

# Virgo Web Server User Guide

Rob Harrop  
Paul Kuzan  
Sam Brannen  
Paul Harris  
Christopher Frost  
Ben Hale  
Glyn Normington  
Juliet Shackell  
Steve Powell



**2.1.0.M01**



# Table of Contents

1. Installing Virgo Web Server .....	1
1.1. Prerequisites .....	1
1.2. Installing from the ZIP Download .....	1
1.3. Post-installation steps .....	1
2. Starting and Stopping VWS .....	9
2.1. Starting Virgo Web Server .....	9
2.2. Starting in Clean Mode .....	9
2.3. Starting in Debug Mode .....	10
2.4. Starting with JMX Access Modifications .....	11
2.5. Starting With a Custom Configuration Directory .....	19
2.6. Stopping Virgo Web Server .....	20
3. Overview of the Web Server Kernel and User Region .....	23
3.1. The VWS Kernel .....	23
3.2. The VWS User Region .....	24
4. Equinox Console Extension .....	27
4.1. Using the Equinox Console Extension .....	27
4.2. Equinox Console Extension Command Reference .....	28
5. The Admin Console .....	47
5.1. Invoking the Admin Console .....	47
5.2. Typical Admin Console Use Cases .....	49
6. The Provisioning Repository .....	59
6.1. Overview of the Provisioning Repository .....	59
6.2. Finding and Downloading Bundles from the SpringSource Enterprise Bundle Repository .....	61
6.3. Configuring the repository .....	62
7. Serviceability .....	63
7.1. Event log files .....	63
7.2. Trace (Logging) .....	63
7.3. Service Dumps .....	65
8. Working with Applications .....	67
8.1. Deploying Artifacts .....	67
8.2. Undeploying Artifacts .....	68
9. Configuring VWS .....	71
9.1. Configuring the Virgo Kernel and User Region .....	71
9.2. Configuring Serviceability .....	75
9.3. Configuring the Embedded Tomcat Servlet Container .....	80
9.4. Configuring the Local Provisioning Repository .....	83
9.5. Configuring a Hosted Repository .....	89
A. Event log codes .....	93
A.1. Format of the event log codes .....	93
B. Known Issues .....	95
B.1. Timeout During Startup Due to Firewall Settings .....	95
B.2. OutOfMemoryError: PermGen space running on Sun VM .....	95
C. Further Reading .....	97



# 1. Installing Virgo Web Server

## 1.1 Prerequisites

The Virgo Web Server, or VWS for short, 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

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

```
prompt$ unzip virgo-web-server-2.1.0.M01.zip -d /opt
```

This creates a directory called `virgo-web-server-2.1.0.M01` under `/opt`.

Virgo Web Server requires write access to the installation directory, in this case `/opt/virgo-web-server-2.1.0.M01`. 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 Virgo Web 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 Virgo Web 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 Virgo Web Server installation directory. Note that the Virgo Web Server does not require that such an environment variable has been set. This variable may have any name of your choosing. The Virgo Web Server's documentation assumes that the variable is named `SERVER_HOME`.

## Linux

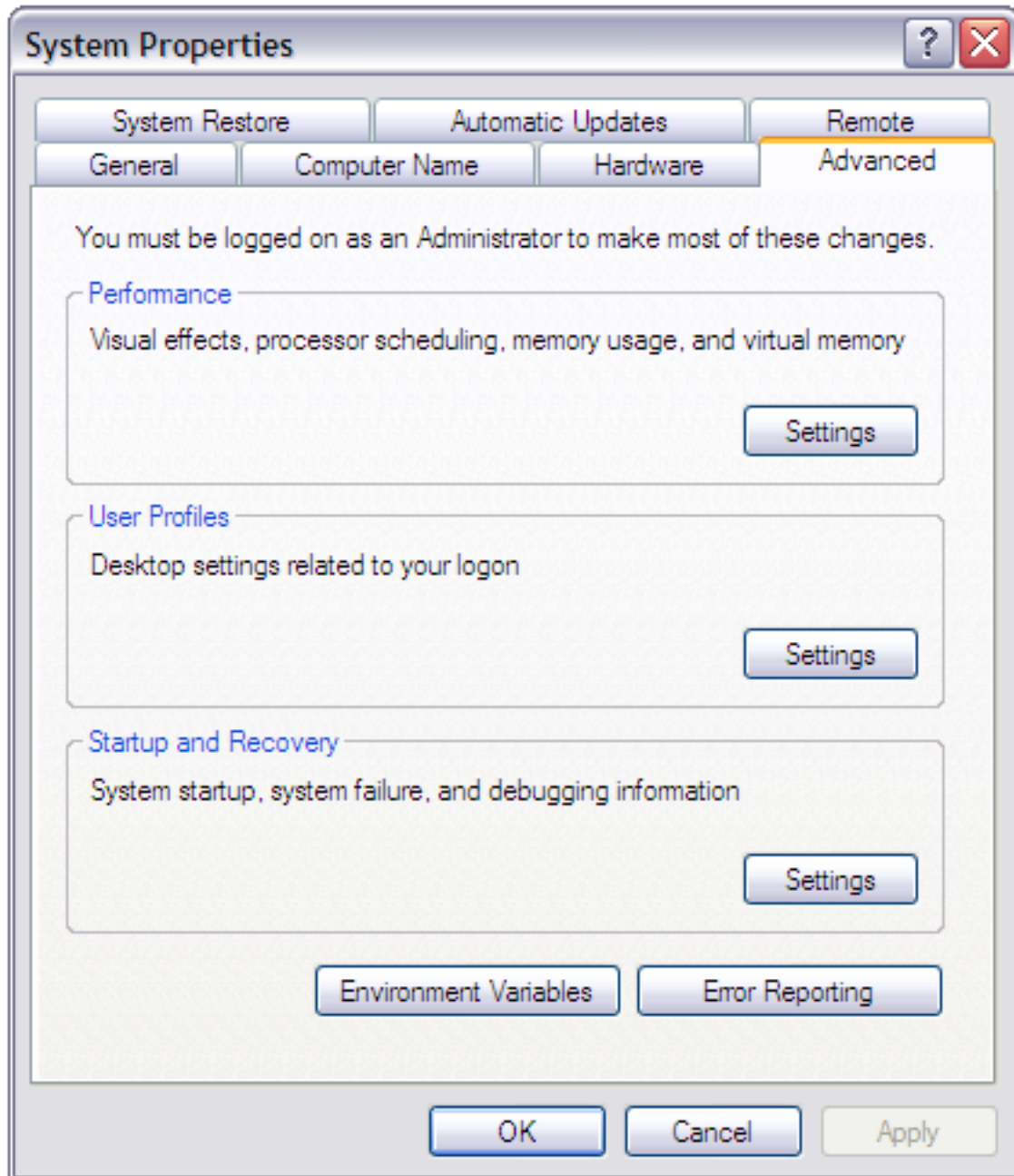
Edit the `.profile` file in your home directory to add the `SERVER_HOME` and `JAVA_HOME` environment variables. For example, if you installed into `/opt`:

```
export SERVER_HOME=/opt/virgo-web-server-2.1.0.M01/  
export JAVA_HOME=/user/java/jdk1.6.0_17  
export PATH=$JAVA_HOME/bin:$PATH
```

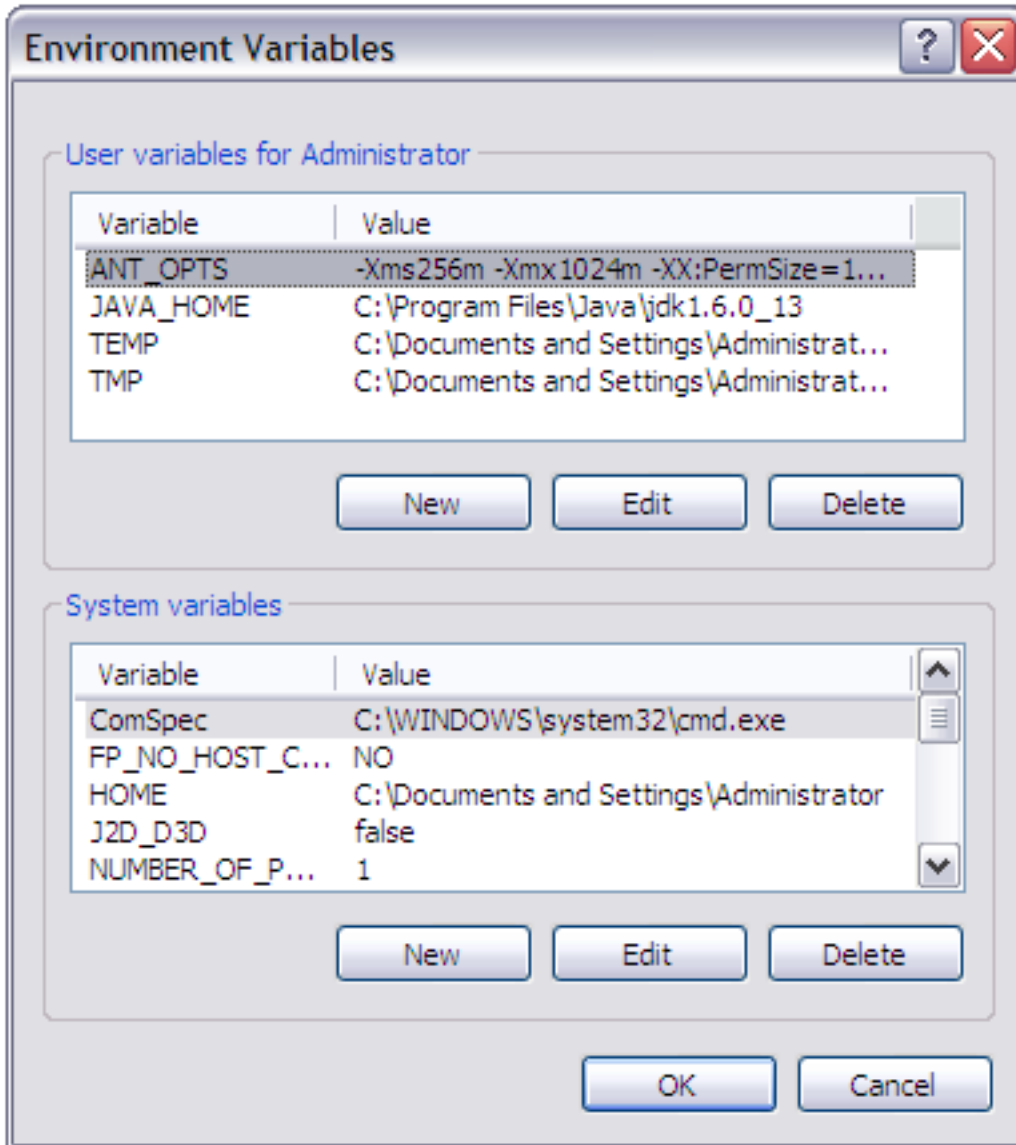
## Microsoft Windows

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.

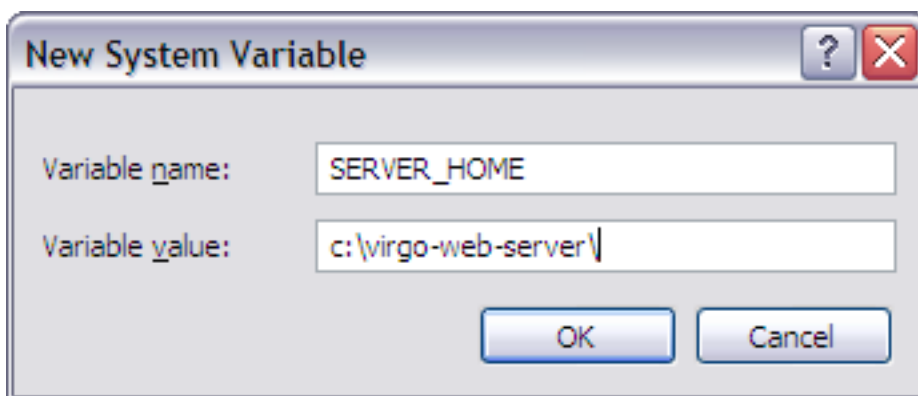
From the Start menu, open the Control Panel and double-click on 'System'.



Click the 'Advanced' tab and select 'Environment Variables'. Next, click the 'New' button in the 'System Variables' section.



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





## Microsoft Windows - Troubleshooting

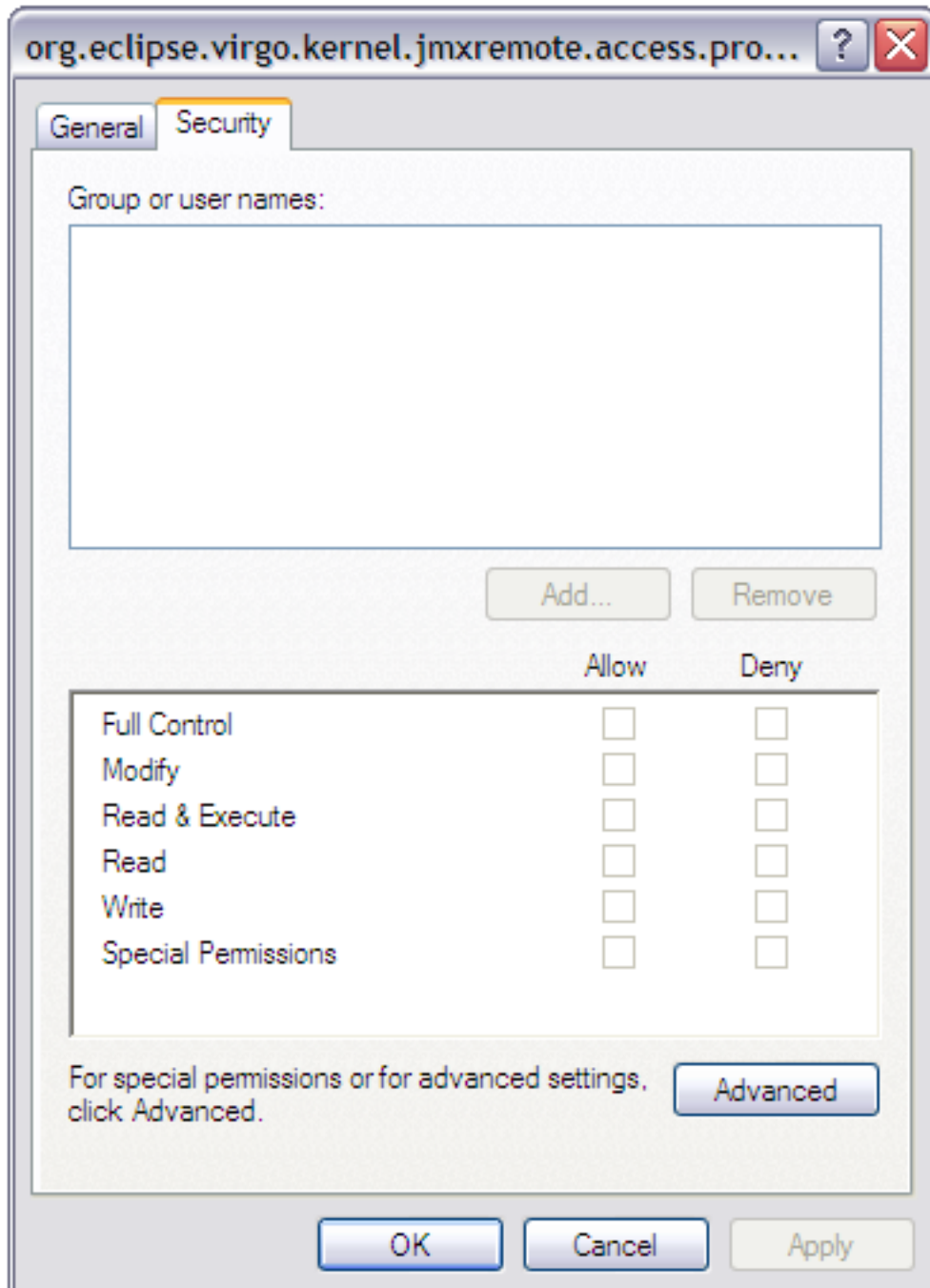
When starting the Virgo Web Server on some variants of Windows you might encounter a problem with file permissions. The error looks like this.

```
C:\dev\virgo-web-server-2.0.1.BUILD-20091208094124>bin\startup.bat
Error: Exception thrown by the agent : java.io.FileNotFoundException: C:\dev\...\config\
org.eclipse.virgo.kernel.jmxremote.access.properties (Access is denied)
```

