displays the four kinds of artifacts that you can deploy to the user region of dm Server: bundles, configuration files, PARs, and plans. When you first install dm Server, there will already be a number of artifacts deployed related to the Admin console itself, the main splash screen, the repository, and so on.

   The following graphic shows an expanded tree that displays a few of the deployed artifacts:



2. To view details of a particular artifact, click the "+" to the left of the artifact to expand the tree. The following graphic shows an expanded `com.springsource.configuration.properties` bundle:



The particular details that the Admin Console displays depends on the artifact. For example, for all artifacts you can view their state and how it was installed (such as by a user using the Admin Console or programmatically). The two most common states are Active (running and ready to be used) and Resolved (all dependencies resolved but you must start it before you can use it.) An artifact can also be in one of the transition states, such as Starting and Stopping.

As shown in the preceding graphic, the Admin Console provides a link for Web modules that you can click on to actually invoke the application (`com.springsource.server.web.contextPath:/config-properties` in the example above.)

For PARs and plans, the Admin Console also displays whether the artifact is:

- **Scoped**. Scoping specifies whether dm Server should deploy the members of the PAR/plan in their own scope; when scoping is disabled, dm Server deploys the artifacts into the global scope and they are accessible for access by all other artifacts.

- **Atomic**. When a PAR/plan is atomic, dm Server manages the lifecycle of all its member artifacts as a single entity, which means if one artifact member is started, then dm Server starts all the PAR/plan artifacts. If one artifact fails to start, then dm Server stops all other artifacts in the PAR/plan.

The following graphic shows details of a PAR, in particular that it is both scoped and atomic:

Finally, for bundles, PARs, and plans, you can see the list of bundles that they depend on; this typically means the bundles that contain the packages that they import.

3. To manage the lifecycle of an artifact, click on its name in the expanded tree to enable the lifecycle buttons. Then, depending on the current state of the artifact, you can:

   - Start the artifact. All dependencies of the artifcat must have been resolved for you to start it. After successfully starting the artifact, it is in the Active state and you can use the application associated with the artifact.

   - Stop the artifact. This moves the artifact from an Active to Resolved state, and you cannot use the application associated with the artifact.

   - Refresh the artifact. This action updates the physical contents of the artifact; use this button when you have changed the artifact in some way and you want your changes to take effect.

   - Uninstall the artifact. This action removes the artifact from dm Server and it does not show up in the Admin Console anymore. To use the application associated with this artifact, you must re-install the artifact.

Installing a New Artifact

The following procedure describes how to install a new artifact (bundle, PAR, plan, or configuration file.) The procedure is similar for all types of artifacts; the procedure uses a WAR file as an example.

1. From the main Admin Console page, click the **Artifacts** link at the top.

2. Click the **Browse** button to invoke the file loader application for your platform. Note that the Browse button searches the computer that is running the browser in which you invoked the

Admin Console and *not* the computer on which dm Server is running, in the case where they are different.

Use the file loader to find the artifact. This can be a WAR file bundle, a configuration artifact that contains properties, an XML file that corresponds to a plan, or a PAR file.

3. Click **Upload** to actually upload the artifact to dm Server.

   dm Server automatically attempts to resolve all dependencies, and then puts the artifact in an Active state if possible. If all is successful, the message `Artifact Deployed` appears next to the **Artifact Console** header. If there is an error, a message to that effect is display; to get more details about the error, see the terminal window from which you started dm Server.

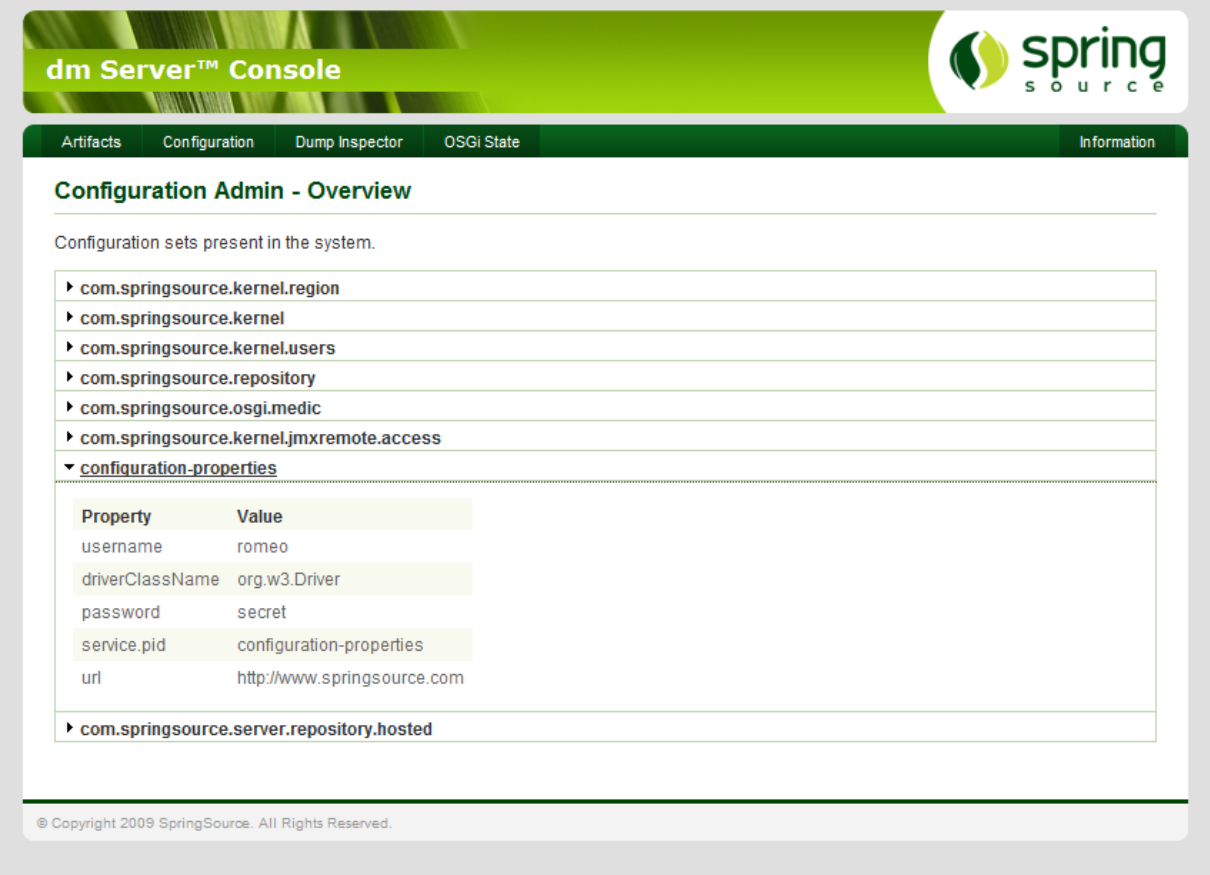
4. Expand the artifact tree to view your newly deployed artifact. If dm Server installed it without errors, it should show up in the appropriate section and be in an Active state.

Viewing Properties of Deployed Configuration Artifacts

The following procedure describes how you can view the list of configuration artifacts that are currently deployed to dm Server, and then view the specific properties that are defined for a particular configuration artifact.

1. From the main Admin Console page, click the **Configuration** link at the top.

   The Admin Console displays all the configuration artifacts that are currently deployed, as shown in the following graphic:



2. To view the properties defined for a particular configuration artifact click the arrow to the left of its name.

Viewing the Details of Dump Files

The following procedure describes how to view the details of any service dumps that have occurred in dm Server. Each time a dump is triggered for dm Server, the server creates a directory in `$SERVER_HOME/serviceability/dump` with a name corresponding to the time the dump occurred, and then the server populates the directory with detailed information. Using the Admin Console, you can easily view this information.

A service dump is triggered when there is either a failure in the dm Server code or dm Server detects a thread deadlock in either its own code or a user application. The service dump contains a snapshot of all the important state from the running dm Server instance. **NOTE:** This snapshot is not intended for end user consumption but is useful for service personnel.

1. From the main Admin Console page, click the **Dump Inspector** link at the top.

2. In the drop-down box on the left, select the dump you want to inspect based on its timestamp.

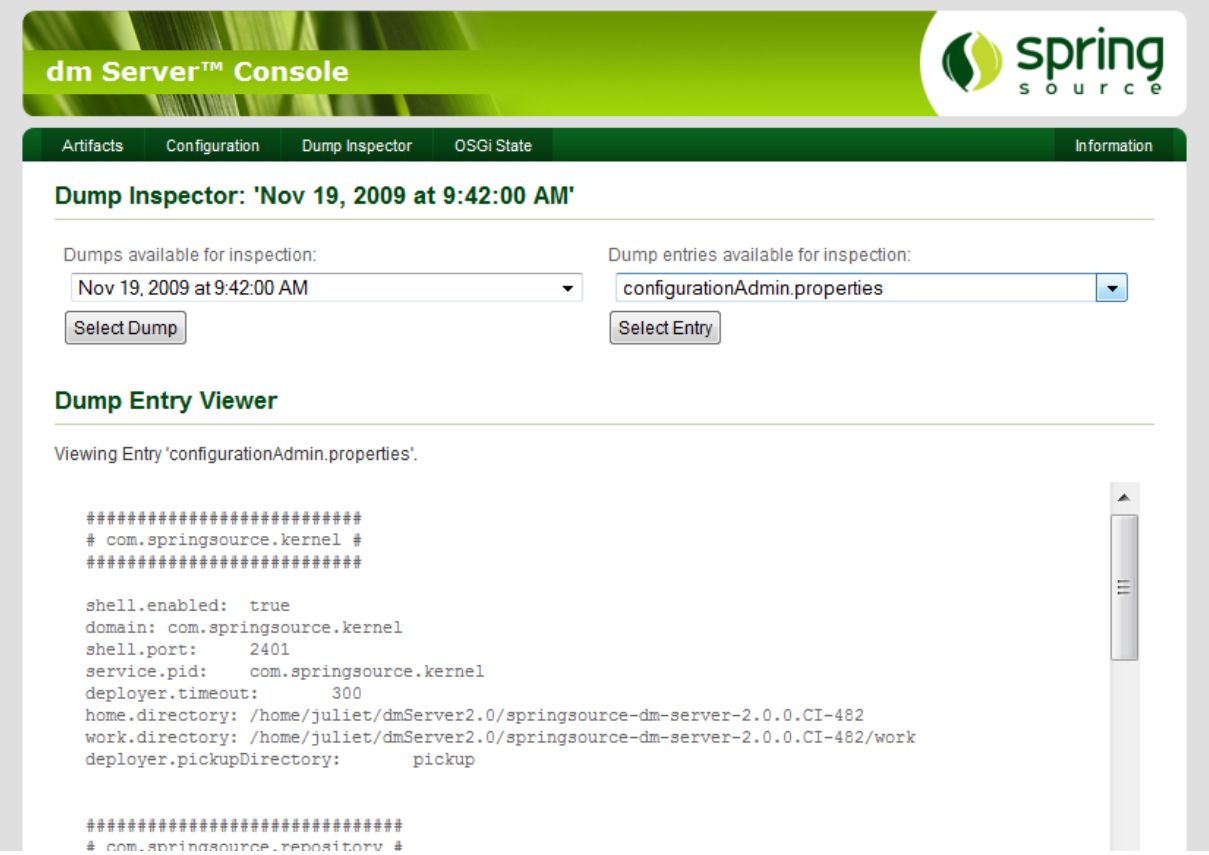
3. Click **Select Dump**.

4. In the right drop-down box, select the type of dump information you want to view.

   For example, `summary.txt` provides a short summary of why the dump might have occurred. The `thread.txt` option provides information about the state of the dm Server threads at the time of the dump, including any that were deadlocked. The `repository` options provide information about what was in the external and user repositories at the time of the dump. The `configurationAdmin.properties` option provides a snapshot of the complete configuration of dm Server, including the kernel and repositories.

5. Click **Select Entry**.

   The Admin Console displays the information in the Dump Entry Viewer, as shown in the following graphic:



Viewing Overview and Details of the OSGi State

The following procedure describes how you can view the OSGi state of the dm Server, either currently or at the time that a particular service dump occurred. The OSGi state is a list of bundles that are currently installed as well as a list of all the services that are provided by these bundles.

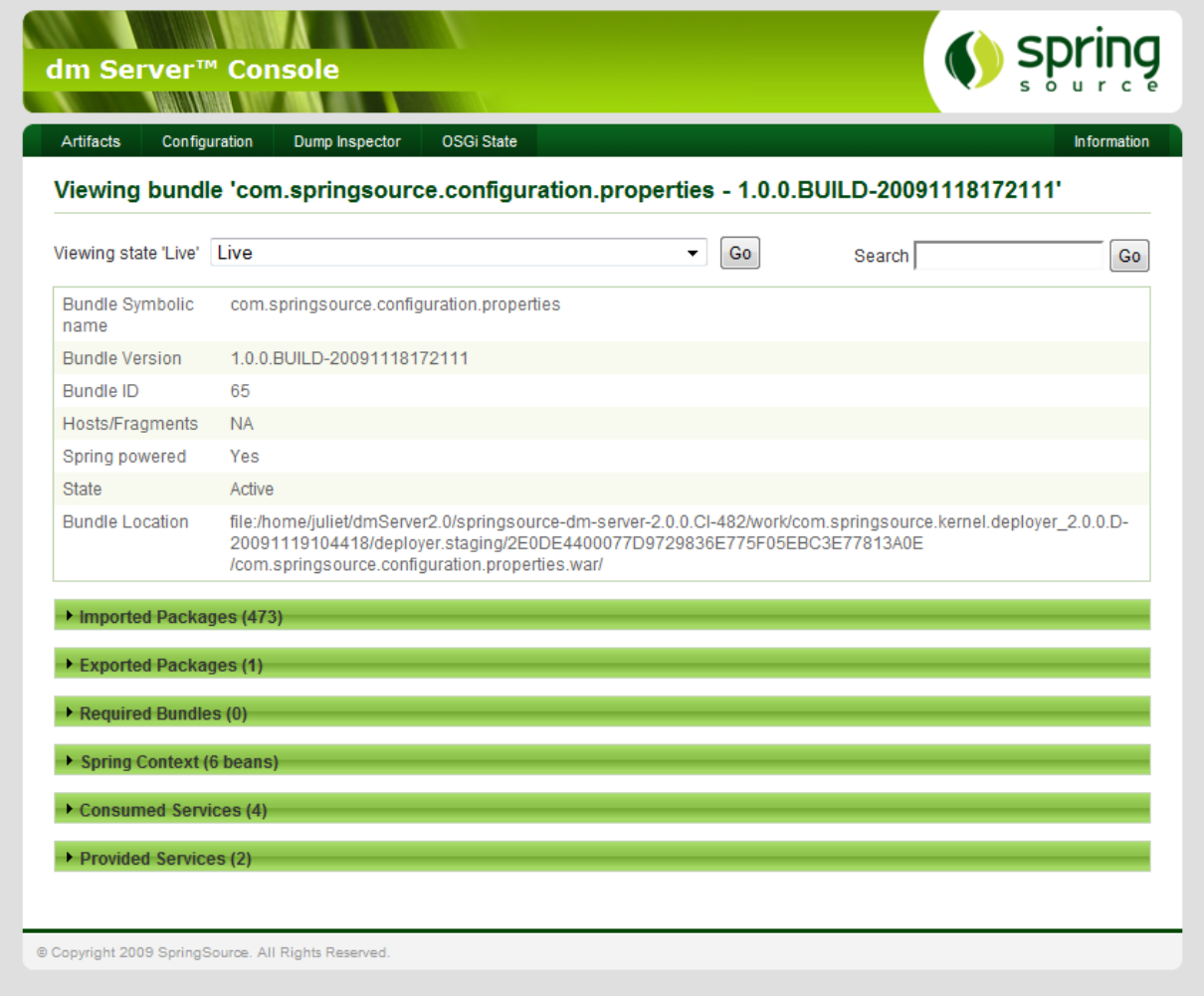
1. From the main Admin Console page, click the **OSGi State** link at the top.

   By default, the Admin Console displays the complete list of bundles that are currently installed in dm Server.

   For each bundle, the console displays its internal ID, its symbolic name, its version, and its current state (usually either Active or Resolved.)

2. To view the bundles that were installed at the time of a service dump, select it based on its timestamp from the drop-down box on the right and click **Go**.

3. To view details about a particular bundle, click on its bundle ID. A full description of the bundle is displayed, as shown in the following graphic:



   The console displays again the symbolic name, version, and internal ID of the bundle. It then displays whether the bundle is Spring powered and the exact physical location of the bundle JAR file on the computer that hosts dm Server.

   The console then displays the full list of packages that the bundle imports, as well as the

bundles that in turn export these imported packages. The console also displays the packages that the current bundle exports, and then in turn the list of other installed bundles that are currently importing these exported packages. For each package, you can drill down and view details of the corresponding bundle.

Similarly, the console displays the consumed and provided OSGi services.

Finally, the console also displays information about the Spring context, if the bundle is Spring powered.

4. To view the full list of OSGi services, click the `Services Overview` link from the main OSGi state page

5. Typically, the list of bundles and services can be very long, making it difficult to find a particular bundle. Use the **Search** box at the top right corner to narrow down the list of displayed bundles.

6. The Provisioning Repository

6.1 Overview of the Provisioning Repository

This section describes the provisioning repository feature of SpringSource dm Server, the reasons for using it, and how to configure it.

In most use cases, your application has a dependency on one or more separate artifacts; these artifacts might include OSGi bundles, configuration artifacts, third-party libraries, PARs or plans. A typical example is a Spring application that depends on a third-party library such as Spring Framework or Hibernate.

The way you express this dependency depends on the artifact. For example, a plan is by definition a list of dependent bundles.

Libraries are another example. Some third-party dependencies consist of multiple bundles but are logically one unit. To support this, the SpringSource dm Server introduces the concept of a library. A library is a collection of related bundles that can be referenced as a whole. You typically express the dependencies between your application and third-party libraries using the `Import-Package` or `Import-Library` manifest header in the `MANIFEST.MF` file of your application. The `Import-Package` header is standard to OSGi; `Import-Library`, however, is specific to SpringSource dm Server.

For additional details about the creation and usage of libraries, as well as general information about dependencies, see Programmer's Guide.

In SpringSource dm Server, you store all third-party dependencies required by your applications, such as Spring Framework and Hibernate, as artifacts in the provisioning repository. As mentioned above, you can store the following types of artifacts in the repository:

- OSGi bundles
- Libraries
- PARs
- Plans
- Configuration Artifacts

When you deploy your application, SpringSource dm Server installs the bundle in which it is packaged to the dm Server runtime; part of this internal installation procedure is to satisfy all the application's dependencies. If your application has a dependency that cannot be satisfied from the bundles that you have already deployed (and dm Server has thus installed), the dm Server searches the provisioning repository for an artifact that can satisfy that dependency.