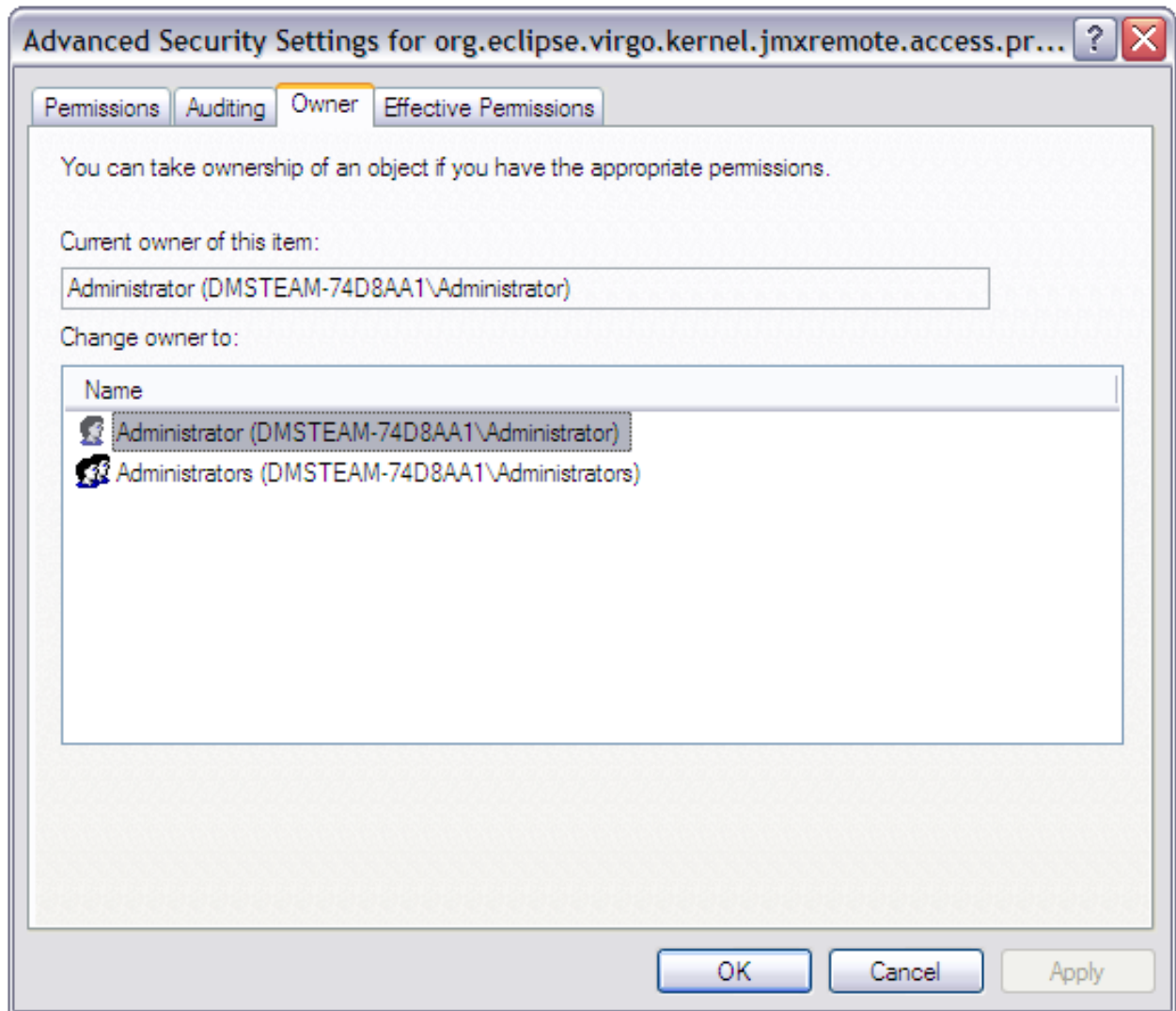
If the VWS starts at this point you can skip this section and carry on. Otherwise, go to the ‘config’ directory of your install in Windows Explorer and you will be able to confirm the problem, an incorrect file ownership.

Name ▲	Size	Type	Owner
keystore	2 KB	File	DMSTEAM-74D8AA1\Administrator
org.eclipse.virgo.kernel.authentication.config	1 KB	CONFIG File	DMSTEAM-74D8AA1\Administrator
org.eclipse.virgo.kernel.jmxremote.access.properties	1 KB	PROPERTIES File	DMSTEAM-74D8AA1\Administrator
org.eclipse.virgo.kernel.properties	1 KB	PROPERTIES File	DMSTEAM-74D8AA1\Administrator
org.eclipse.virgo.kernel.userregion.properties	4 KB	PROPERTIES File	DMSTEAM-74D8AA1\Administrator
org.eclipse.virgo.kernel.users.properties	1 KB	PROPERTIES File	DMSTEAM-74D8AA1\Administrator
org.eclipse.virgo.medic.properties	1 KB	PROPERTIES File	DMSTEAM-74D8AA1\Administrator
org.eclipse.virgo.repository.properties	1 KB	PROPERTIES File	DMSTEAM-74D8AA1\Administrator
serviceability.xml	4 KB	XML Document	DMSTEAM-74D8AA1\Administrator

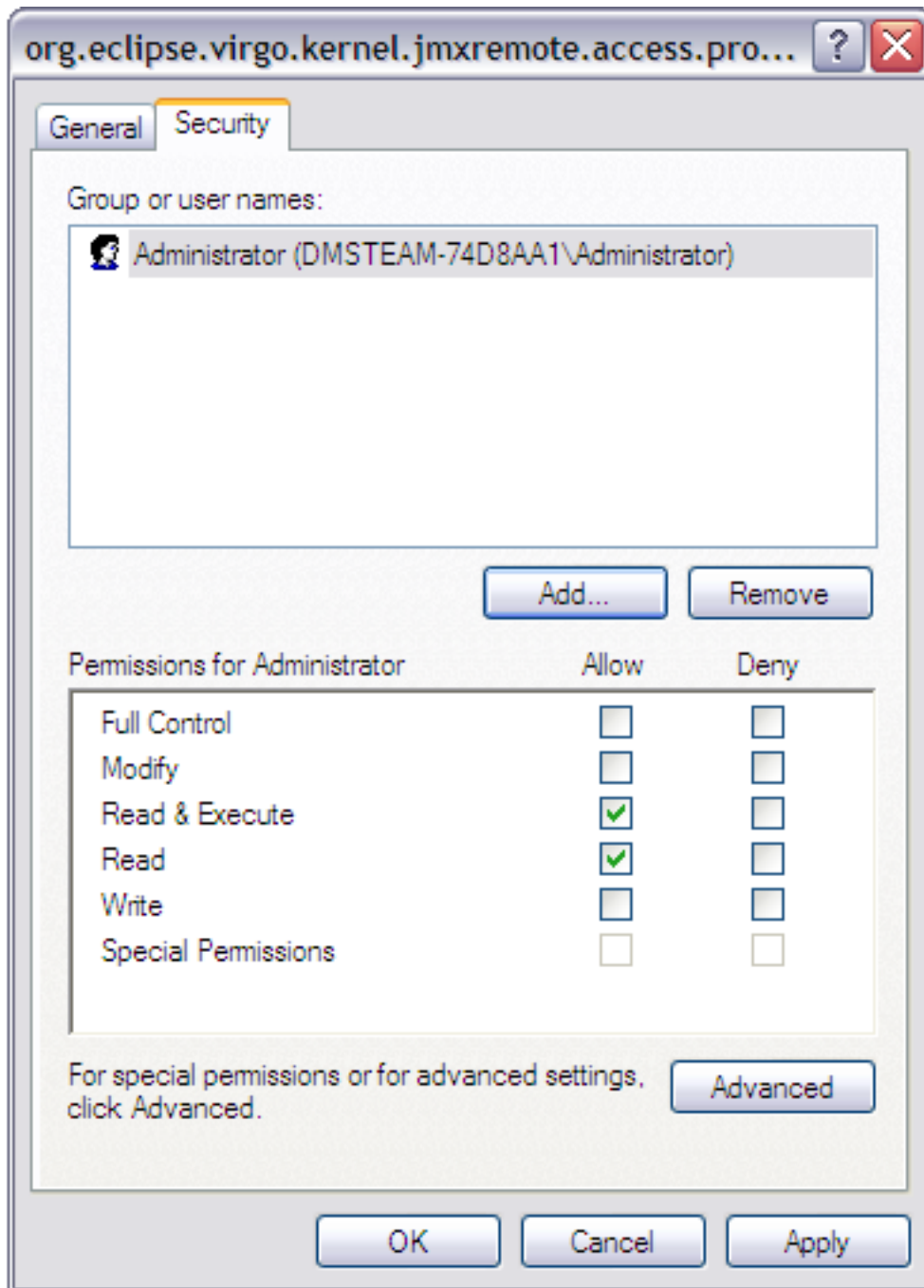
Right click on the ‘org.eclipse.virgo.kernel.jmxremote.access.properties’ file and view its properties, then select the ‘Security’ tab.



Within the security page select the 'Advanced' options. On the 'Owner' tab, choose the owner that you are trying to run the VWS as and select 'Apply'.



Once this is done select 'OK' to return to the 'Security' tab and now add the owner to the list of groups and users that have permission to edit the file.



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

```
C:\dev\virgo-web-server-2.0.1.BUILD-20091208094124>bin\tstartup.bat
[2009-12-08 13:09:09.545] startup-tracker <KE0001I> Kernel starting.
```

## 2. Starting and Stopping Virgo Web Server

### 2.1 Starting Virgo Web Server

To start Virgo Web 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 Virgo Web Server, open a terminal window and run `startup.sh`:

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

Once Virgo Web 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 VWS.

#### Microsoft Windows

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

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

Once Virgo Web 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 VWS.

### 2.2 Starting in Clean Mode

When you start Virgo Web 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 Virgo Web 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 Virgo Web 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 Virgo Web 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 Virgo Web 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 Virgo Web 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 Virgo Web 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 Virgo Web Server always starts with JMX access enabled, allowing you to use a management tool such as JConsole to attach to the Web 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

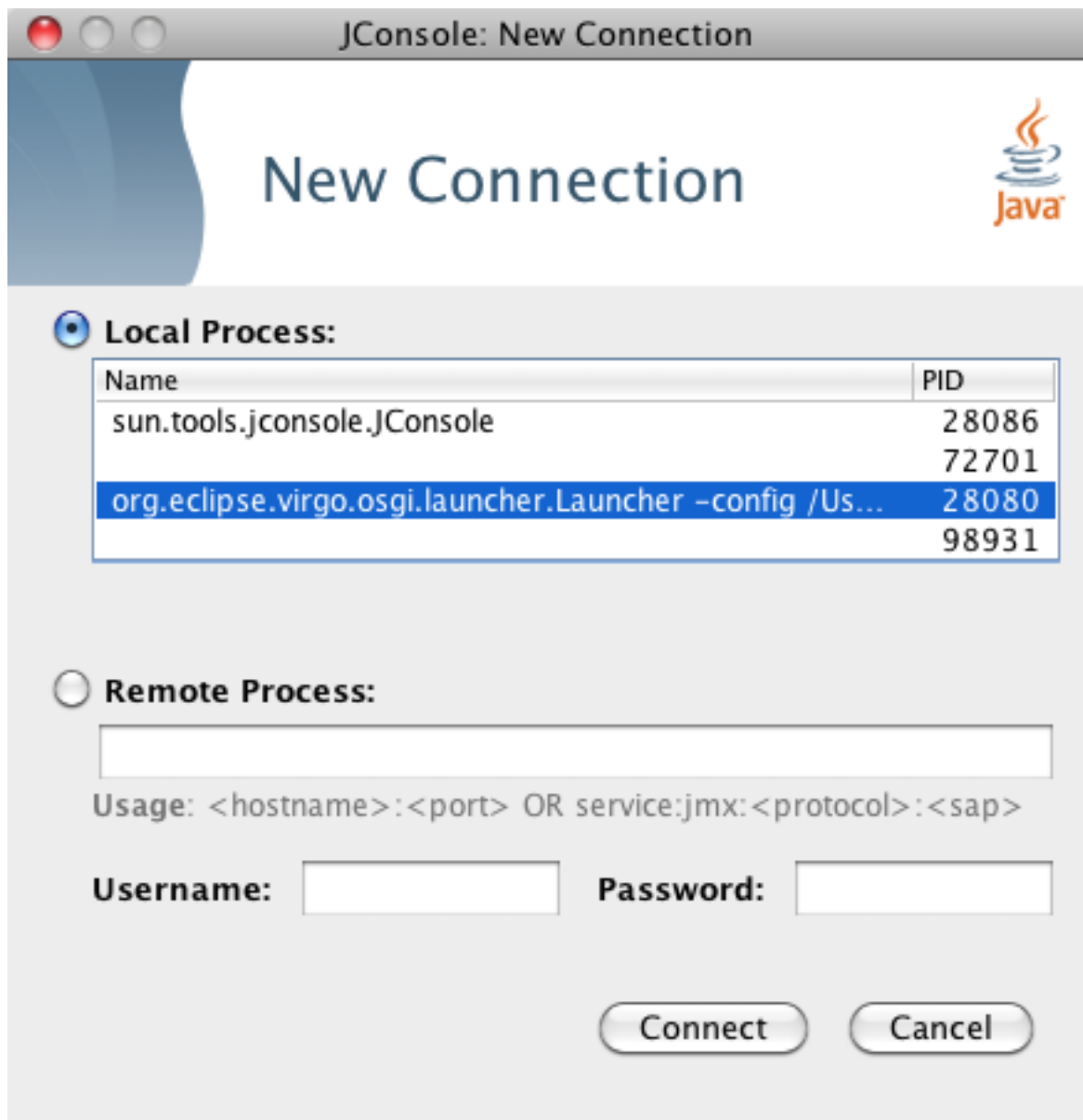
To start Virgo Web Server with default JMX access enabled, run `startup.sh` passing in no arguments:

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

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

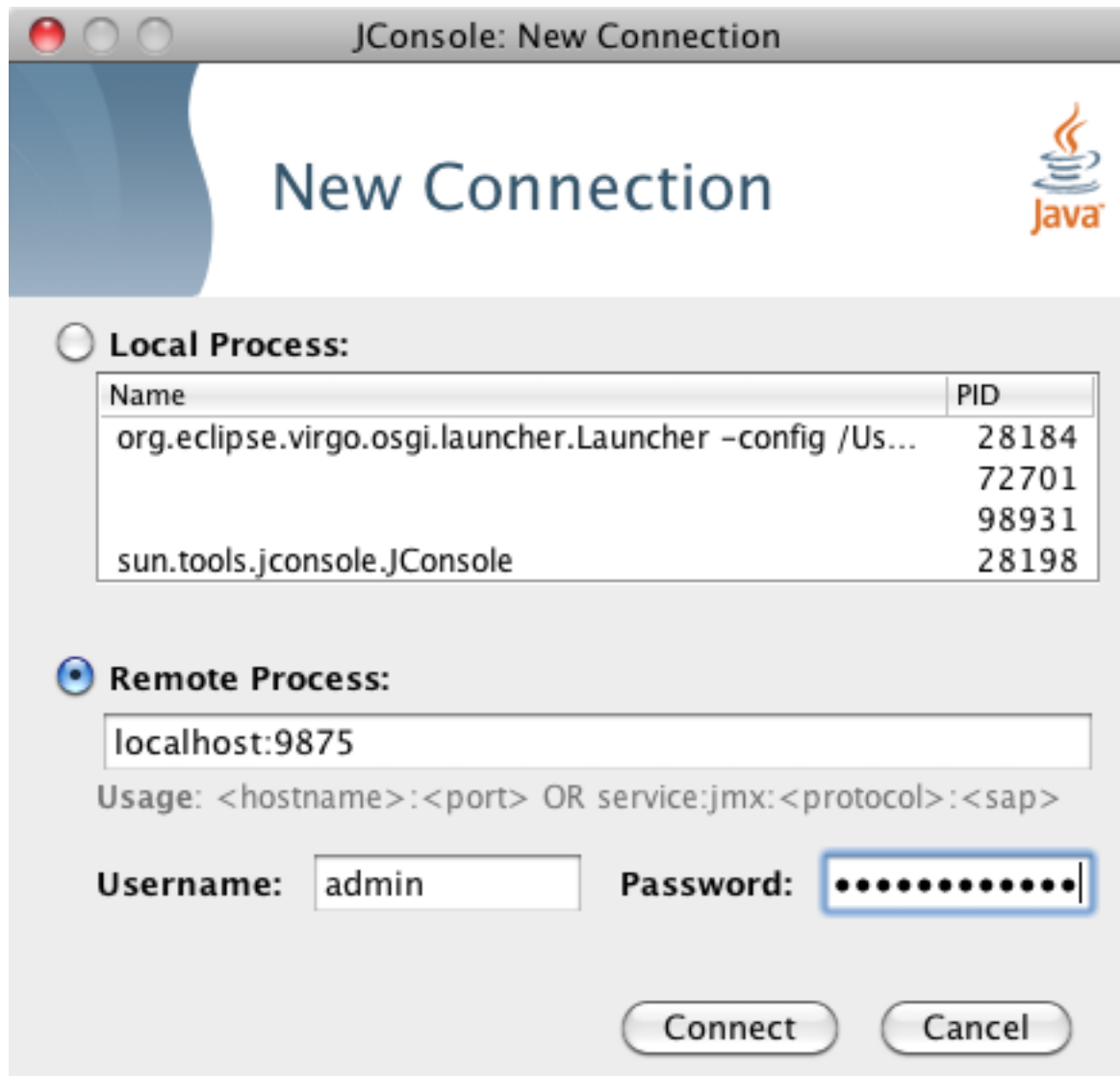
```
prompt$ cd $SERVER_HOME
prompt$ bin/jconsole.sh
```

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).

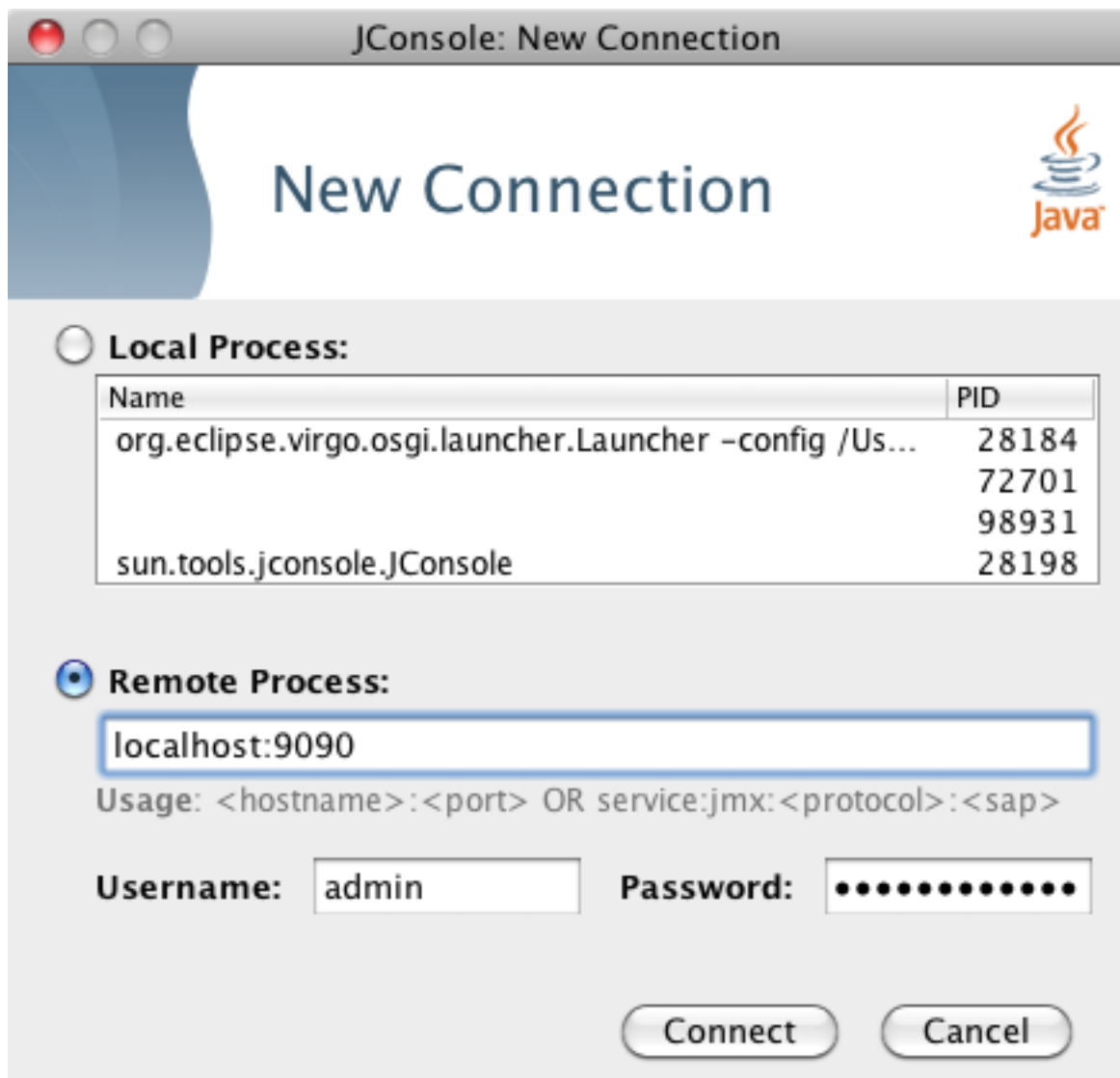




To start with the JMX remote access on a specific port number other than the default 9875, pass this port number in as the value of the `-jmxport` argument:

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

This will start the Virgo Web Server with JMX enabled for remote connections on port 9090.



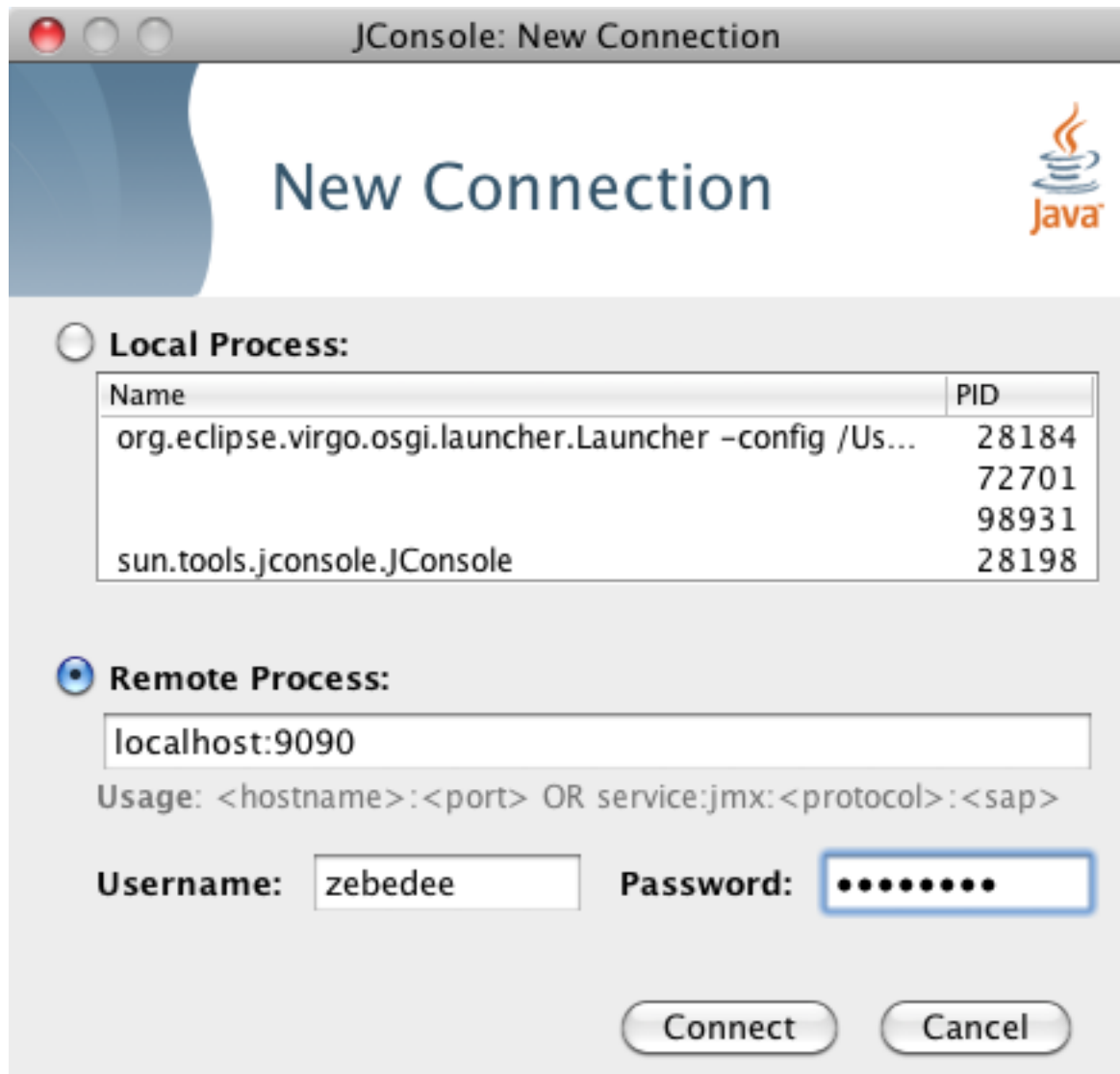
To start the JMX remote access with a custom username and password, update the `$SERVER_HOME/config/org.eclipse.virgo.kernel.users.properties` file. First specify the custom username by changing the value of the `role.admin` property. Then set the password of this new user by adding a new property called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