The provisioning repository for a particular instance of SpringSource dm Server can include artifacts in the following general locations:

- Local: This means that artifacts have been physically installed in the provisioning repository directory structure of the local SpringSource dm Server instance. The artifacts in a local repository include installed third-party libraries, bundles supplied by dm Server, bundles supplied by an end user, and internal bundles used only by dm Server. You can further categorize this location into `external` directories that adhere to a specified search pattern and are scanned by dm Server just at startup, or `watched` directories that point to a single directory location and dm Server scans on a regular basis.

- Remote: This means that a local instance of SpringSource dm Server gets the artifact from a remotely-hosted repository that is physically located on a remote SpringSource dm Server instance.

You configure the provisioning repository using the `SERVER_HOME/config/com.springsource.repository.properties` file.

As previously described, a particular instance of SpringSource dm Server can itself also act as a repository host for remote server instances to use when satisfying the dependencies of the applications deployed to it. In this case, you configure a hosted repository using the `SERVER_HOME/config/com.springsource.repository.hosted.properties` file. Typically, only remote clients use hosted repositories and their contents; the SpringSource dm Server instance that actually hosts the repository does not typically use the artifacts in it. Rather, it uses artifacts in its local repository.

Making a third-party dependency available to your application is simply a matter of adding its artifact to the appropriate location in the provisioning repository. This could be either in the local directories or the remote ones if you are getting artifacts from a remotely-hosted repository.

Local Repository Structure

When you first install SpringSource dm Server, the local provisioning repository is located at `$SERVER_HOME/repository` by default and consists of two main directories: `ext` and `user`. The `ext` directory contains bundles and libraries supplied with the SpringSource dm Server and `usr` contains bundles and libraries installed by the end user.

Installing Artifacts to a Repository

To install an artifact into the default repository, simply copy it into the `$SERVER_HOME/repository/usr` directory.

If you have configured additional watched or external repositories (additional, that is, to the default ones already configured in a freshly-installed dm Server instance), you install the artifacts in the same way: simply copy the files to the configured directories. You configure additional watched or external repositories in the same file as the default repositories: `SERVER_HOME/config/com.springsource.repository.properties`.

When you install a plan or a library, you must ensure that all referenced bundles within the plan or library have been installed as well.

Artifacts must have unique names so it is considered best practice to include the version number in the file name, allowing for multiple versions of the artifact to be installed at the same time. For example, a bundle file name might be `my-exciting-bundle.2.1.0.jar`.

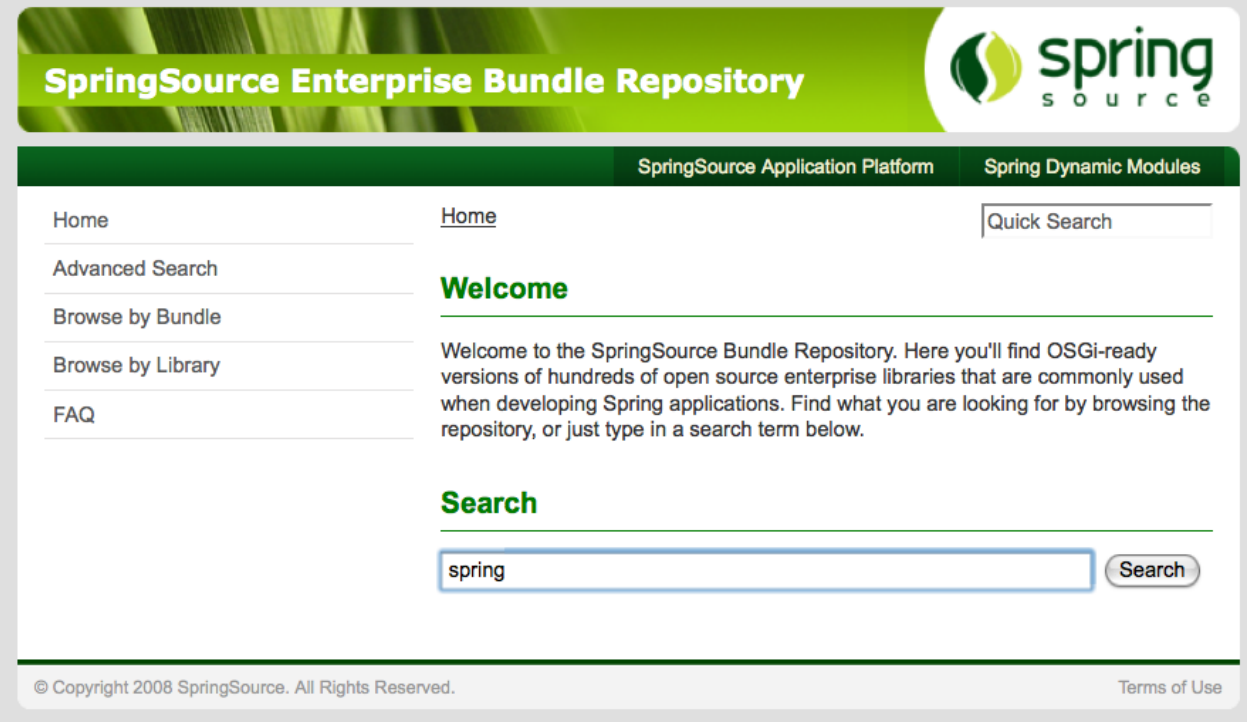
In some cases the SpringSource dm Server manages to automatically detect changes in its provisioning repository at runtime, thereby avoiding the need to restart the dm Server.

Of specific relevance during development is picking up changes to an application's direct dependencies during deployment of the application. For example, if you deploy an application and receive a message that a dependency is missing, you can simply add the dependency to the repository and then redeploy the application. The redeploy will cause the new dependency to be picked up, allowing progress to be made without restarting the dm Server. For other changes such as addition of indirect dependencies, the SpringSource dm Server must be restarted to pick up any changes to the provisioning repository.

6.2 Finding and Downloading Bundles from the SpringSource Enterprise Bundle Repository

The SpringSource Enterprise Bundle Repository is a public collection of open source libraries commonly used for developing enterprise Java applications with the Spring Framework and dm Server. It contains more than 400 of the most popular enterprise Java libraries made available for general use in an OSGi-ready format. You can browse the collection and then download the bundles that you need into your own local repository.

The SpringSource Enterprise Bundle Repository is located here.



You can find bundles in the repository using a number of options. You use the ‘Search’ facility by typing in a keyword. The matching criteria returned can be explored by name, symbolic name, class, package or resource.

There is also the option of clicking on ‘Browse by Bundle’. This gives an alphabetical list of bundles. You can select the desired bundle to see details and find the download link. Finally, you can also choose to ‘Browse by Library’, which allows you to browse the alphabetical list of libraries in the repository.

6.3 Configuring the repository

Details of how to configure a SpringSource dm Server installation’s provisioning repository can be found in Configuring the Provisioning Repository. See Configuring a Hosted Repository for details on how to configure a repository that remote clients can access, also called a hosted repository.

The two configuration chapters describe the format of the repository properties files of SpringSource dm Server, how to add new directories to the local repository, how to configure the repository to get artifacts from a remote repository hosted on a remote dm Server, instance, and how to configure the local dm Server instance to itself host a repository that other remote servers access.

7. Serviceability

Logging (both event logging and trace logging) in dm Server comes in two forms: application logging and server logging. Both are configured together in the `serviceability.xml` file in the `config` directory. This takes the form of a Logback configuration—dm Server uses a Logback implementation behind the SLF4J logging interface.

What was previously referred to as *Logging*, is now referred to as *Event Logging*, and what was previously referred to as *Trace logging* (or *Tracing*) is now simply *Logging*.

7.1 Event log files

Event log files are low-volume logs of important events in SpringSource dm Server. Each server message written to an event log file is accompanied by a code enclosed in angle brackets. An example is shown below:

```
[2009-08-25 15:04:57.044] server-dm-7                <OF0001I> OSGi telnet console available on port 2401.
```

(For a description of the log code syntax, see Appendix A, *Event log codes*.) The format of event log messages from the server is fully configurable.

By default, event log messages are stored in `$SERVER_HOME/serviceability/eventlogs/eventlog_`*i*`.log` and output to the console. The index *i* varies from 1 to 4, at 10Mb boundaries. An examination of the *Logback* configuration will show these defaults being set. They may be modified.

For a description of the syntax and facilities provided by this file see the *Logback* documentation (referenced in Appendix C, *Further Reading*).

7.2 Trace (Logging)

The SpringSource dm Server's logging (trace) support serves two main purposes:

- It provides global trace files that capture high-volume information regarding the SpringSource dm Server's internal events. The files are intended for use by support personnel to diagnose runtime problems.
- It provides application trace files that contain application-generated output. This includes output generated using popular logging and tracing APIs, as well as output generated by calls to `System.out` and `System.err`. These files are intended for use by application developers and system administrators.

By default, the dm Server trace file is called `$SERVER_HOME/serviceability/logs/dm-server/log_`*i*`.log`, and, again by default, the application trace files are called

application_name/`log_`*i*`.log`, where *application_name* is automatically set by dm Server for each application artifact installed and run (it is a combination of the artifact name and the version).

The index *i* varies from 1 to 4, on a rolling basis, as each log file exceeds 10Mb.

Entries in trace files are by default of the form <timestamp> <thread-name> <source> <level> <entry-text>. For example:

```
[2008-05-15 09:09:46.940] server-dm-2 org.apache.coyote.http11.Http11Protocol I Initializing Coyote HTTP/1.1 on http-48080
```

although this format is completely determined by the Logback configuration file `serviceability.xml`.