For example, if you want change the JMX remote access username to `zebedee` with password `florence`, change the file as follows:

```
#####
# User definitions
#####
user.zebedee=florence

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

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 Virgo Web Server with JMX enabled for remote connections using an SSL certificate from `customKeystore` with a password of `customKeystorePassword`.

## Microsoft Windows

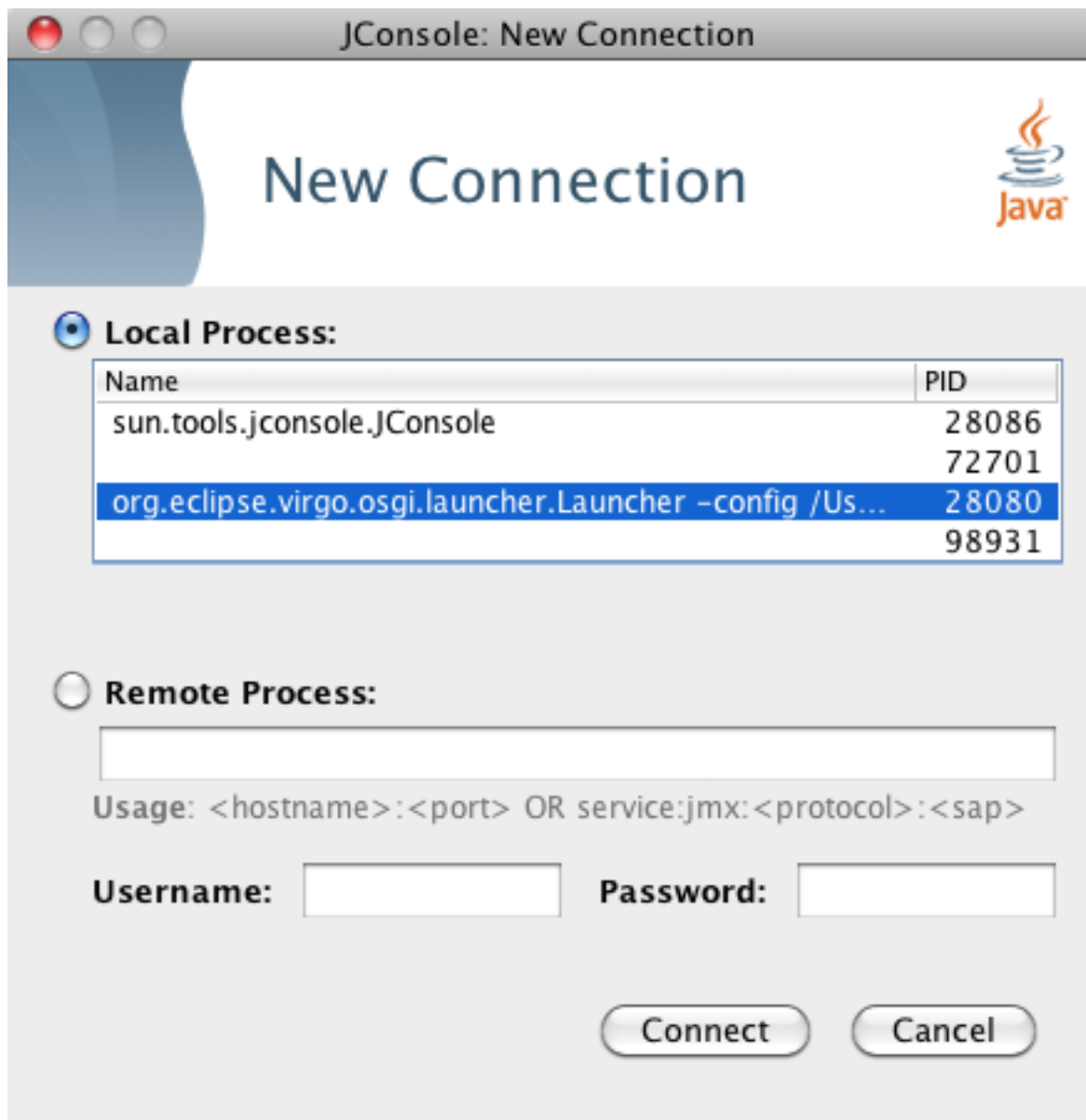
To start Virgo Web Server with default JMX access enabled, run `startup.bat` passing in no arguments:

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

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

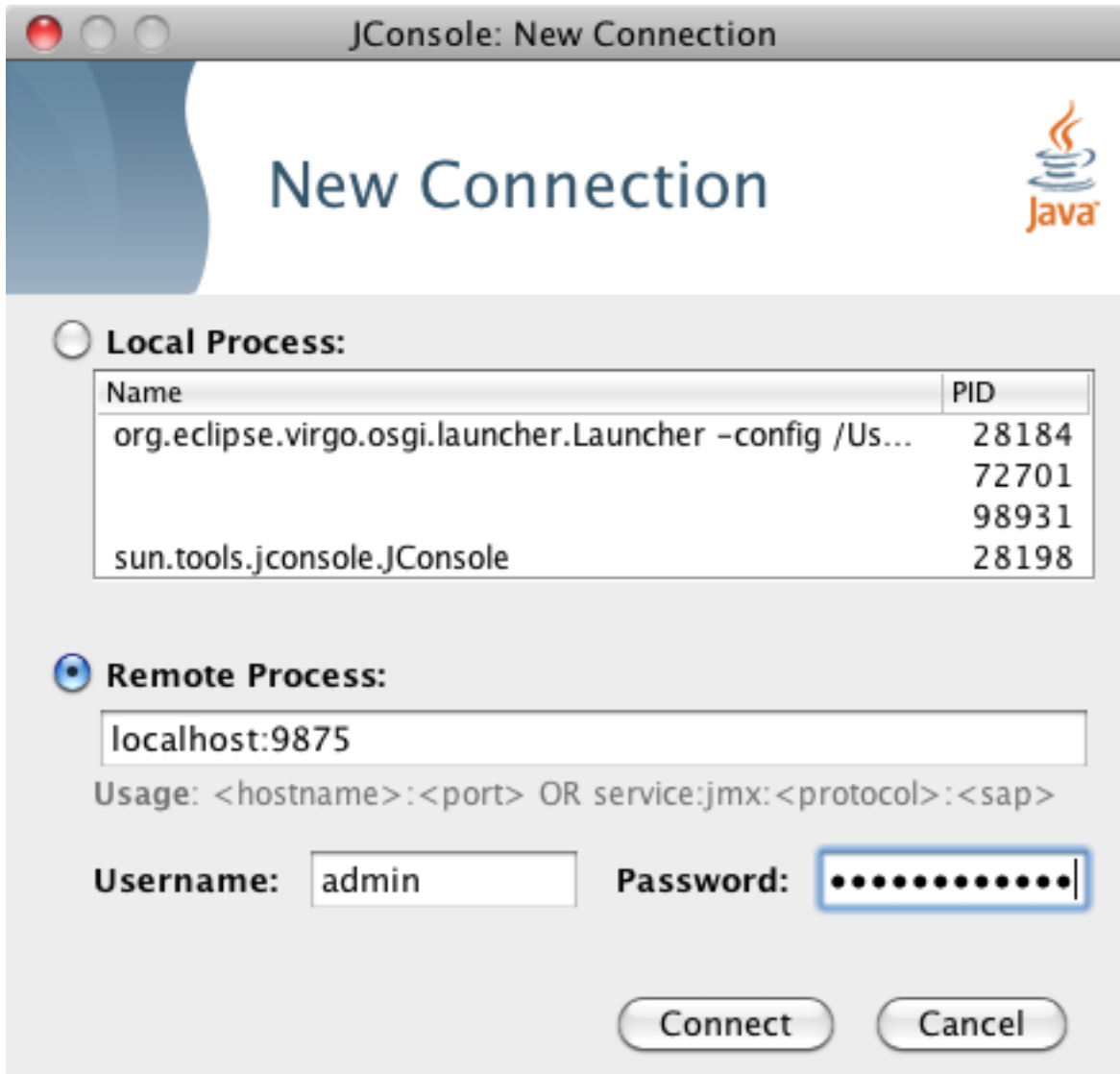
```
prompt> cd %SERVER_HOME%
prompt> bin\jconsole.bat
```

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

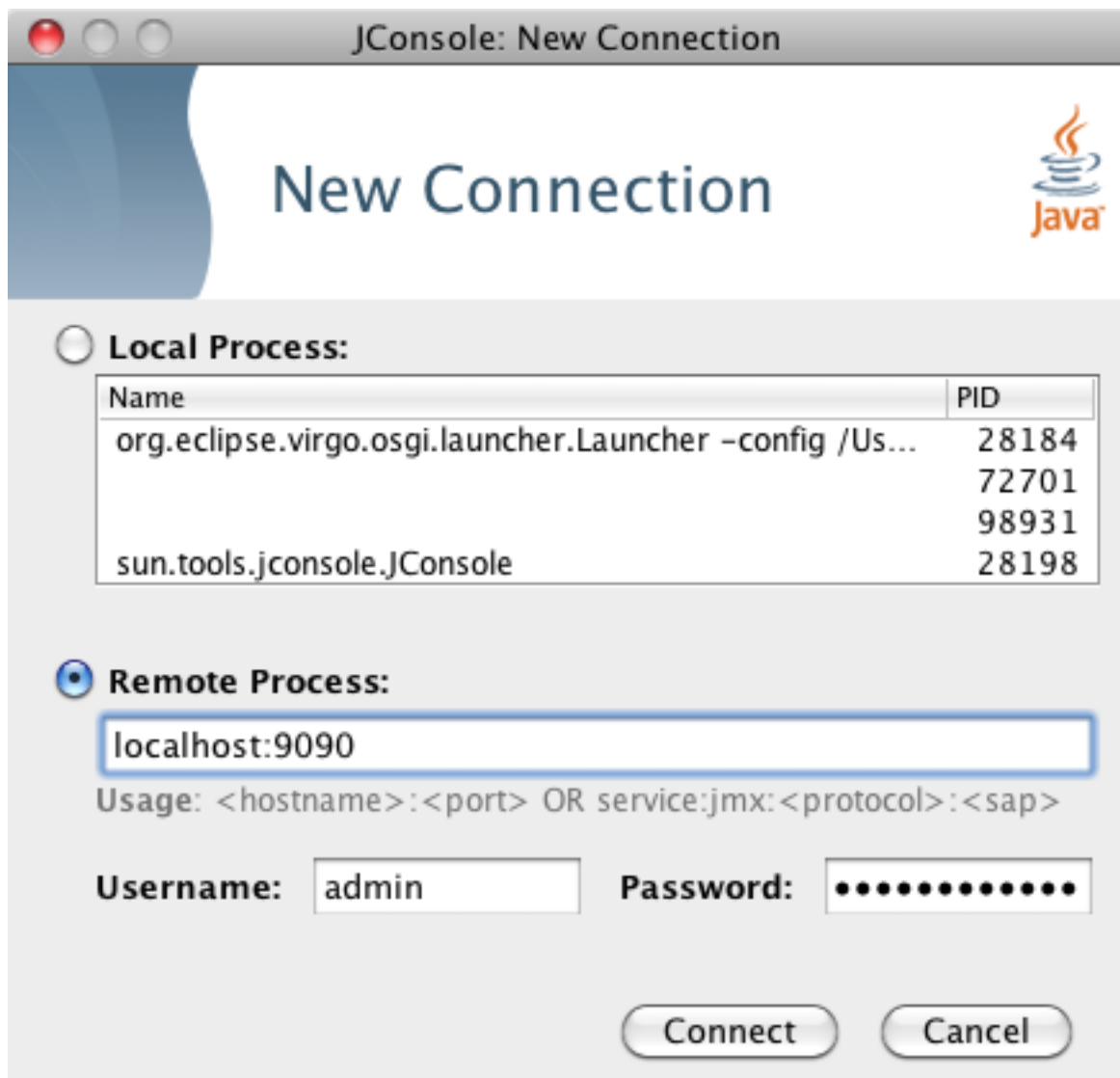
the default username/password (admin/springsource and default secure port of 9875).



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 Virgo Web Server with JMX enabled for remote connections on port 9090.



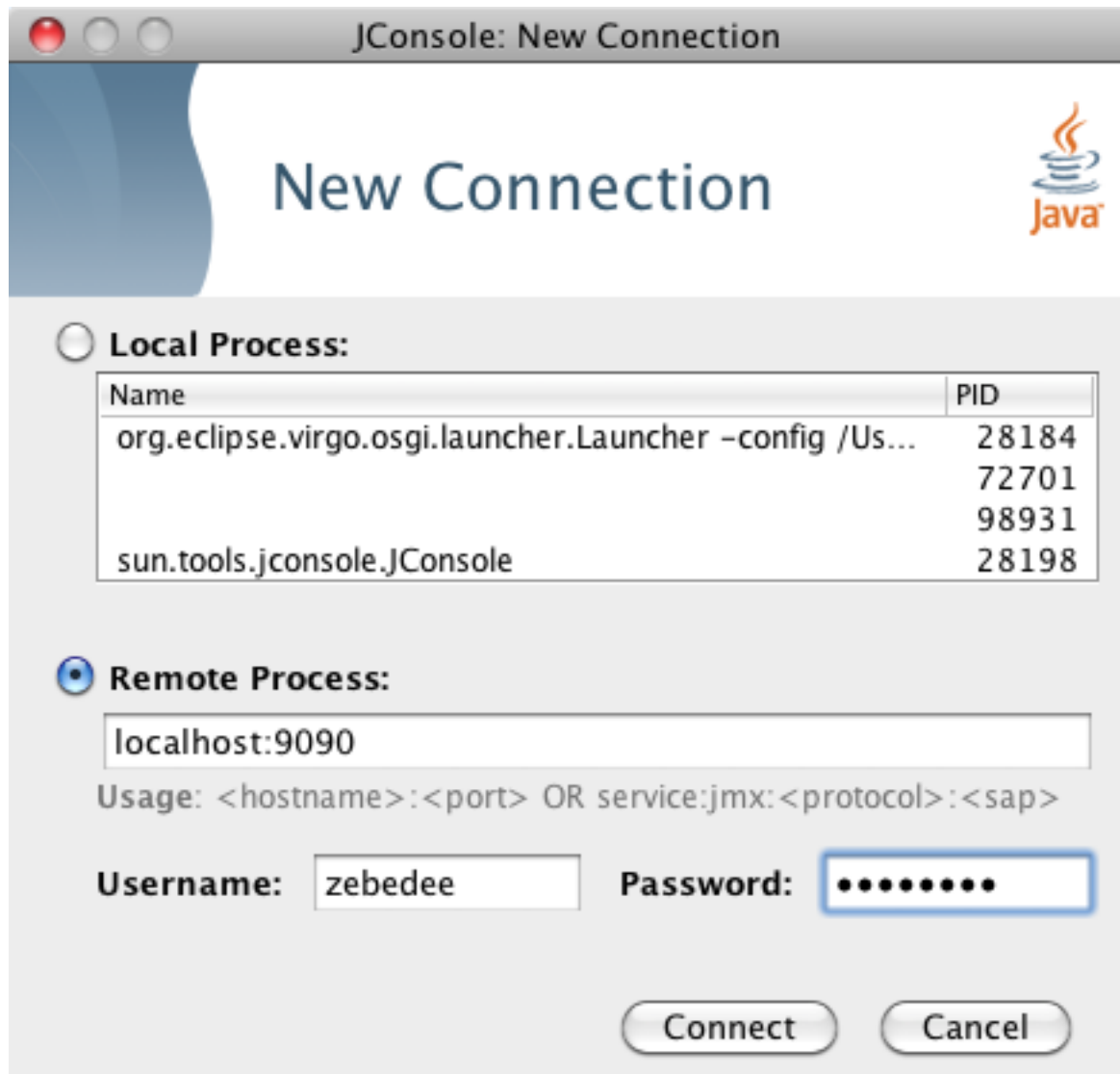
To start the JMX remote access with a custom username and password, update the `%SERVER_HOME%\config\org.eclipse.virgo.kernel.users.properties` file. First specify the custom username by changing the value of the `role.admin` property. Then set the password of this new user by adding a new property called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

For example, if you want change the JMX remote access username to `zebedee` with password `florence`, change the file as follows:

```
#####
# User definitions
#####
user.zebedee=florence

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

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 Virgo Web 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 Virgo Web Server installation to run multiple instances of Web 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 Virgo Web Server.

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

## Linux

To start Virgo Web Server using a config directory of `/config/node1`:

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

## Windows

To start Virgo Web Server using a config directory of `c:\config\node1`:

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

## 2.6 Stopping Virgo Web Server

### Linux

To stop a running instance of Virgo Web 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 Virgo Web 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 Web 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 Web Server instance:

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



## Microsoft Windows

To stop a running instance of Virgo Web 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 Virgo Web 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 Web 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 Web Server instance:

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



## 3. Overview of the Web Server Kernel and User Region

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

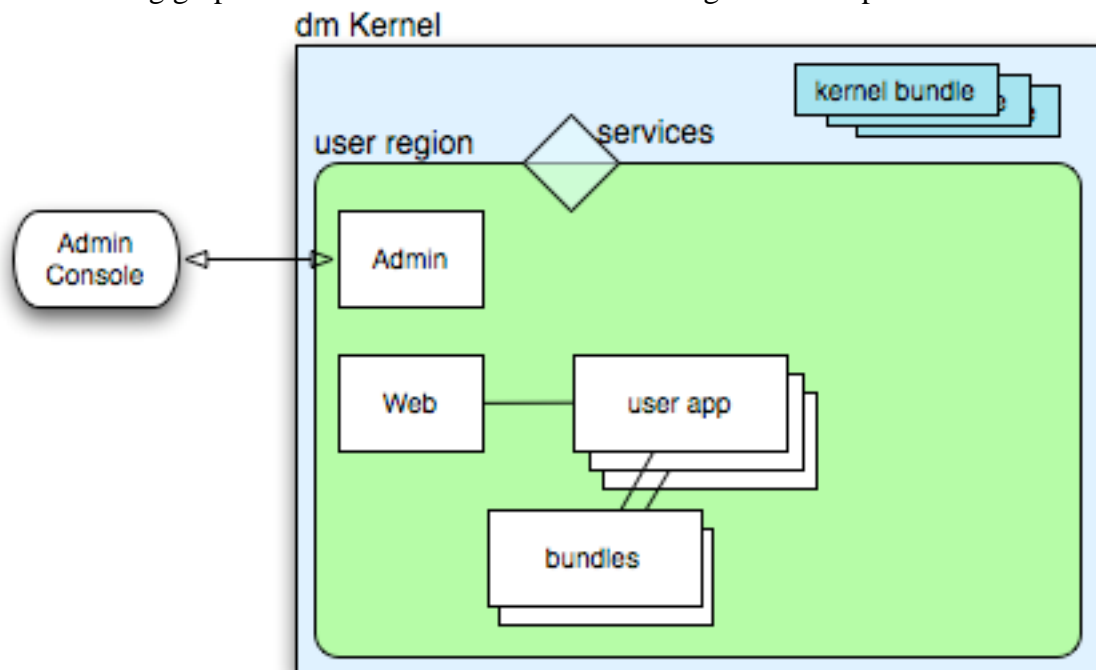
- The *kernel*, which is the heart of VWS. It makes up most of the VWS, except for the part that supports Web applications. In other words, the kernel provides full OSGi modular support for your applications, as long as they are not Web-based.

See [The Web 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 VWS itself, such as the Admin Console, making it much easier for you to administer VWS.

See [The Web Server User Region](#) for additional information.

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



When you download and install Virgo Web 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 VWS Kernel

The Virgo Kernel encapsulates almost all of VWS except for the deployment of Web applications. In sum, the kernel provides the following VWS 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 VWS](#) for details about configuring the default kernel to better suit your environment.

## 3.2 The VWS User Region

The user region isolates the kernel from deployed applications, including both your own user applications and the user-oriented VWS 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 VWS greatly simplifies the administration tasks you perform. In particular, when you use the Admin Shell or the Admin Console to manage Web Server, you do not see the many bundles that are internal to the kernel. The only exceptions are the kernel bundles that VWS 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 accommodate 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 VWS, the kernel creates a single user region. The configuration of the user region and the kernel is completely separate; see [Configuring VWS](#) 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. Equinox Console Extension

Virgo provides an extension to the Equinox console that allows you to examine artifacts currently installed to a particular Web 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 Equinox console via telnet to the configured port (2401 by default):

```
prompt$ telnet localhost 2401
Trying ::1...
Connected to localhost.
Escape character is '^]'.
osgi>
```

### 4.1 Using the Equinox Console Extension

You invoke the Equinox Console Extension using the `vsh` command. For example:

```
osgi> vsh help

bundle   - Management and examination of bundle artifacts
config   - Management and examination of configuration artifacts
exit      - Exit the kernel shell environment
help      - Get help on commands
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  - Shutdown Virgo Kernel

osgi>
```

## Equinox Console Extension Commands

The following table lists the Equinox Console Extension 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. Equinox Console Extension Commands*

Command	Description
<a href="#">bundle</a>	Used to manage and display information about bundle artifacts.
<a href="#">config</a>	Used to manage and display information about configuration artifacts.
<a href="#">package</a>	Used to manage and display information about exported packages.
<a href="#">par</a>	Used to manage and display information about PAR artifacts.
<a href="#">plan</a>	Used to manage and display information about plan artifacts.
<a href="#">service</a>	Displays information about services in the OSGi registry.
<a href="#">install</a>	Used to install an artifact to Web Server.
<a href="#">shutdown</a>	Shuts down the Web Server instance to which the Equinox Console Extension is connected.
<a href="#">help</a>	Display help about the list of available commands, as well as more detailed help about individual commands.
<a href="#">exit</a>	Return to the Equinox console with no effect.

## 4.2 Equinox Console Extension Command Reference

This section contains reference information about the following Equinox Console Extension commands:

- [bundle](#)
- [config](#)
- [package](#)
- [par](#)
- [plan](#)



- [service](#)
- [install](#)
- [shutdown](#)
- [help](#)
- [exit](#)

## bundle Command

Use the `bundle` command to manage the lifecycle of bundles deployed to VWS 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	Description
<code>list</code>	<p>Displays the list of bundles that are currently installed to the current Web Server instance. With the exception of a few kernel bundles and their services, which Web 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.</p> <p>Each bundle is identified by an internal ID which you can then use with the other <code>bundle</code> commands that manage a particular bundle, such as <code>start id</code>. The <code>list</code> command also displays the version of the bundle, along with its state, which is one of the following standard OSGi lifecycle states:</p> <ul style="list-style-type: none"><li>• <b>Installed:</b> The bundle is installed but its dependencies have not yet been resolved.</li><li>• <b>Resolved:</b> The bundle is resolved and you can now start it.</li><li>• <b>Uninstalled:</b> The bundle is uninstalled and you can not use it.</li></ul>

Option	Description
	<ul style="list-style-type: none"> <li>• <b>Starting:</b> The bundle is in the process of starting.</li> <li>• <b>Active:</b> The bundle is running and you can now use it.</li> <li>• <b>Stopping:</b> The bundle is in the process of stopping.</li> </ul> <p>Use one of the other <code>bundle</code> command to change the state of a bundle. For example, use the <code>bundle start id</code> command to change the state of a bundle from <code>Installed</code> to <code>Active</code>.</p>
<code>examine id</code>	<p>Displays detailed information about the specified bundle. Use the <code>bundle list</code> command to get the internal id of a particular bundle.</p> <p>In addition to the information provided by the <code>bundle list</code> command (id, full name, version, and state), the <code>examine</code> command specifies whether the bundle includes a Spring application context (or is <i>Spring Powered</i>) and the exact physical location of the bundle JAR file.</p> <p>The <code>examine</code> 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.</p>
<code>start id</code>	<p>Starts the specified bundle. Use the <code>bundle list</code> command to get the internal id of a particular bundle.</p> <p>To start a bundle, it must have already been resolved by Web Server, or in other words, be in the <code>OSGi Resolved</code> state. After Web Server successfully starts the bundle, it is listed in the <code>Active</code> state.</p>
<code>stop id</code>	<p>Stops the specified bundle. Use the <code>bundle</code></p>

Option	Description
	<p><code>list</code> command to get the internal id of a particular bundle.</p> <p>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 application that the bundle contains.</p>
<code>refresh id</code>	<p>Updates the contents of the specified bundle. Use the <code>bundle list</code> 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.</p>
<code>uninstall id</code>	<p>Uninstalls the specified bundle from Web Server. Use the <code>bundle list</code> command to get the internal id of a particular bundle.</p> <p>When the uninstall process is complete, the bundle does not show up in the list of bundles displayed by the <code>bundle list</code> command. If you want to use the application in the bundle, you must re-install it using the <code>install</code> command.</p>
<code>diag id</code>	<p>Provides diagnostic information about the specified bundle. In particular, this command displays information about the imported packages that Web Server could not resolve. Use the <code>bundle list</code> command to get the internal id of a particular bundle.</p>
<code>headers id</code>	<p>Displays the complete list of manifest headers of the specified bundle. Use the <code>bundle list</code> command to get the internal id of a particular bundle.</p> <p>The manifest headers include: <code>Import-Package</code>, <code>Module-Type</code>, <code>Bundle-SymbolicName</code>, and so on.</p>

The following examples show how to use this command.

First, use the `bundle list` command to view all the installed bundles:

```
osgi> vsh bundle list
```

```

Id      Name                                     Version                                     State
0      org.eclipse.osgi                           3.5.1.R35x_v20091005                     ACTIVE
1      org.eclipse.virgo.region.user              0.0.0                                     ACTIVE
2      org.eclipse.virgo.kernel.userregion        2.1.0.BUILD-20100609153042                ACTIVE
3      org.eclipse.virgo.kernel.osgicommand       2.1.0.BUILD-20100609153042                ACTIVE
4      org.springframework.osgi.core              1.2.1                                     ACTIVE
5      S org.springframework.osgi.extender         1.2.1                                     ACTIVE
6      org.springframework.osgi.io                1.2.1                                     ACTIVE
7      org.eclipse.virgo.kernel.agent.dm          2.1.0.BUILD-20100609153042                ACTIVE
8      S org.eclipse.virgo.kernel.deployer.dm     2.1.0.BUILD-20100609153042                ACTIVE
9      com.springsource.org.aopalliance            1.0.0                                     ACTIVE
10     org.eclipse.virgo.kernel.dmfragment         2.1.0.BUILD-20100609153042                RESOLVED
11     org.springframework.aop                    3.0.0.RELEASE                             ACTIVE
12     org.springframework.asm                   3.0.0.RELEASE                             ACTIVE
13     org.springframework.beans                 3.0.0.RELEASE                             ACTIVE
14     org.springframework.context               3.0.0.RELEASE                             ACTIVE
15     org.springframework.core                  3.0.0.RELEASE                             ACTIVE
16     org.springframework.expression            3.0.0.RELEASE                             ACTIVE
osgi>

```

The following example shows how to view the headers of the `org.springframework.osgi.extender` bundle (only the first few lines are shown):

```

osgi> vsh bundle examine 5

Id:                5
Name:              org.springframework.osgi.extender
Version:           1.2.1
State:             ACTIVE
Spring Powered:    true
Bundle Location:   file:<... omitted ...>/org.springframework.osgi.extender-1.2.1.jar/

Imported Packages:
  org.springframework.osgi.context [1.2.1, 1.2.1]
    exported by org.springframework.osgi.core 1.2.1 [4]
  <... remainder omitted ...>

Exported Packages:
  org.springframework.osgi.extender 1.2.1
  <... remainder omitted ...>

Published services:
  58 org.springframework.beans.factory.xml.NamespaceHandlerResolver
    consumed by org.springframework.osgi.extender 1.2.1 [5]
    consumed by org.eclipse.virgo.kernel.deployer.dm 2.1.0.BUILD-20100609153042 [8]
  <... remainder omitted ...>

Consumed services:
  34 org.eclipse.virgo.kernel.shim.serviceability.TracingService
    published by org.eclipse.virgo.region.user 0.0.0 [1]
  <... remainder omitted ...>

Fragments:
  org.eclipse.virgo.kernel.dmfragment 2.1.0.BUILD-20100609153042 [10]
osgi>

```

## config Command

Use the `config` command to view and manage the configuration artifacts that have been installed to Web 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	Description
list	<p>Lists the configuration artifacts that are currently installed in Web Server.</p> <p>The <code>list</code> 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 <a href="#">the bundle command</a> for the list of possible states.</p>
<i>examine name [version]</i>	<p>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 <code>config list</code> command to view all configuration artifacts and versions currently installed in Web Server.</p> <p>A configuration artifact must be active for you to examine it; if it is not currently active, use <code>config start</code> to start it and thus change its state to <code>Active</code>.</p> <p>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.</p>
<i>start name [version]</i>	<p>Starts the specified configuration artifact and makes it visible to the internal configuration sub-system of Web 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 <code>config list</code> command to view all configuration artifacts and versions currently installed in Web Server.</p> <p>Starting the configuration sets its state to <code>Active</code>.</p>
<i>stop name [version]</i>	<p>Stops the specified configuration artifact and makes it invisible to the internal configuration</p>

Option	Description
	<p>sub-system of Web 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 <code>config list</code> command to view all configuration artifacts and versions currently installed in Web Server.</p> <p>Stopping the configuration sets its state to <code>Resolved</code>.</p>
<code>refresh name [version]</code>	<p>Updates the contents of the specified configuration artifact to the internal configuration sub-system of Web 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 <code>config list</code> command to view all configuration artifacts and versions currently installed in Web Server.</p> <p>Use this command if you have changed the contents of the configuration artifact, and you want to make this information known to Web Server and the associated bundle.</p>
<code>uninstall name [version]</code>	<p>Uninstalls the specified configuration artifact and make it completely unavailable to Web 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 <code>config list</code> command to view all configuration artifacts and versions currently installed in Web Server.</p> <p>Stopping the configuration removes it from Web Server's list of deployed artifacts and it will not show up when you perform a <code>config list</code>.</p>

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

<code>osgi&gt; vsh config list</code>		
Name org.eclipse.virgo.kernel	Version 0.0.0	State ACTIVE

```
org.eclipse.virgo.kernel.jmxremote.access 0.0.0 ACTIVE
org.eclipse.virgo.kernel.userregion 0.0.0 ACTIVE
org.eclipse.virgo.kernel.users 0.0.0 ACTIVE
org.eclipse.virgo.medic 0.0.0 ACTIVE
org.eclipse.virgo.repository 0.0.0 ACTIVE
osgi>
```

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

```
osgi> vsh config examine org.eclipse.virgo.medic

Factory pid:
Bundle Location: file:lib/kernel/org.eclipse.virgo.kernel.shell-2.1.0.BUILD-20100609153042.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:
    org.eclipse.virgo.medic
osgi>
```

## package Command

Use the `package` command to view the complete list of packages exported by all bundles installed to Web 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	Description
<code>list</code>	Displays all the exported packages for all bundles in Web 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 <code>bundle examine id</code> .
<code>examine name version</code>	Displays details about the exported bundle. You must specify both the name of the exported package and its version; use <code>package list</code> to view the exact names and version.  This command provides the following additional information about the exported

Option	Description
	<p>package:</p> <ul style="list-style-type: none"> <li>• The name and version of the bundle that exports the package. This means that the package name is explicitly listed in the bundle's <code>MANIFEST.MF</code> file as part of the <code>Export-Package</code> header.</li> <li>• Any attributes that are part of the <code>Export-Package</code>, in addition to version.</li> <li>• The directives that are part of the <code>Export-Package</code> header. A typical directive is <code>uses</code>, which declares up-front constraints on a number of other packages.</li> <li>• The list of all bundles that import the package.</li> </ul>

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

```
osgi> vsh package list

Name                               Version    Providing Bundle
javafx.accessibility              0.0.0     0
javafx.activation                 0.0.0     0
javafx.activation                 1.1.1     0
<... remainder omitted ...>

osgi>
```

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

```
osgi> vsh package examine org.slf4j 1.5.10

Exporter: org.eclipse.virgo.region.user 0.0.0 [1]

Attributes:
  None

Directives:
  uses:
    org.slf4j.impl, org.slf4j.spi
  x-equinox-ee:
    -1
  x-internal:
    false

Importer(s):
  org.eclipse.virgo.kernel.userregion 2.1.0.BUILD-20100609153042 [2]
    Import-Package attributes:
      bundle-version:
        0.0.0
      version:
        [1.5.10,2.0.0)
    Import-Package directives:
      resolution:
        static
    <... remainder omitted ...>

osgi>
```



## par Command

Use the `par` command to view all the PARs currently installed in Web 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	Description
<code>list</code>	<p>Displays all the PARs that are currently installed in Web Server.</p> <p>The <code>list</code> 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 <a href="#">the bundle command</a> for the list of possible states.</p>
<code>examine name version</code>	<p>Displays information about the specified PAR; you are required to identify the PAR with both its name and its version. Use the <code>par list</code> command to view all installed PAR files and their versions. The command displays the following information:</p> <ul style="list-style-type: none"> <li>• The current state of the PAR (see <a href="#">the bundle command</a> for the full list of possible states).</li> <li>• Whether the PAR is <i>scoped</i>. Scoping specifies whether Web Server should deploy the members of the PAR in their own scope; when scoping is disabled, Web Server deploys the artifacts into the global scope and they are accessible for access by all other artifacts.</li> <li>• Whether the PAR is <i>atomic</i>. When a PAR is atomic, Web Server manages the lifecycle of all its member artifacts as a single entity, which means if one artifact member is started, then Web Server starts <i>all</i> the PAR artifacts. If one artifact fails to start, then Web Server stops all other artifacts in the</li> </ul>

Option	Description
	<p>PAR.</p> <ul style="list-style-type: none"> <li>The individual members, or children, of the PAR. These could be other PARs, plans, bundles, configuration artifacts, and so on.</li> </ul>
<code>start name version</code>	<p>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 <code>par list</code> command to get the list of PARs currently installed in Web Server.</p> <p>To start a PAR, it must have already been resolved by Web Server, or in other words, be in the <code>OSGi Resolved</code> state. After Web Server successfully starts the PAR, it is listed in the <code>Active</code> state.</p>
<code>stop name version</code>	<p>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 <code>par list</code> command to get the list of PARs currently installed in Web Server.</p> <p>When you stop a PAR, it goes from the <code>OSGi Active</code> state to the <code>Resolved</code> state, and you must re-start it if you want to use the application that the PAR contains.</p>
<code>refresh name version</code>	<p>Updates the contents of the specified PAR. You must specify both the name and version of the PAR you want to refresh. Use the <code>par list</code> command to this information.</p> <p>Use this command if you have changed the contents of the PAR file and you want to refresh the artifact as installed in the <code>OSGi</code> framework.</p>
<code>uninstall name version</code>	<p>Uninstalls the specified PAR. You must specify both the name and version of the PAR you want to refresh. Use the <code>par list</code> command to this information.</p> <p>When the uninstall process is complete, the PAR will not show up in the list of PARs displayed by the <code>par list</code> command. If you want to use the application in the PAR, you</p>

Option	Description
	must re-install it using the <code>install</code> command.