Application Output

SpringSource dm Server provides advanced support for capturing and tracing application-generated output by automatically separating trace output on a per-application basis and will also capture any `System.out` and `System.err` output.

Per-application trace

SpringSource dm Server uses SLF4J interfaces to Logback, and the root logger (by default) captures all logging output and appends it to the application-specific trace files as described above. To modify this, define application-specific loggers in the `serviceability.xml` file in the normal way.

System.out and System.err

`System.out` and `System.err` output from applications is, by default, captured in the application's trace file. This happens because the output streams are intercepted and written to the loggers named `System.out` and `System.err` respectively. Since there are no explicit loggers defined with these names in the `serviceability.xml` file, this output is logged by the root logger (which captures `INFO` level and above).

The capture of `System.out` and `System.err` output is configured in the `config/com.springsource.osgi.medic.properties` file by the `log.wrapSysOut` and `log.wrapSysErr` properties. By default the properties have a value of `true` and capture is enabled. Capture can be disabled by configuring the properties with a value of `false`.

The trace entries for `System.out` and `System.err` output are of the form:

```
[2008-05-16 09:28:45.874] server-tomcat-thread-1 System.out Hello world!
[2008-05-16 09:28:45.874] server-tomcat-thread-1 System.err Hello world!
```

The third column indicates where the output came from (`System.out` or `System.err`).

To over-ride this behaviour, simply define explicit loggers named `System.out` and/or

`System.err` in the configuration file to send this output to an appender of your choice. Be aware that all applications' output streams will be caught by these loggers, and that a sifting appender might be useful to separate them.

7.3 Service Dumps

A service dump is triggered when one of the following events occurs:

1. A failure is detected in the SpringSource dm Server code, or

2. a thread deadlock is detected.

A service dump contains a snapshot of all the important state from the running SpringSource dm Server instance. This snapshot is not intended for end user consumption but is useful for service personnel.

By default, service dumps are created in `$SERVER_HOME/serviceability/dump`.

8. Working with Applications

8.1 Deploying Artifacts

In the context of SpringSource dm Server, *deploying* refers to installing an artifact to the server and then starting it to make it available to users. Typically, when you install an artifact, dm Server automatically starts it as long as the server is able to successfully resolve all its dependencies. For this reason, the terms *deploying* and *installing* are often used interchangeably.

You deploy artifacts to SpringSource dm Server using either the hot-deploy directory on the file system or by using the Admin Console. The artifacts that you can deploy to dm Server are:

- Bundles, including Web applications
- PARs
- Plans
- Configuration Files

Hot Deploy

To hot deploy an artifact, copy it into the pickup directory (by default `$SERVER_HOME/pickup`):

```
prompt$ cd /home/applications
prompt$ cp helloWorld.war $SERVER_HOME/pickup
```

When the artifact is hot deployed, messages similar to the following appear in the log file:

```
[2009-12-10 06:41:01.021] fs-watcher          <HD0001I> Hot deployer processing 'CREATED' event for file 'helloWorld.war
[2009-12-10 06:41:01.087] fs-watcher          <DE0000I> Installing bundle 'helloWorld' version '0.0.0'.
[2009-12-10 06:41:01.274] fs-watcher          <DE0001I> Installed bundle 'helloWorld' version '0.0.0'.
[2009-12-10 06:41:01.397] fs-watcher          <DE0004I> Starting bundle 'helloWorld' version '0.0.0'.
[2009-12-10 06:41:01.414] Thread-3            <WE0000I> Starting web bundle 'helloWorld' version '0.0.0' with context pa
[2009-12-10 06:41:01.537] Thread-3            <WE0001I> Started web bundle 'helloWorld' version '0.0.0' with context pat
[2009-12-10 06:41:01.550] start-signalling-1  <DE0005I> Started bundle 'helloWorld' version '0.0.0'.
```

If there is a problem with the deployment, such as the server being unable to resolve all dependencies, the console and log both show an error message to help you with troubleshooting.

If there are no problems, dm Server automatically starts the artifact so that it is immediately available to users.

Deploying Using the Admin Console

The Admin Console allows you to upload a file, which will be deployed automatically, from your local file system to the SpringSource dm Server. As soon as SpringSource dm Server

deploys the artifact, it appears in the list of artifacts in the Admin Console. Note that the GUI for uploading varies according to the browser and operating system you use.

See Installing a New Artifact for details about using the Admin Console to install (deploy) an artifact. See The Web Admin Console for general informatin about the Admin Console.

What Happens When You Deploy

When you deploy an artifact, either using hot-deployment or the Admin Console, dm Server copies the file to its work directory (`SERVER_HOME/work`) and registers it in its internal registry. The server then checks any dependencies the artifact might have to see if deployment can go ahead, and if all dependencies are resolved, SpringSource dm Server starts the artifact. Because of all these additional internal activities, you should NOT simply copy the artifact into the `work` directory and assume it will be deployed, because SpringSource dm Server will not do so.

Deployment Ordering

When deploying bundles that have dependencies, it is important that you deploy them in the correct order. SpringSource dm Server honors this ordering when it redeploys the artifacts on startup.

If you use hot deployment to deploy your artifacts, be sure to copy the corresponding files into the pickup directory one-by-one. Copying the files in one group, for example by using a single `cp` command. provides no guarantees of ordering.

Restrictions

The SpringSource dm Server does not support deploying fragment bundles.

8.2 Undeploying Artifacts

You undeploy artifacts from SpringSource dm Server by using either the hot-deploy directory on the file system, or the Admin Console.

Note: As with deploying, in this guide the terms *undeploying* and *uninstalling* are used interchageably.

Hot Undeploy

To hot-undeploy an artifact, remove the corresponding file from the pickup directory (by default `$SERVER_HOME/pickup`):

```
prompt$ cd $SERVER_HOME/pickup
```

```
prompt$ rm helloWorld.war
```

When SpringSource dm Server completes the undeployment of the artifact, messages similar to the following appear in the log:

```
[2009-12-10 06:46:33.254] fs-watcher   <HD0001I> Hot deployer processing 'DELETED' event for file 'helloWorld.war'.
[2009-12-10 06:46:33.259] Thread-3     <WE0002I> Stopping web bundle 'helloWorld' version '0.0.0' with context path '/he
[2009-12-10 06:46:33.285] Thread-3     <WE0003I> Stopped web bundle 'helloWorld' version '0.0.0' with context path '/hel
[2009-12-10 06:46:33.290] fs-watcher   <DE0010I> Stopping bundle 'helloWorld' version '0.0.0'.
[2009-12-10 06:46:33.295] fs-watcher   <DE0011I> Stopped bundle 'helloWorld' version '0.0.0'.
[2009-12-10 06:46:33.302] fs-watcher   <DE0013I> Uninstalling bundle 'helloWorld' version '0.0.0'.
[2009-12-10 06:46:33.319] fs-watcher   <DE0014I> Uninstalled bundle 'helloWorld' version '0.0.0'.
```

Undeploying Using the Admin Console

You can undeploy only whole artifacts from the Admin Console, or in other words, you cannot undeploy the separate modules or bundles that make up an artifact.

The only artifact that you cannot undeploy from the Admin Console is the Admin Console itself. If you need to undeploy this application, you must remove it from the pickup directory (by default `SERVER_HOME/pickup`); the name of the artifact is `com.springsource.server.admin-2.0.0.0.plan`.

See Viewing and Managing the Lifecycle of Deployed Artifacts for details about uninstalling (undeploying) an artifact using the Admin Console. The high-level steps are to highlight the artifact in the artifact tree then click `Uninstall`.

9. Configuring the SpringSource dm Server

You use configuration files in the `SERVER_HOME/config` directory to configure dm Server. This section divides the configuration of the server into the following high-level tasks:

- Configuring the kernel and the user region.
- Configuring embedded Tomcat servlet container.
- Configuring serviceability.
- Configuring the local provisioning repository.
- Configuring the hosted repository.

9.1 Configuring the dm Kernel and User Region

This section provides information about configuring the dm Server kernel and the user region by updating the following files in the `SERVER_HOME/config` directory:

Table 9.1. Kernel Configuration Files

Property File	Description
`com.springsource.kernel.properties`	Configures kernel deployment and the dm Shell of dm Server.
`com.springsource.kernel.userregion.properties`	Configures the user region of dm Server.
`com.springsource.kernel.users.properties`	Configures the users that are allowed to access the dm Shell and Admin Console, and roles to which they map.
`com.springsource.kernel.jmxremote.access.properties`	Configures the permissions for users that are allowed to access the dm Shell and Admin Console.
`com.springsource.kernel.authentication.config`	Configures the Java Authentication and Authorization Service (JAAS) for the Tomcat server users.

Configuring Deployment

You can configure three properties of deployment: the pickup directory into which you copy applications for hot-deployment, the deployment timeout, and whether automatic cloning of deployed bundles is enabled.

To change any of these properties, edit the `deployer.XXX` properties of the `SERVER_HOME/config/com.springsource.kernel.properties` file. The following table describes these properties.

Table 9.2. Deployment Configuration Properties

Property	Description
`deployer.timeout`	Specifies the amount of time, in seconds, after which dm Server times out while trying to deploy a bundle, library, or plan. The default value is `300`. If you want to disable deployment timeout, specify `0`.
`deployer.pickupDirectory`	Specifies the absolute or relative path to the pickup directory to which you copy applications for hot-deployment. Relative paths are relative to `SERVER_HOME`. The default value is `./target/pickup.`.

The following listing displays the default configuration distributed with the dm Server; only relevant sections of the `com.springsource.kernel.properties` file are shown.

```
deployer.timeout=300
deployer.pickupDirectory=pickup
```

As the default configuration shows, the default pickup directory is `SERVER_HOME/pickup` and the deployment timeout is 300 seconds.

Configuring the dm Shell

The dm Shell is a command line utility that allows you to examine artifacts currently installed to a particular dm Server instance, manage the lifecycle of the installed artifacts, install new artifacts, and shutdown the server. For complete documentation on the dm Shell, see Chapter 4, *The dm Shell*.

You configure the dm Shell by updating the `shell.XXX` properties in the `SERVER_HOME/config/com.springsource.kernel.properties` file, as described in the following table:

Table 9.3. dm Shell Configuration Properties

Property	Description
`shell.enabled`	Specifies whether the dm Shell is enabled or not. Valid values are `true` or `false`.
`shell.port`	Defines the port on which you can remotely access the dm Shell. If not set, the shell is only

Property	Description
	available from stdout of the dm Server process.

The following example shows the default dm Shell configuration in a freshly-installed `com.springsource.kernel.properties` file; only the relevant section of the file is shown.

```
shell.enabled=true
shell.port=2401
```

The example shows that the dm Shell is enabled by default, and you connect to it remotely using the port `2401`.

Configuring the User Region

The user region is the subsystem of dm Server that supports deployed applications, both your own user applications and those of the server itself, such as the Admin Console. The user region is deliberately isolated from the kernel, which makes it much simpler for you to manage your applications with the Admin Console or dm Shell because the internal server bundles are not visible.

You configure the user region by updating properties in the `SERVER_HOME/config/com.springsource.kernel.userregion.properties` file; these properties are described in the following table.

WARNING: SpringSource strongly recommends that you update only the `initialArtifacts` property; updating the other properties could cause dm Server to fail. These properties are documented for your information only.

Table 9.4. User Region Configuration Properties

Property	Description
`baseBundles`	Specifies the hard-coded list of bundles that dm Server installs directly into the user region. SpringSource dm Server does not perform any automatic dependency satisfaction for these bundles; in other words, you only get the bundles in the list and nothing more.
`packageImports`	Specifies the packages that exist in the kernel that dm Server imports into the user region so that they are in turn available to be imported by bundles in the user region. This property supports a `.*` wildcard. For example, `com.springsource.util.*` will import all packages that start with

Property	Description
	`com.springsource.util`.
`serviceImports`	Specifies the services in the kernel that are imported into the user region so that they're available to bundles in the user region.
`serviceExports`	Specifies the services in the user region that are imported into the kernel so that they're available to bundles in the kernel.
`inheritedFrameworkProperties`	Specifies the framework properties, configured in the `SERVER_HOME/lib/com.springsource.kernel.launc` file, that will also be set on the user region's nested framework.
`initialArtifacts`	Specifies the artifacts that dm Server deploys into the user region when the server starts. SpringSource dm Server performs dependency satisfaction when it deploys these artifacts. This means that you only need to list the top-level artifacts that you care about; dm Server automatically installs any other artifacts upon which they depend from the repository. You can use this property to convert a dm Server into a dm Kernel by removing the `repository:plan/com.springsource.server.web` plan.

Configuring Authentication

SpringSource dm Server uses the Java Authentication and Authorization Service (JAAS) framework to authenticate the administration user that connects to dm Servers using the Admin Console or dm Shell. This section describes how the authentication mechanism is configured by default, and the files that you need to update if you want to change the administration user, change their password, and so on.

The `SERVER_HOME/config/com.springsource.kernel.authentication.config` file configures the underlying authentication technology for dm Server. The short file consists of the following entry:

```
dm-kernel {
       com.springsource.kernel.authentication.KernelLoginModule REQUIRED;
};
```

The entry is named `dm-kernel`. This name corresponds to the `<Realm>` element in the `SERVER_HOME/config/tomcat-server.xml` file that configures the JAAS authentication mechanism for the `Catalina` service of the Tomcat servlet container. The `dm-kernel` entry specifies that the JAAS LoginModule that dm Server uses to authenticate users is `com.springsource.kernel.authentication.KernelLoginModule` and that this `KernelLoginModule` is required to "succeed" in order for authentication to be considered successful. The `KernelLoginModule` succeeds only if the name and password supplied by the user are the ones it expects. The default administration username/password pair for dm Server is `admin/springsource`.

You configure the administration user in the `com.springsource.kernel.users.properties` file. The default file for a freshly installed dm Server is as follows:

```
##################
# User definitions
##################
user.admin=springsource

##################
# Role definitions
##################
role.admin=admin
```

The administration user that connect to the Admin Console and dm Shell must have the `admin` role. The preceding file shows how, by default, the `admin` role is assigned the `admin` user with password `springsource`.

If you want to change the administration user, update the `com.springsource.kernel.users.properties` file. For example, if you want the `juliet` user, with password `supersecret`, to be the new adminstration user, update the file as shown:

```
##################
# User definitions
##################
user.juliet=supersecret

##################
# Role definitions
##################
role.admin=juliet
```

Be sure to restart dm Server after you make this change for it to take effect.

The final file involved in dm Server authentication is `SERVER_HOME/config/com.springsource.kernel.jmxremote.access.properties`. This file specifies the JMX access privileges that the administration user has; by default they are read and write, as shown in the following listing:

```
admin=readwrite
```

The only other value you can enter is `readonly`, which means that the adminstration user would only be able to view information using the Admin Console and dm Shell.

9.2 Configuring Serviceability

The serviceability sub-system of dm Server allows you to gather and view data and information that you can then use to diagnose problems and failures. Serviceability includes data from:

- Service dumps: Contain a snapshot of all the important state from the running dm Server instance when an internal failure or thread deadlock is detected.
  You configure service dumps for SpringSource dm Server using the com.springsource.medic.properties file in the `SERVER_HOME/config` directory. This file also includes some additional logging configuration.

- Event logs and server/application logging (previously called tracing): Logging support in dm Server is based on Logback. This means that you now have complete control over the format of log output and have the complete range of Logback's appenders available for your use.

  You configure logging for SpringSource dm Server using the serviceability.xml file in the `SERVER_HOME/config` directory. This file is essentially the Logback `logback.xml` (or `logback-test.xml`) configuration file but renamed for dm Server.

For additional conceptual information about the serviceability subsystem, see Chapter 7, *Serviceability*.

The com.springsource.medic.properties File

The `SERVER_HOME/config/com.springsource.medic.properties` file configures dm Server service dumps and whether you want to capture `System.out` and `System.err` output to your application's trace file.

The service dump support provides an in-memory buffer of log output. Whenever a dump is triggered this in-memory buffer is written out as part of the dump.

The following table describes the properties you can include in the `SERVER_HOME/config/com.springsource.medic.properties` file. This file configures serviceability properties that dm Server includes in addition to those supplied by the Logback, configured in the `serviceability.xml` file.

Table 9.5. Serviceability Properties

Property	Description
`dump.root.directory`	Specifies the directory to which dm Server writes the service dumps. The directory name is relative to `SERVER_HOME`.
`log.wrapSysOut`	Specifies whether you want to capture `System.out` output from your applications to the application trace file. The output is logged by dm Server's root logger, which

Property	Description
	captures `INFO` level and above. Valid values for this property are `true` to capture `System.out` output, or `false` to disable the capture. For more information, see System.out and System.err.
`log.wrapSysErr`	Specifies whether you want to capture `System.err` output from your applications to the application trace file. The output is logged by dm Server's root logger, which captures `INFO` level and above. Valid values for this property are `true` to capture `System.err` output, or `false` to disable the capture. For more information, see System.out and System.err.
`log.dump.level`	Specifies the log-level of the entries that are captured in the in-memory buffer. Valid values of this property are the same as the log-levels offered by Logback: TRACE, DEBUG, INFO, WARN and ERROR. For more details about these levels, see Logback Architecture.
`log.dump.bufferSize`	Specifies the number of entries will be held in the in-memory buffer. Once the buffer is full, it wraps so that oldest entries start to be overwritten by newer entries; in other words, the buffer is circular.
`log.dump.pattern`	Specifies the formatting of the entries when they're written out as part of the service dump. Use the same pattern layout as for Logback logs; see Layouts in the Logback documentation.

The following sample `com.springsource.medic.properties` is from a freshly-installed dm Server:

```
dump.root.directory=serviceability/dump
log.wrapSysOut=true
log.wrapSysErr=true
log.dump.level=DEBUG
```

```
log.dump.bufferSize=10000
log.dump.pattern=[%d{yyyy-MM-dd HH:mm:ss.SSS}] %-28.28thread %-64.64logger{64} %X{medic.eventCode} %msg %ex%n
```

The serviceability.xml File

Logging support in dm Server is based on Logback, which is a successor of the log4j project. The Logback logging framework is faster, more reliable, and easier to use than log4j and other logging systems.

You configure logging for SpringSource dm Server using the `SERVER_HOME/config/serviceability.xml` file. This file is the standard Logback `logback.xml` or `logback-test.xml` configuration file, but renamed for dm Server due to internal requirements.