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

```
osgi> vsh par list

Name                                Version                                State
org.eclipse.virgo.server.repository.hosted  2.0.0.D-20091109225604  ACTIVE
osgi>
```

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

```
osgi> vsh par examine org.eclipse.virgo.server.repository.hosted 2.0.0.D-20091109225604

State:  ACTIVE
Scoped: true
Atomic: true

Children:
  bundle org.eclipse.virgo.server.repository.hosted.core 2.0.0.D-20091109225604
  bundle org.eclipse.virgo.server.repository.hosted.web 2.0.0.D-20091109225604
  bundle org.eclipse.virgo.server.repository.hosted-synthetic.context 2.0.0.D-20091109225604
osgi>
```

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

```
osgi> vsh par refresh my.exciting.par 1.2.0

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

## plan Command

Use the `plan` command to view all the plans currently installed in Web 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	Description
list	<p>Displays all the plans that are currently installed in Web Server.</p> <p>The <code>list</code> 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 <a href="#">the bundle command</a></p>

Option	Description
	for the list of possible states.
<i>examine name version</i>	<p>Displays information about the specified plan; you are required to identify the plan with both its name and its version. Use the <code>plan list</code> command to view all installed plans and their versions. The command displays the following information:</p> <ul style="list-style-type: none"> <li>• The current state of the plan (see <a href="#">the bundle command</a> for the full list of possible states).</li> <li>• Whether the plan is <i>scoped</i>. Scoping specifies whether Web Server should deploy the members of the plan in their own scope; when scoping is disabled, Web Server deploys the artifacts into the global scope and they are accessible for access by all other artifacts.</li> <li>• Whether the plan is <i>atomic</i>. When a plan is atomic, Web Server manages the lifecycle of all its member artifacts as a single entity, which means if one artifact member is started, then Web Server starts <i>all</i> the plan artifacts. If one artifact fails to start, then Web Server stops all other artifacts in the plan.</li> <li>• The individual members, or children, of the plan. These could be other plans, PARs, bundles, configuration artifacts, and so on.</li> </ul>
<i>start name version</i>	<p>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 <code>plan list</code> command to get the list of plans currently installed in Web Server.</p> <p>To start a plan, it must have already been resolved by Web Server, or in other words, be in the <code>OSGi Resolved</code> state. After Web Server successfully starts the plan, it is listed in the <code>Active</code> state.</p>
<i>stop name version</i>	<p>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 <code>plan list</code> command to get the list of plans</p>

Option	Description
	<p>currently installed in Web Server.</p> <p>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.</p>
<code>refresh name version</code>	<p>Updates the contents of the specified plan. You must specify both the name and version of the plan you want to refresh. Use the <code>plan list</code> command to this information.</p> <p>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.</p>
<code>uninstall name version</code>	<p>Uninstalls the specified plan. You must specify both the name and version of the plan you want to refresh. Use the <code>plan list</code> command to this information.</p> <p>When the uninstall process is complete, the plan will not show up in the list of plans displayed by the <code>plan list</code> command. If you want to use the application in the plan, you must re-install it using the <code>install</code> command.</p>

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

```
osgi> vsh plan list

Name                                     Version      State
org.eclipse.virgo.kernel.region.springdm 2.0.0        ACTIVE
org.eclipse.virgo.server.admin.plan      2.0.0        ACTIVE
org.eclipse.virgo.server.web              2.0.0        ACTIVE

osgi>
```

The following example shows how to examine a particular plan:

```
osgi> vsh plan examine org.eclipse.virgo.kernel.userregion.springdm 2.1.0

State:  ACTIVE
Scoped: false
Atomic: false

Children:
  bundle org.eclipse.virgo.kernel.deployer.dm 2.1.0.BUILD-20100609153042
  bundle org.springframework.osgi.io 1.2.1
  bundle org.springframework.osgi.extender 1.2.1
  bundle org.springframework.osgi.core 1.2.1
  bundle org.eclipse.virgo.kernel.agent.dm 2.1.0.BUILD-20100609153042

osgi>
```

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

```
osgi> vsh plan stop org.eclipse.virgo.server.web 2.0.0
plan org.eclipse.virgo.server.web:2.0.0 stopped successfully
osgi>
```

The following example shows how to start a plan:

```
osgi> vsh plan start org.eclipse.virgo.server.web 2.0.0
plan org.eclipse.virgo.server.web:2.0.0 started successfully
osgi>
```

**service Command**

Use the `service` command to view all the services that have been registered in the OSGi service registry of Web 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	Description
list	<p>Displays the list of services that are currently registered in the OSGi service registry of Web Server.</p> <p>Each service is identified by an internal ID which you can then use with the <code>service examine</code> command to view the details about a particular service. The <code>list</code> command also displays the object class that implements the service and the internal id of the bundle that provides the service.</p>
examine <i>id</i>	<p>Displays detailed information about the specified service. Use the <code>service list</code> command to get the internal id of a particular service.</p> <p>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.</p>

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

```
osgi> vsh 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
<... remainder omitted ...>

65  com.springsource.osgi.webcontainer.core.spi.ServletContainer 35
66  com.springsource.osgi.webcontainer.core.WebContainer 34
67  org.eclipse.virgo.server.web.core.WebApplicationRegistry 36
<... remainder omitted ...>

osgi>
```

The following example shows how to examine a particular service:

```
osgi> vsh service examine 38

Properties:
  Bundle-SymbolicName:
    org.eclipse.virgo.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):
  org.eclipse.virgo.kernel.userregion 2.0.0.D-20091110110152 [2]

osgi>
```

## install Command

Use the `install` command to deploy an artifact to Web 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, Web 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 Equinox Console Extension 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 Web 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`):

```
osgi> vsh 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
<... remainder omitted ...>			
59	org.eclipse.virgo.server.splash	2.0.0.D-20091110115520	ACTIVE
60	swf-booking-mvc.war	0.0.0	RESOLVED

```
osgi>
```

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:

```
osgi> vsh bundle start 60
```

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

```
osgi> vsh bundle examine 60
```

```
Id: 60
Name: swf-booking-mvc.war
Version: 0.0.0
State: ACTIVE
Spring Powered: true
Bundle Location: file:<... omitted ...>/swf-booking-mvc.war/
```

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]
  <... remainder omitted ...>
```

```
osgi>
```

## shutdown Command

Use the `shutdown` command to shut down the Web Server instance to which you are connected. When Web 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 Equinox Console Extension 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:

```
osgi> vsh help

bundle      - Management and examination of bundle artifacts
config      - Management and examination of configuration artifacts
exit        - Exit the kernel shell environment
help        - Get help on commands
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    - Shutdown Virgo Kernel