The following listing shows the default `serviceability.xml` file in a freshly-installed dm Server; see the text after the listing for a brief overview of the file:

```
<configuration>

    <appender name="SIFTED_LOG_FILE" class="ch.qos.logback.classic.sift.SiftingAppender">
        <discriminator>
            <Key>applicationName</Key>
            <DefaultValue>dm-server</DefaultValue>
        </discriminator>
        <sift>
            <appender name="${applicationName}_LOG_FILE" class="ch.qos.logback.core.rolling.RollingFileAppender">
                <file>serviceability/logs/${applicationName}/log.log</file>
                <rollingPolicy class="ch.qos.logback.core.rolling.FixedWindowRollingPolicy">
                    <FileNamePattern>serviceability/logs/${applicationName}/log_%i.log</FileNamePattern>
                        <MinIndex>1</MinIndex>
                        <MaxIndex>4</MaxIndex>
                </rollingPolicy>
                <triggeringPolicy class="ch.qos.logback.core.rolling.SizeBasedTriggeringPolicy">
                  <MaxFileSize>10MB</MaxFileSize>
                </triggeringPolicy>
                <layout class="ch.qos.logback.classic.PatternLayout">
                    <Pattern>[%d{yyyy-MM-dd HH:mm:ss.SSS}] %-28.28thread %-64.64logger{64} %X{medic.eventCode} %msg %ex%n</Patte
                </layout>
            </appender>
        </sift>
    </appender>

    <appender name="LOG_FILE" class="ch.qos.logback.core.rolling.RollingFileAppender">
        <file>serviceability/logs/log.log</file>
        <rollingPolicy class="ch.qos.logback.core.rolling.FixedWindowRollingPolicy">
            <FileNamePattern>serviceability/logs/log_%i.log</FileNamePattern>
            <MinIndex>1</MinIndex>
            <MaxIndex>4</MaxIndex>
        </rollingPolicy>
        <triggeringPolicy class="ch.qos.logback.core.rolling.SizeBasedTriggeringPolicy">
            <MaxFileSize>10MB</MaxFileSize>
        </triggeringPolicy>
        <layout class="ch.qos.logback.classic.PatternLayout">
            <Pattern>[%d{yyyy-MM-dd HH:mm:ss.SSS}] %-28.28thread %-64.64logger{64} %X{medic.eventCode} %msg %ex%n</Pattern>
        </layout>
    </appender>

    <appender name="EVENT_LOG_STDOUT" class="com.springsource.osgi.medic.log.logback.ReroutingAwareConsoleAppender">
        <layout class="ch.qos.logback.classic.PatternLayout">
             <Pattern>[%d{yyyy-MM-dd HH:mm:ss.SSS}] %-28.28thread &lt;%X{medic.eventCode}&gt; %msg %ex%n</Pattern>
        </layout>
    </appender>

    <appender name="EVENT_LOG_FILE" class="ch.qos.logback.core.rolling.RollingFileAppender">
        <file>serviceability/eventlogs/eventlog.log</file>
        <rollingPolicy class="ch.qos.logback.core.rolling.FixedWindowRollingPolicy">
            <FileNamePattern>serviceability/eventlogs/eventlog_%i.log</FileNamePattern>
            <MinIndex>1</MinIndex>
            <MaxIndex>4</MaxIndex>
        </rollingPolicy>
        <triggeringPolicy class="ch.qos.logback.core.rolling.SizeBasedTriggeringPolicy">
            <MaxFileSize>10MB</MaxFileSize>
        </triggeringPolicy>
        <layout class="ch.qos.logback.classic.PatternLayout">
            <Pattern>[%d{yyyy-MM-dd HH:mm:ss.SSS}] %-28.28thread &lt;%X{medic.eventCode}&gt; %msg %ex%n</Pattern>
```

```
        </layout>
    </appender>

    <logger level="INFO" additivity="false" name="com.springsource.osgi.medic.eventlog.localized">
        <appender-ref ref="EVENT_LOG_STDOUT" />
        <appender-ref ref="EVENT_LOG_FILE" />
    </logger>

    <logger level="INFO" additivity="false" name="com.springsource.osgi.medic.eventlog.default">
        <appender-ref ref="SIFTED_LOG_FILE" />
        <appender-ref ref="LOG_FILE" />
    </logger>

    <root level="WARN">
        <appender-ref ref="SIFTED_LOG_FILE" />
        <appender-ref ref="LOG_FILE" />
    </root>

</configuration>
```

Logback allows dm Server to use logger, appender, and layout objects to log messages according to message type and level and to format these messages and define where they are written. The default `serviceability.xml` file shown above includes four appenders and three loggers (two user and one root.)

The main information to get from this file is that dm Server writes log messages to four different locations that map to the four appenders:

- The `SIFTED_LOG_FILE` appender logs both global and application-specific messages to the `SERVER_HOME/serviceability/logs/`*`applicationName`*`/log.log` file, where *`applicationName`* refers to the name of the application. The log messages for the dm Server itself are logged to the `SERVE_HOME/serviceability/logs/dm-server/log.log` file. Because this appender creates different log files for each application, it is called a *sifting appender*.

  When dm Server creates the first log file, it calls it `log.log`; however, when this file reaches a size of 10MB, dm Server creates a new log file called `log_1.log`, and so on up to 4. At that point, the cycle starts again and dm Server overwrites the existing `log.log`. This is called its *rolling policy*.

  The `<Pattern>` element defines the format of each log message; messages include the timestamp, the thread that generated the log message, the context-specific event code, and a stack trace of the exception, if any. For example:

  ```
  [2008-05-15 09:09:46.940] server-dm-2
  org.apache.coyote.http11.Http11Protocol I Initializing Coyote
  HTTP/1.1 on http-48080
  ```

- The `LOG_FILE` appender is very similar to the first one, but it logs *all* log messages to the `SERVER_HOME/serviceability/log/log.log` file rather than sifting application-specific messages to their own log file. The rolling policy and message format for this appender is similar to that of the `SIFTED_LOG_FILE` appender.

- The `EVENT_LOG_STDOUT` appender does not log messages to a file, but rather to the console window from which you started dm Server. The format of the messages is similar to that of the preceding appenders, although with slightly less information. For example:

  ```
  [2009-08-25 15:04:57.044] server-dm-7 <OF0001I> OSGi telnet
  ```

```
console available on port 2401.
```

- The `EVENT_LOG_FILE` appender logs only important events to the `SERVER_HOME/serviceability/eventlogs/eventlog.log` file, and thus the volume of information is much lower than with the first two appenders. The rolling policy for the event log is the same as with the first two appenders, but the format of the messages is similar to that of the `EVENT_LOG_STDOUT` appender.

The loggers and root logger specify the level of log that is written for each of the referenced appenders.

Typically, the default logging configuration as specified by the `serviceability.xml` file is adequate for all dm Server environments. However, if you want to customize the logging framework even further, you can edit this file as well as the `com.springsource.medic.properties.`. See the logback documentation for detailed information about the architecture and the configuration of Logback.

9.3 Configuring the Embedded Tomcat Servlet Container

SpringSource dm Server embeds an OSGi-enhanced version of the Tomcat Servlet Container in order to provide support for deploying Java EE WARs and *Web Bundles*. You configure the embedded Servlet container using the standard Apache Tomcat configuration. The main difference is that the configuration file is called `tomcat-server.xml` rather than `server.xml`. As with the other dm Server configuration files, the `tomcat-server.xml` file is located in the `$SERVER_HOME/config` directory.

The following listing displays the default configuration distributed with the dm Server; for clarity, the listing does not include the standard Apache License.

```
<?xml version='1.0' encoding='utf-8'?>
<Server port="8005" shutdown="SHUTDOWN">

  <Listener className="org.apache.catalina.core.AprLifecycleListener" SSLEngine="on" />
  <Listener className="org.apache.catalina.core.JasperListener" />
  <Listener className="org.apache.catalina.mbeans.ServerLifecycleListener" />
  <Listener className="org.apache.catalina.mbeans.GlobalResourcesLifecycleListener" />

  <Listener className="com.springsource.server.web.tomcat.ServerLifecycleLoggingListener"/>

  <Service name="Catalina">
    <Connector port="8080" protocol="HTTP/1.1"
               connectionTimeout="20000"
               redirectPort="8443" />

    <Connector port="8443" protocol="HTTP/1.1" SSLEnabled="true"
               maxThreads="150" scheme="https" secure="true"
               clientAuth="false" sslProtocol="TLS"
               keystoreFile="config/keystore"
               keystorePass="changeit"/>

    <Connector port="8009" protocol="AJP/1.3" redirectPort="8443" />

    <Engine name="Catalina" defaultHost="localhost">

      <Realm className="org.apache.catalina.realm.JAASRealm" appName="dm-kernel"
             userClassNames="com.springsource.kernel.authentication.User"
             roleClassNames="com.springsource.kernel.authentication.Role"/>
```

```
      <Host name="localhost"  appBase="webapps"
            unpackWARs="true" autoDeploy="true"
            xmlValidation="false" xmlNamespaceAware="false">

        <Valve className="org.apache.catalina.valves.AccessLogValve"
               directory="serviceability/logs/access"
               prefix="localhost_access_log." suffix=".txt" pattern="common"
               resolveHosts="false"/>

        <Valve className="com.springsource.server.web.tomcat.ApplicationNameTrackingValve"/>
      </Host>
    </Engine>
  </Service>
</Server>
```

Description of the Default Apache Tomcat Configuration

The following bullets describe the main elements and attributes in the default `tomcat-server.xml` file; for details about updating this file to further configure the embedded Apache Tomcat server, see the Apache Tomcat Configuration Reference.

Relative paths

If the configured path to a directory or file does not represent an absolute path, dm Server typically interprets it as a path relative to the `SERVER_HOME` directory.

- The root element of the `tomcat-server.xml` file is `<Server>`. The attributes of this element represent the characteristics of the entire embedded Tomcat servlet container. The `shutdown` attribute specifies the command string that the shutdown port number receives via a TCP/IP connection in order to shut down the servlet container. The `port` attribute specifies the TCP/IP port number that listens for a shutdown message.

- The `<Listener>` XML elements specify the list of lifecycle listeners that monitor and manage the embedded Tomcat servlet container. Each listener class is a Java Management Extensions (JMX) MBean that listens to a specific component of the servlet container and has been programmed to do something at certain lifecycle events of the component, such as before starting up, after stopping, and so on.

  The first four `<Listener>` elements configure standard Tomcat lifecycle listeners. The listener implemented by the `com.springsource.server.web.tomcat.ServerLifecycleLoggingListener` class is specific to SpringSource dm Server and manages server lifecycle logging.

- The `<Service>` XML element groups together one or more connectors and a single engine. Connectors define a transport mechanism, such as HTTP, that clients use to to send and receive messages to and from the associated service. There are many transports that a client can use, which is why a `<Service>` element can have many `<Connector>` elements. The engine then defines how these requests and responses that the connector receives and sends are in turn handled by the servlet container; you can defined only a single `<Engine>` element for any given `<Service>` element.

  The sample `tomcat-server.xml` file above includes three `<Connector>` elements: one for the HTTP transport, one for the HTTPS transport, and one for the AJP transport. The file

also includes a single `<Engine>` element, as required.

- The first connector listens for HTTP requests at the `8080` TCP/IP port. The connector, after accepting a connection from a client, waits for a maximum of 20000 milliseconds for a request URI; if it does not receive one from the client by then, the connector times out. If this connector receives a request from the client that requires the SSL transport, the servlet container automatically redirects the request to port `8443`.

- The second connector is for HTTPS requests. The TCP/IP port that users specify as the secure connection port is `8443`. Be sure that you set the value of the `redirectPort` attribute of your non-SSL connectors to this value to ensure that users that require a secure connection are redirected to the secure port, even if they initially start at the non-secure port. The `SSLEnabled` attribute specifies that SSL is enabled for this connector. The `secure` attribute ensures that a call to `request.isSecure()` from the connecting client always returns `true`. The `scheme` attribute ensures that a call to `request.getScheme()` from the connecting client always returns `https` when clients use this connector.

  The `maxThreads` attribute specifies that the servlet container creates a maximum of 150 request processing threads, which determines the maximum number of simultaneous requests that can be handled. The `clientAuth` attribute specifies that the servlet container does not require a certificate chain unless the client requests a resource protected by a security constraint that uses CLIENT-CERT authentication.

  The `keystoreFile` attribute specifies the name of the file that contains the servlet container's private key and public certificate used in the SSL handshake, encryption, and decryption. You use an alias and password to access this information. In the example, this file is `SERVER_HOME/config/keystore`. The `keystorePass` attributes specify the password used to access the keystore.

- The third AJP Connector element represents a Connector component that communicates with a web connector via the AJP protocol.

- The engine has a logical name of `Catalina`; this is the name used in all log and error messages so you can easily identify problems. The value of the `defaultHost` attribute refers to the name of a `<Host>` child element of `<Engine>`; this host processes requests directed to host names on this servlet container.

- The `<Realm>` child element of `<Engine>` represents a database of users, passwords, and mapped roles used for authentication in this service. SpringSource dm Server uses an implementation of the Tomcat 6 Realm interface that authenticates users through the Java Authentication and Authorization Service (JAAS) framework which is provided as part of the standard J2SE API.

  With the JAASRealm, you can combine practically any conceivable security realm with Tomcat's container managed authentication. For details, see Realm Configuration.

- The `<Host>` child element represents a virtual host, which is an association of a network name for a server (such as `www.mycompany.com`) with the particular server on which Catalina is running. The servlet container unpacks Web applications into a directory hierarchy

if they are deployed as WAR files. The `xmlValidation` attribute specifies that the servlet container does not validate XML files when parsing them, or in other words, it accepts invalid XML. The `xmlNamespaceAware` attribute specifies that the servlet container does not take namespaces into account when reading XML files.

- Finally, the `org.apache.catalina.valves.AccessLogValve` valve creates log files in the same format as those created by standard web servers. The servlet container creates the log files in the `SERVER_HOME/serviceability/logs/access` directory. The log files are prefixed with the string `localhost_access_log.`, have a suffix of `.txt`, use a standard format for identifying what should be logged, and do not include DNS lookups of the IP address of the remote host.

Connector Configuration

The SpringSource dm Server supports the configuration of any connector supported by Apache Tomcat. See the default configuration above for syntax examples, and for further details on the configuration properties supported for various `<Connector>` implementations, consult the official Tomcat HTTP Connector documentation.

Configuring SSL for Tomcat

The SpringSource dm Server distribution includes a preconfigured `SERVER_HOME/config/keystore` file that contains a single self-signed SSL Certificate. The password for this `keystore` file is `changeit`. This `keystore` file is intended for testing purposes only. For detailed instructions on how to configure Tomcat's SSL support, consult the official Tomcat SSL Configuration HOW-TO.

Cluster Configuration

SpringSource dm Server supports standard Apache Tomcat cluster configuration. By default, clustering of the embedded servlet container is disabled, and the default configuration does not include any clustering information. See Tomcat Clustering/Session Replication HOW-TO for detailed information about enabling and configuring clustering.

9.4 Configuring the Local Provisioning Repository

You configure the locations that SpringSource dm Server includes in its provisioning repository by editing the `com.springsource.repository.properties` file in the `$SERVER_HOME/config` directory.

When you specify a property in the file, use the format `repository-name.property=value`, where:

- `repository-name` refers to the name of the local repository.
- `property` refers to the name of a particular property.
- `value` refers to the value of the property.

For example, `ext.type=external` specifies that the `type` property of the repository with name `ext` is `external`.

For each specific repository, you configure a number of properties, such as its type (external, watched, or remote) and its searchpath, watched directory, or URI that specifies the actual location of the artifacts (OSGi bundles, libraries, PARs, plans, or configuration files.) The particular properties that configure these options are listed in the table after the example.

The `chain` property specifies the order in which SpringSource dm Server searches the searchpaths when it looks for dependencies; the first path listed specifies the first actual directory that SpringSource dm Server searches, until the last listed path. The `chain` property uses the names of the searchpaths as specified in the individual repository properties; for example, in the property `ext.type=external`, the name of the repository is `ext`.

The default repository configuration for a newly installed SpringSource dm Server is as follows:

```
ext.type=external
ext.searchPattern=repository/ext/{artifact}

usr.type=watched
usr.watchDirectory=repository/usr

chain=ext,usr
```

The default configuration shown above has two searchpaths corresponding to the two default sub-directories of the `SERVER_HOME/repository` directory created when you first install dm Server: `ext` and `usr`. SpringSource dm Server searches each of these searchpaths when locating entries for inclusion in the repository.

The `chain` property shows the order in which SpringSource dm Server searches the searchpaths: first `ext` and then `usr`.

The following table lists all the available properties that you can use to describe a named path and the repository search chain in the `com.springsource.repository.properties` file.

Table 9.6. Repository Properties in repository.properties

Property	**Description**
`repository-name``.type`	Specifies the type of path. You can set this property to one of the following three valid values: • `external`: Specifies that this path points to a number of directories that satisfy a

Property	Description
	given search pattern and are local to the current SpringSource dm Server instance. Use the `searchPattern` property to specify the directory search pattern. • `watched`: Specifies that this path points to a single directory, local to the current SpringSource dm Server instance. SpringSource dm Server regularly scans watched repositories so it automatically picks up any changes to the artifacts in the directory at runtime. Use the `watchDirectory` property to specify the watched directory and the `watchInterval` property to specify how often dm Server checks the directory. • `remote`: Specifies that the path points to a remotely-hosted repository, hosted by a remote instance of SpringSource dm Server. Use the `uri` property to specify the full URI of the remote repository. You can also specify the optional `indexRefreshInterval` property. See Watched or External Repository? for additional information about when to configure watched or external repositories for your particular environment.
repository-name`.searchPattern`	Specifies the pattern that an external repository uses when deciding which local directories it should search when identifying artifacts. Use this property together with *repository-name*`.type=external`. See Search Paths: Additional Information for detailed information about specifying a search pattern.
repository-name`.watchDirectory`	Specifies the single directory of a watched repository. You can specify either an absolute or relative pathname for the directory. If you specify a relative pathname, it is relative to the root of the dm Server installation (`SERVER_HOME`). Use this property together

Property	Description
	with *repository-name*.type=watched.
repository-name.watchInterval	Specifies the interval in seconds between checks of a watched directory by a watched repository. This property is optional, if it is not specified the default interval of 5 seconds is used. Use this property together with *repository-name*.type=watched.
repository-name.uri	Specifies the URI of the hosted repository to which a remote repository connects. The value of this property takes the following format: http://*host*:*port*/com.springsource.server.repo where: • *host* refers to the computer on which the remote dm Server instance hosts the remote repository. • *port* refers to Tomcat listen port of the remote dm Server instance which hosts the remote repository. • *remote-repository-name* refers to the name of the remote repository, as specified in the hostedRepository.properties file of the remote dm Server instance. Use this property together with *repository-name*.type=remote.
repository-name.indexRefreshInterval	Specifies the interval in seconds between checks by a remote repository that its local copy of the hosted repository index is up-to-date (a remote repository acts as a proxy for a hosted repository and thus it holds a local copy of the hosted repository's index). This property is optional, if it is not specified the default interval of 5 seconds is used. Use this property together with *repository-name*.type=remote.

Property	Description

Should I Configure a Watched or External Repository?

The main difference between a watched and an external repository is that SpringSource dm Server regularly scans watched directories and automatically picks up any changed artifacts, while dm Server scans external directories only at startup, and then only if there is no cached index available. This means that dm Server always performs a scan of an external repository when you start the server with the `-clean` (as this deletes the index) and only scans during a normal startup if the index isn't there because, for example, this is the first time you start the server.

There is a performance cost associated with using a watched repository due to dm Server using resources to scan the directory at the configured interval. The cost is small if the watched repository contains just a few artifacts; however, the performance cost increases as the number of artifacts increases.

For this reason, SpringSource recommends that you put most of your dependencies in external repositories, even when in development mode. If you make any changes to the artifacts in the external repositories, remember to restart dm Server with the `-clean` option so that the server picks up any changes. Use watched directories for artifacts that you are prototyping, actively updating, or when adding new dependencies so that dm Server quickly and easily picks them up. To increase performance even during development, however, you can use an external repository for most of your dependencies, in particular the ones that are fairly static.

In production environments, where dependencies should not change, SpringSource recommends that you use *only* external repositories.

Search Paths: Additional Information

The *`repository-name`*`.searchPattern` and *`repository-name`*`.watchDirectory` properties specify search paths for external and watched repositories, respectively, that define a physical location that SpringSource dm Server searches when looking for a library or bundle dependency. If a search path is relative, its location is relative to the root of the installation, in other words, the `SERVER_HOME` directory.

Using Wildcards

Search paths specified with the *`repository-name`*`.searchPattern` property provide support for wildcards. In the entries above, the path segments surrounded by curly braces, for example `{bundle}` and `{library}`, are wildcards entries for a directory with any name. Allowing wildcards to be named in this way is intended to improve the readability of search path configuration.

In addition to supporting the above-described form of wildcards, SpringSource dm Server also

supports Ant-style paths, that is `*` and `**` can be used to represent any directory and any series of directories, respectively. For example, `repository/usr/{bundle}` and `repository/usr/*` are directly equivalent.

A common usage of the `**` wildcard is to allow dependencies stored in a directory structure of varying depth, such as a local Maven repository, to be provisioned by the SpringSource dm Server.

Using System properties

You can use system properties when specifying the values of the *`repository-name`*`.searchPattern`, *`repository-name`*`.watchDirectory`, *`repository-name`*`.watchInterval` *`repository-name`*`.uri`, and *`repository-name`*`.indexRefreshInterval` properties. You reference system properties as `${system.property.name}`; for example, a search path of `${user.home}/repository/bundles` references the `repository/bundles` directory in the user's home directory.

Example repository configurations

The following examples provide sample configuration that could be used for some common use cases.

Add an Ivy cache repository

The following example shows how to add an external repository whose location is actually an Ivy cache.

```
ext.type=external
ext.searchPattern=repository/ext/{artifact}

usr.type=watched
usr.watchDirectory=repository/usr

ivy-repo.type=external
ivy-repo.searchPattern=${user.home}/.ivy2/cache/{org}/{name}/{version}/{bundle}.jar

chain=ext,usr,ivy-repo
```

Add a Maven local repository

The following example shows how to add an external repository whose location is actually a Maven repository.

```
ext.type=external
ext.searchPattern=repository/ext/{artifact}

usr.type=watched
usr.watchDirectory=repository/usr

maven-repo.type=external
maven-repo.searchPattern=${user.home}/.m2/repository/**/{bundle}.jar

chain=ext,usr,maven-repo
```

Add remote and watched repositories

The following example shows the default `com.springsource.repository.properties` file from a freshly-installed dm Server, but then updated to include new remote and watched repositories. Both of these repositories are part of the repository chain.

The remote repository is called `remote-repo`. The URI of the hosted repository from which `remote-repo` gets its artifacts is `http://my-host:8080/com.springsource.server.repository/my-hosted-repo`; this means that there is a dm Server instance running on host `my-host` whose Tomcat server listens at the default port, `8080`, and this server instance hosts a repository called `my-hosted-repo`, configured in the `hostedRepository.properties` file of the remote server instance. The remote repository checks for changes in the hosted repository every 30 seconds.

The watched repository is called `watched-repo` and the directory that holds the artifacts is `repository/watched`, relative to the installation directory of the dm Server instance. The server checks for changes in this watched repository every 5 seconds.

```
ext.type=external
ext.searchPattern=repository/ext/{artifact}

usr.type=watched
usr.watchDirectory=repository/usr

remote-repo.type=remote
remote-repo.uri=http://my-host:8080/com.springsource.server.repository/my-hosted-repo
remote-repo.indexRefreshInterval=30

watched-repo.type=watched
watched-repo.watchedDirectory=repository/watched
watched-repo.watchedInterval=5

chain=ext,usr,remote-repo,watched-repo
```

9.5 Configuring a Hosted Repository

You configure a dm Server instance to host a repository by editing the `SERVER_HOME/config/com.springsource.repository.hosted.properties` file; remote clients can then access the artifacts in this hosted repository and use them locally.

When you specify a property in the file, use the format `repository-name.property=value`, where:

- `repository-name` refers to the name of the hosted repository.
- `property` refers to the name of a particular property.
- `value` refers to the value of the property.

For example, `my-hosted-repo.type=external` specifies that the `type` property of the `my-hosted-repository` repository is `external`.

The following table lists the properties that you can include in the `hostedRepository.properties` file.

Table 9.7. Hosted Repository Properties

Property	Description
repository-name`.type`	Specifies the type of path of the hosted repository. All paths are local to the current dm Server instance. You can set this property to one of the following valid values: • `external`: Specifies that this path points to a number of directories that satisfy a given search pattern. Use the `searchPattern` property to specify the directory search pattern. • `watched`: Specifies that this path points to a single directory. SpringSource dm Server regularly scans watched repositories so it automatically picks up any changes to the artifacts in the directory at runtime. Use the `watchDirectory` property to specify the actual watched directory and the `watchInterval` property to specify how often dm Server checks the directory. See Watched or External Repository? for additional information about when to configure watched or external repositories for your particular environment.
repository-name`.searchPattern`	Specifies the pattern that an external hosted repository uses when deciding which local directories it should search when identifying artifacts. Use this property when `repository-name.type=external`. See Search Paths: Additional Information for detailed information about specifying a search pattern.
repository-name`.watchDirectory`	Specifies the single directory of a watched hosted repository. You can specify either an absolute or relative pathname for the directory. If you specify a relative pathname, it is relative to the root of the dm Server installation (`SERVER_HOME`). Use this

Property	Description
	property when `repository-name.type=watched`.
`repository-name``.watchInterval`	Specifies the interval in seconds between checks of a watched directory by a watched hosted repository. This property is optional. Use this property when `repository-name.type=watched`.

The following sample shows a `com.springsource.repository.hosted.properties` file with a single external repository called `my-hosted-repo` with search pattern `SERVER_HOME/repository/hosted/*`.

```
my-hosted-repo.type=external
my-hosted-repo.searchPattern=repository/hosted/*
```

See Example of watched and remote repositories for details on how a local repository can remotely access the artifacts in this hosted repository.

Appendix A. Event log codes

A.1 Format of the event log codes

Event log codes issued by dm Server have the general syntax <XXnnnnL> where:

XX	is a two-letter code (upper-case) identifying the region of the dm Server code which issued the log message;
nnnn	is a four-digit message number; and
L	is a single-letter (upper-case) code identifying the level of the message.

The two-letter codes are (this list is not complete):

CC	com.springsource.kernel.services.concurrent
DE	com.springsource.kernel.deployer.core the Deployer
HD	com.springsource.kernel.deployer.hot the Hot Deployer
KD	com.springsource.kernel.dm
KE	com.springsource.kernel.core the Kernel
OF	com.springsource.kernel.osgi Osgi Framework
OP	com.springsource.kernel.osgi.provisioning Osgi Provisioning
RP	com.springsource.repository the Repository

The four-digit numbers identify the message text (with placeholders for inserted values). These are not listed here, but can be discovered by examining the files called EventLogMessages.properties, found in the relevant packages.

The single-digit level code is one of:

E	Error level: enabled if level is ERROR.
W	Warning level: enabled if level is WARNING or above.
I	Info level: enabled if level is INFO or above.
D	Debug level: enabled if level is DEBUG or above.
T	Trace level: always enabled.

There are never two messages with the same prefix and number, but with different levels.

Appendix B. Known Issues

This section describes two known issues that you might run into, along with corresponding workarounds.

For the full list of known issues, see the SpringSource Issue Tracker for the dm Server project. The issues are organized by component as well as by release. You can also use the Issue Tracker application to enter a new issue if you cannot find an existing issue that describes the problem you are running into.

B.1 Timeout During Startup Due to Firewall Settings

The dm Server will fail to start correctly if it is prevented from connecting to needed ports by the firewall. Typically this manifests as error `SPPM0003E` . Configuring the firewall to allow the dm Server process to bind to the necessary ports will prevent this error from occurring.

B.2 OutOfMemoryError: PermGen space running on Sun VM

As a result of Sun Java bug 4957990, the SpringSource dm Server may consume more PermGen space than expected when running with the server HotSpot compiler. This problem may be resolved by configuring the `JAVA_OPTS` environment variable to specify an increased `MaxPermSize`, for example `-XX:MaxPermSize=128M`.

Appendix C. Further Reading

SpringSource dm Server Programmer Guide

Spring Framework Reference Guide

Spring Dynamic Modules Reference Guide

The Logback Manual