osgi> vsh 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

osgi>
```

## exit Command

This returns to the Equinox console and otherwise has no effect.

The `exit` command does not have any options.



## 5. The Web Admin Console

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

- [View an overview of the Web Server properties.](#)
- [View and manage the lifecycle](#) of artifacts already deployed to the Web 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 Web Server.](#) .
- [View the properties of the configuration artifacts](#) deployed to Web Server.
- [View details of dump files](#) that Web Server might have generated after encountering a problem. This feature is particularly valuable if Web 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 Web Server, or in other words, a list of all bundles currently installed in Web 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 Admin Console, start the Virgo Web Server and then enter the following URL in your browser of choice.

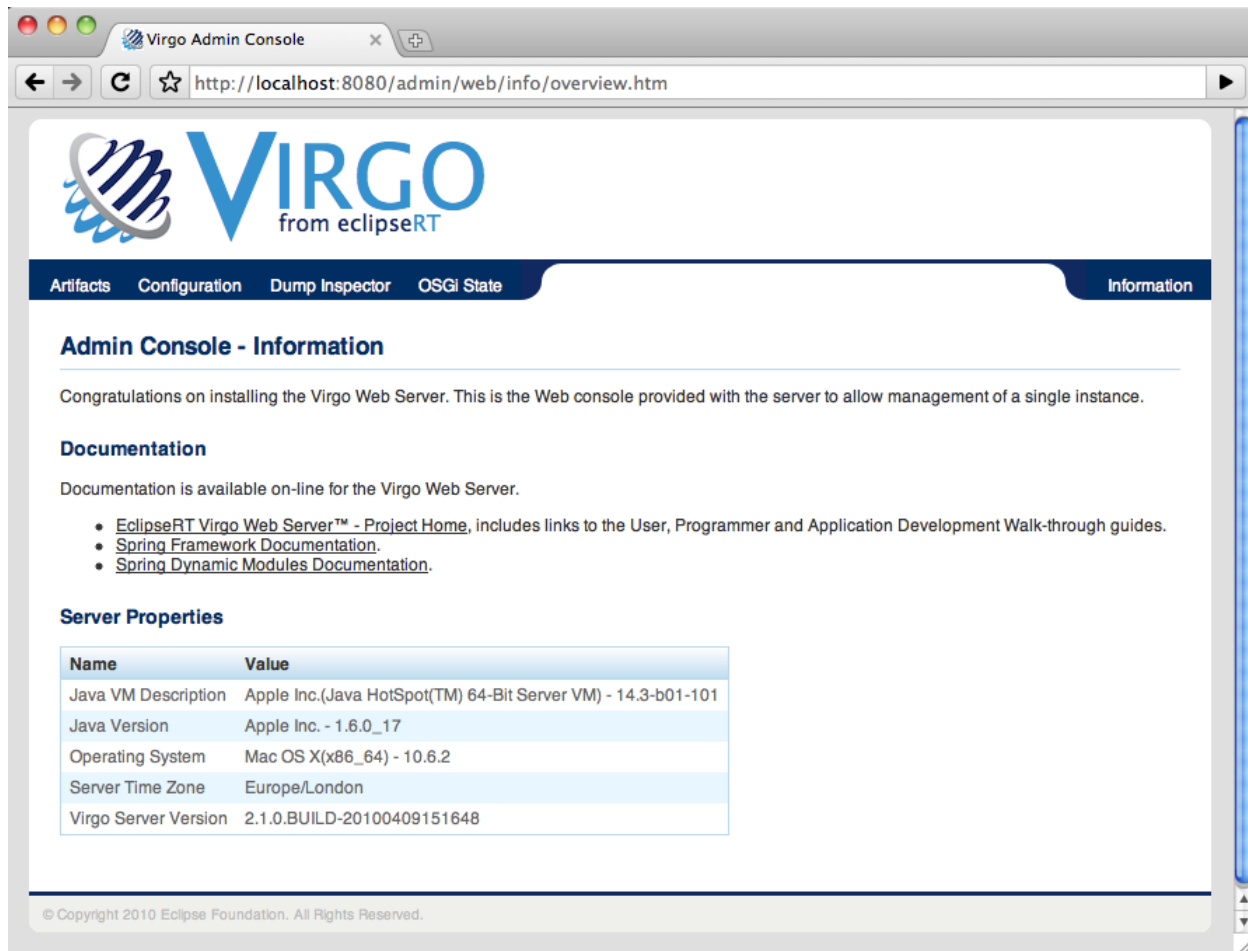
```
http://localhost:8080/admin
```

Replace `localhost` with the hostname of the computer on which the Virgo Web 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 Web 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 Web Server itself, such as details about the Java Virtual Machine (JVM), the operating system on which Web Server is installed, the time zone configured for the computer, and the complete version of Web Server.

## Changing the Admin User

To change the ID and password for the Admin Console, update the `SERVER_HOME/config/org.eclipse.virgo.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 Web 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 Web 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](#)

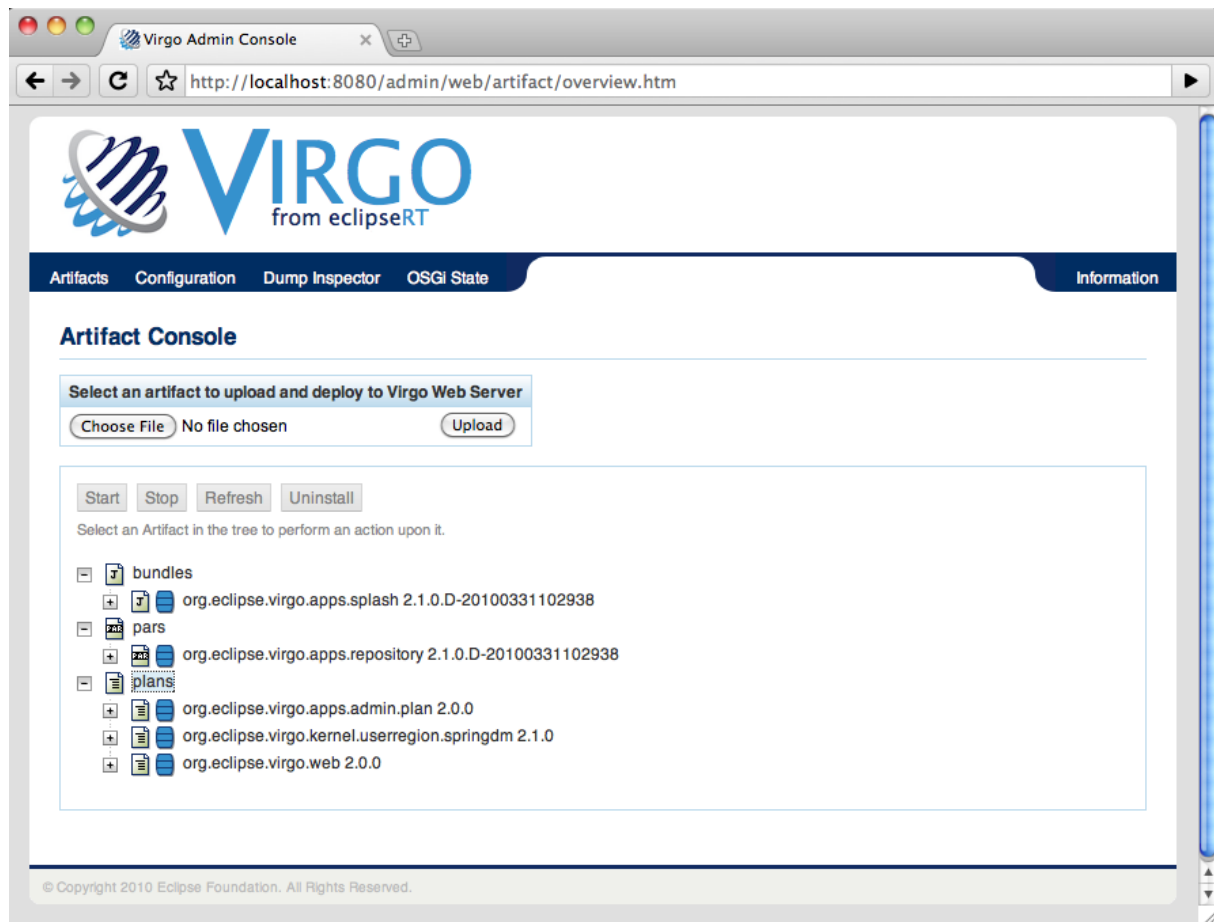
### Viewing and Managing the Lifecycle of Deployed Artifacts

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

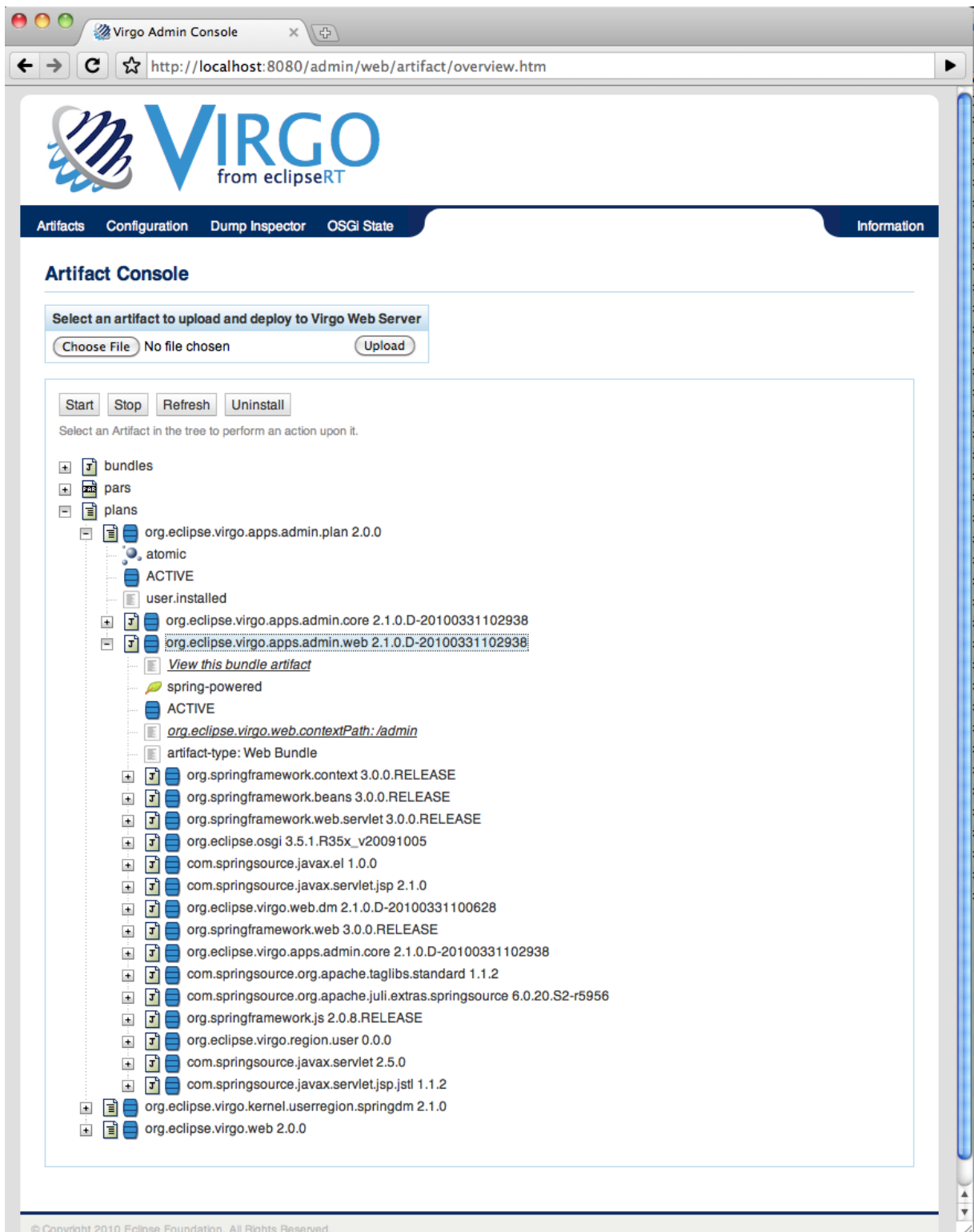
1. From the main Admin Console page, click the **Artifacts** link at the top.

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 Web Server: bundles, configuration files, PARs, and plans. When you first install Web 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 `org.eclipse.virgo.apps.admin.web` 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.

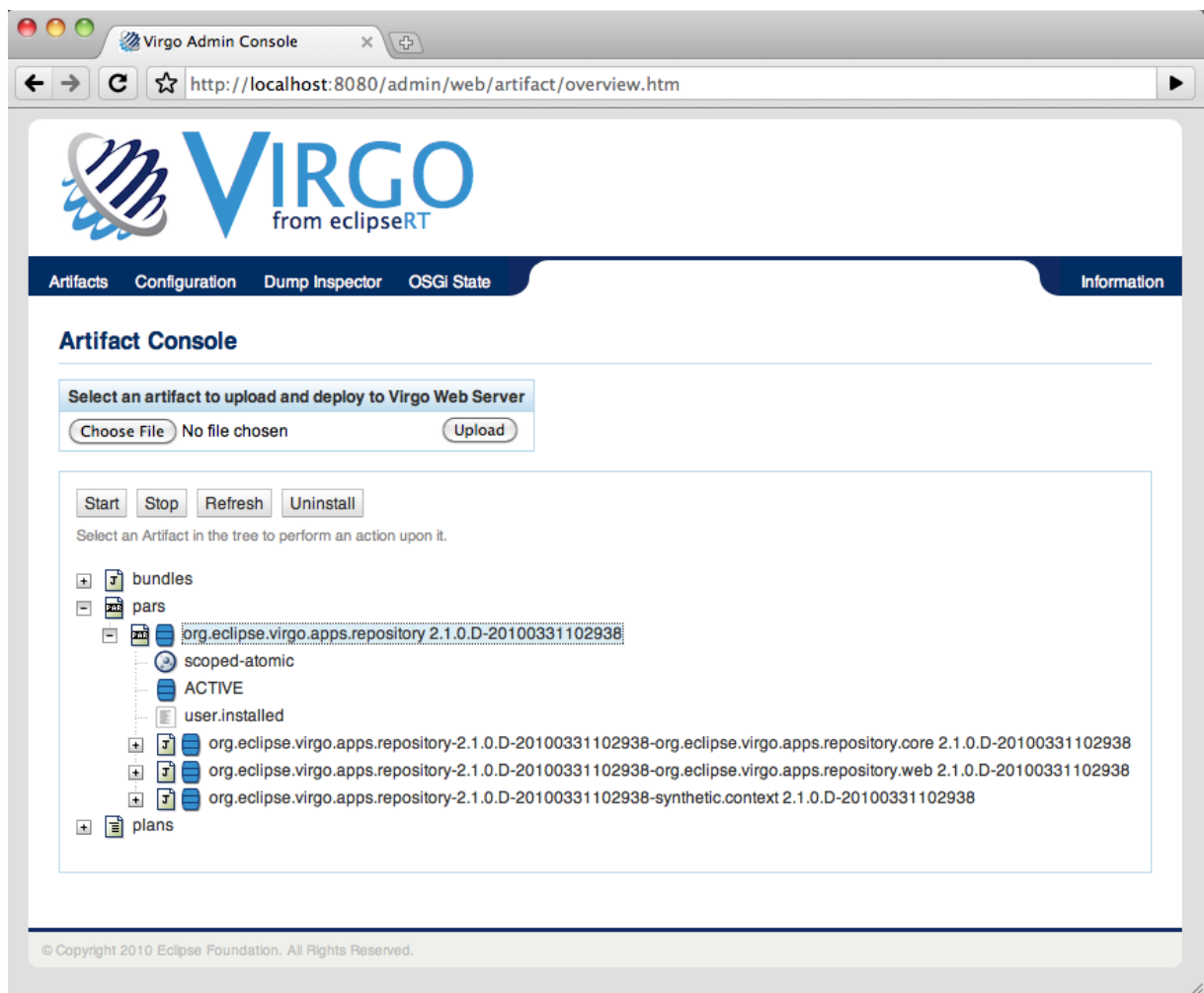
As shown in the preceding graphic, the Admin Console provides a link for Web modules that you can click on to actually invoke the application  
(`org.eclipse.virgo.server.web.contextPath:/config-properties` in the example above.)

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

- **Scoped.** Scoping specifies whether Web Server should deploy the members of the PAR/plan in their own scope; when scoping is disabled, Web 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, Web Server manages the lifecycle of all its member artifacts as a single entity, which means if one artifact member is started, then Web Server starts all the PAR/plan artifacts. If one artifact fails to start, then Web 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 artifact 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 Web 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 Web 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 Web Server.

Web 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 Web Server.

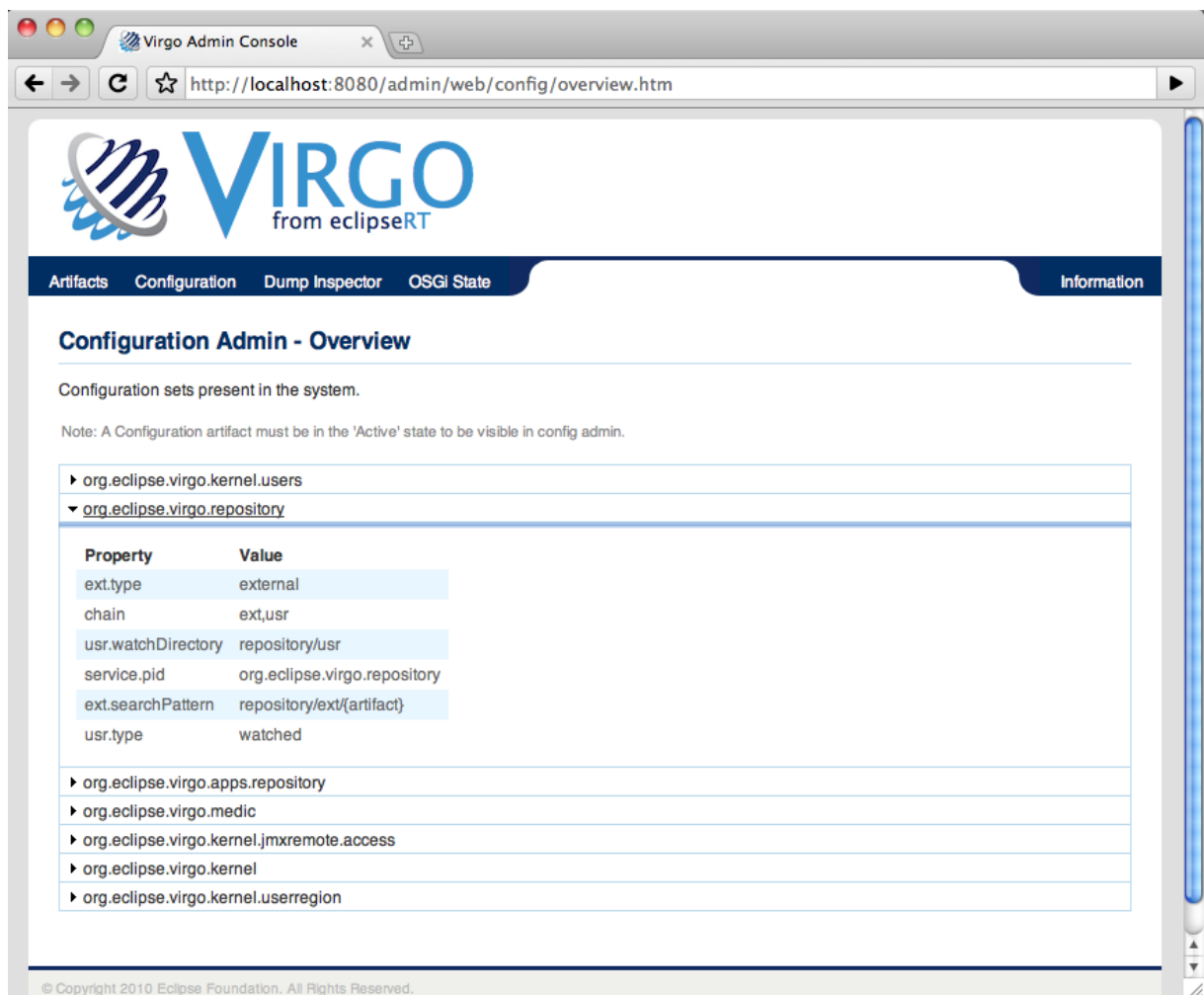
4. Expand the artifact tree to view your newly deployed artifact. If Web Server installed it without errors, it should show up in the appropriate section and be in an Active state.

## Viewing Properties of Deployed Configuration Artifacts

The following procedure describes how you can view the list of configuration artifacts that are currently deployed to Web 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 Web Server. Each time a dump is triggered for Web 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.

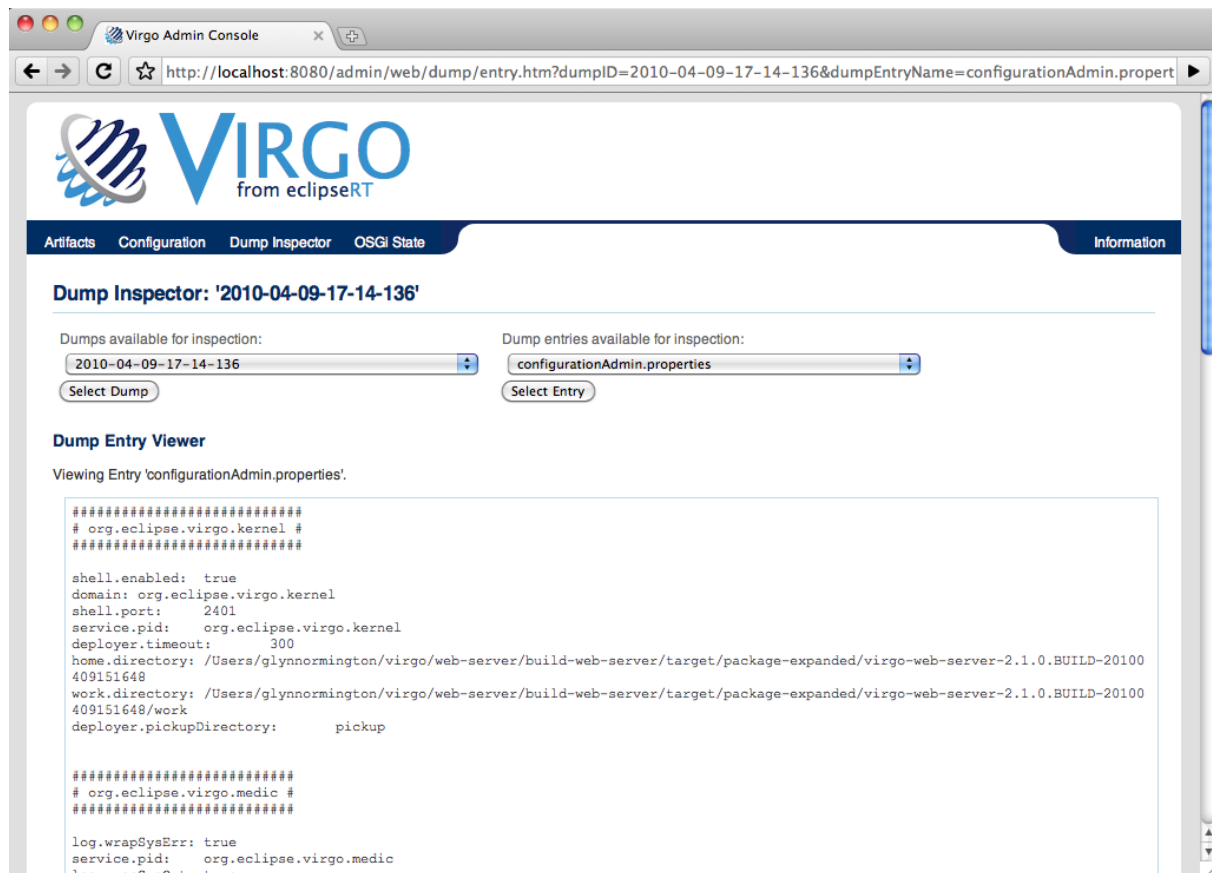
A service dump is triggered when there is either a failure in the Web Server code or Web Server detects a thread deadlock in either its own code or a user application. The service dump contains a snapshot of all the important state from the running Web Server instance. **NOTE:** This snapshot is not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

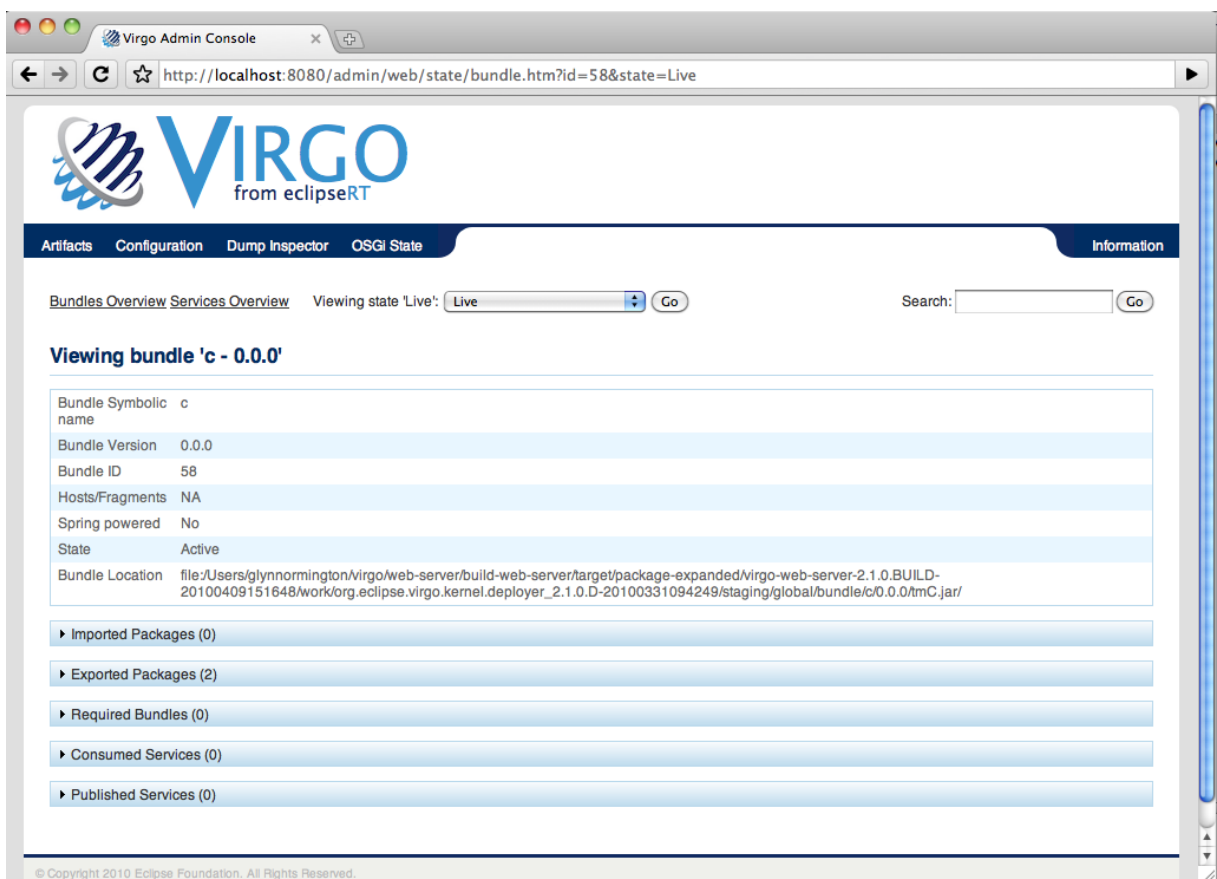
The following procedure describes how you can view the OSGi state of the Web Server, either currently or at the time that a particular service dump occurred. The OSGi state is a list of bundles that are currently installed as well as a list of all the services that are provided by these bundles.

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 Web 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 Web 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 Virgo Web 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 Virgo Web 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 Virgo Web Server.

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

In Virgo Web 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, Virgo Web Server installs the bundle in which it is packaged to the VWS 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 VWS has thus installed), the VWS searches the provisioning repository for an artifact that can satisfy that dependency.

The provisioning repository for a particular instance of Virgo Web 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 Virgo Web Server instance. The artifacts in a local repository include installed third-party libraries, bundles supplied by VWS, bundles supplied by an end user, and internal bundles used only by VWS. You can further categorize this location into external directories that adhere to a specified search pattern and are scanned by VWS just at startup, or watched directories that point to a single directory location and VWS scans on a regular basis.
- **Remote:** This means that a local instance of Virgo Web Server gets the artifact from a remotely-hosted repository that is physically located on a remote Virgo Web Server instance. You configure the provisioning repository using the `SERVER_HOME/config/com.springsource.repository.properties` file.

As previously described, a particular instance of Virgo Web 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 Virgo Web 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 Virgo Web Server, the local provisioning repository is located at `$SERVER_HOME/repository` by default and consists of two main directories: `ext` and `usr`. The `ext` directory contains bundles and libraries supplied with the Virgo Web 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 VWS 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`.

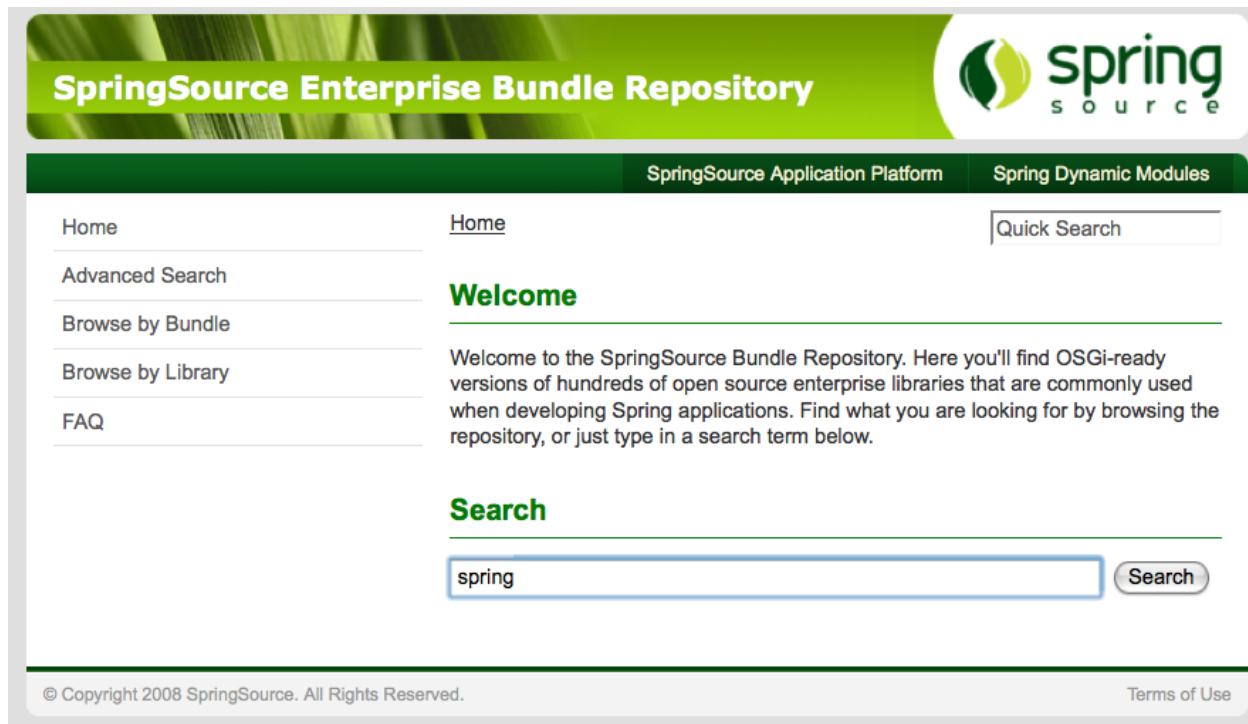
In some cases the Virgo Web Server manages to automatically detect changes in its provisioning repository at runtime, thereby avoiding the need to restart the VWS.

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 VWS. For other changes such as addition of indirect dependencies, the Virgo Web 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 VWS. 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 Virgo Web 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 Virgo Web 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 VWS, instance, and how to configure the local VWS instance to itself host a repository that other remote servers access.

## 7. Serviceability

Logging (both event logging and trace logging) in VWS 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—VWS 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 Virgo Web 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 Virgo Web Server's logging (trace) support serves two main purposes:

- It provides global trace files that capture high-volume information regarding the Virgo Web 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 VWS 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 VWS 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

Virgo Web 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

Virgo Web 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 Virgo Web Server code, or
2. a thread deadlock is detected.

A service dump contains a snapshot of all the important state from the running Virgo Web 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 Virgo Web 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, VWS 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 Virgo Web Server using either the hot-deploy directory on the file system or by using the Admin Console. The artifacts that you can deploy to VWS 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, VWS 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 Virgo Web Server. As soon as Virgo Web 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, Web 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, Virgo Web 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 Virgo Web Server will not do so.

## Deployment Ordering

When deploying bundles that have dependencies, it is important that you deploy them in the correct order. Virgo Web 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 Virgo Web Server does not support deploying fragment bundles.

## 8.2 Undeploying Artifacts

You undeploy artifacts from Virgo Web 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 interchangeably.

### 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 Virgo Web 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 '/hel
[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 `org.eclipse.virgo.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 Virgo Web Server

You use configuration files in the `SERVER_HOME/config` directory to configure VWS. 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 Virgo Kernel and User Region

This section provides information about configuring the VWS 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
<code>org.eclipse.virgo.kernel.properties</code>	Configures <a href="#">kernel deployment</a> and the <a href="#">Admin Shell</a> of VWS.
<code>org.eclipse.virgo.kernel.userregion.properties</code>	Configures the <a href="#">user region</a> of VWS.
<code>org.eclipse.virgo.kernel.users.properties</code>	Configures the <a href="#">users that are allowed to access</a> the Admin Shell and Admin Console, and roles to which they map.
<code>org.eclipse.virgo.kernel.jmxremote.properties</code>	Configures the <a href="#">permissions for users</a> that are allowed to access the Admin Shell and Admin Console.
<code>org.eclipse.virgo.kernel.authentication.properties</code>	Configures the <a href="#">Java Authentication and Authorization Service (JAAS)</a> 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/org.eclipse.virgo.kernel.properties` file. The following table describes these properties.

*Table 9.2. Deployment Configuration Properties*

Property	Description
<code>deployer.timeout</code>	Specifies the amount of time, in seconds, after which VWS 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.
<code>deployer.pickupDirectory</code>	Specifies the absolute or relative path to the pickup directory to which you copy applications for hot-deployment. Relative paths are relative to <code>SERVER_HOME</code> . The default value is <code>./target/pickup..</code>

The following listing displays the default configuration distributed with the VWS; only relevant sections of the `org.eclipse.virgo.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 Admin Shell

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

You configure the Admin Shell by updating the `shell.XXX` properties in the `SERVER_HOME/config/org.eclipse.virgo.kernel.properties` file, as described in the following table:

*Table 9.3. Admin Shell Configuration Properties*

Property	Description
<code>shell.enabled</code>	Specifies whether the Admin Shell is enabled or not. Valid values are <code>true</code> or <code>false</code> .
<code>shell.port</code>	Defines the port on which you can remotely access the Admin Shell. If not set, the shell is

Property	Description
	only available from stdout of the VWS process.

The following example shows the default Admin Shell configuration in a freshly-installed `org.eclipse.virgo.kernel.properties` file; only the relevant section of the file is shown.

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

The example shows that the Admin 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 VWS 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 Admin Shell because the internal server bundles are not visible.

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

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

*Table 9.4. User Region Configuration Properties*

Property	Description
<code>baseBundles</code>	Specifies the hard-coded list of bundles that VWS installs directly into the user region. Virgo Web 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.
<code>packageImports</code>	Specifies the packages that exist in the kernel that VWS 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 <code>. *</code> wildcard. For example, <code>com.springsource.util. *</code> will import all packages that start with

Property	Description
	<code>com.springsource.util.</code>
<code>serviceImports</code>	Specifies the services in the kernel that are imported into the user region so that they're available to bundles in the user region.
<code>serviceExports</code>	Specifies the services in the user region that are imported into the kernel so that they're available to bundles in the kernel.
<code>inheritedFrameworkProperties</code>	Specifies the framework properties, configured in the <code>SERVER_HOME/lib/org.eclipse.virgo.kernel.launch</code> file, that will also be set on the user region's nested framework.
<code>initialArtifacts</code>	<p>Specifies the artifacts that VWS deploys into the user region when the server starts. Virgo Web 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; VWS automatically installs any other artifacts upon which they depend from the repository.</p> <p>You can use this property to convert a VWS into a Virgo Kernel by removing the <code>repository:plan/org.eclipse.virgo.server.web</code> plan.</p>

## Configuring Authentication

Virgo Web Server uses the [Java Authentication and Authorization Service \(JAAS\)](#) framework to authenticate the administration user that connects to Web Servers using the Admin Console or Admin 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/org.eclipse.virgo.kernel.authentication.config` file configures the underlying authentication technology for VWS. The short file consists of the following entry:

```
dm-kernel {
    org.eclipse.virgo.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 VWS uses to authenticate users is org.eclipse.virgo.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 Web Server is admin/springsource.

You configure the administration user in the org.eclipse.virgo.kernel.users.properties file. The default file for a freshly installed VWS is as follows:

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

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

The administration user that connect to the Admin Console and Admin 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 org.eclipse.virgo.kernel.users.properties file. For example, if you want the juliet user, with password supersecret, to be the new administration user, update the file as shown:

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

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

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

The final file involved in VWS authentication is SERVER\_HOME/config/org.eclipse.virgo.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 administration user would only be able to view information using the Admin Console and Admin Shell.

## 9.2 Configuring Serviceability

The serviceability sub-system of VWS 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 VWS instance when an internal failure or thread deadlock is detected.  
You configure service dumps for Virgo Web 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 VWS 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 Virgo Web 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 VWS.

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 VWS 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 VWS includes in addition to those supplied by the Logback, configured in the `serviceability.xml` file.

*Table 9.5. Serviceability Properties*

Property	Description
<code>dump.root.directory</code>	Specifies the directory to which VWS writes the service dumps. The directory name is relative to <code>SERVER_HOME</code> .
<code>log.wrapSysOut</code>	Specifies whether you want to capture <code>System.out</code> output from your applications to the application trace file. The output is logged by VWS's root logger, which captures INFO level and above.  Valid values for this property are <code>true</code> to



Property	Description
	<p>capture <code>System.out</code> output, or <code>false</code> to disable the capture.</p> <p>For more information, see <a href="#">System.out and System.err</a>.</p>
<code>log.wrapSysErr</code>	<p>Specifies whether you want to capture <code>System.err</code> output from your applications to the application trace file. The output is logged by VWS's root logger, which captures INFO level and above.</p> <p>Valid values for this property are <code>true</code> to capture <code>System.err</code> output, or <code>false</code> to disable the capture.</p> <p>For more information, see <a href="#">System.out and System.err</a>.</p>
<code>log.dump.level</code>	<p>Specifies the log-level of the entries that are captured in the in-memory buffer.</p> <p>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 <a href="#">Logback Architecture</a>.</p>
<code>log.dump.bufferSize</code>	<p>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.</p>
<code>log.dump.pattern</code>	<p>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 <a href="#">Layouts</a> in the Logback documentation.</p>

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

```
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 VWS 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 Virgo Web 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 VWS due to internal requirements.

The following listing shows the default `serviceability.xml` file in a freshly-installed VWS; 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</Pattern>
        </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" />
  </logger>
</configuration>
```

```

    <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 VWS 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 VWS 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 VWS itself are logged to the `SERVER_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 VWS creates the first log file, it calls it `log.log`; however, when this file reaches a size of 10MB, VWS creates a new log file called `log_1.log`, and so on up to 4. At that point, the cycle starts again and VWS 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 VWS. 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 VWS 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

Virgo Web 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 VWS configuration files, the `tomcat-server.xml` file is located in the `$SERVER_HOME/config` directory.

Here's an extract of the default configuration distributed with the VWS.

```
<?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="org.eclipse.virgo.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="org.eclipse.virgo.kernel.authentication.User"
        roleClassNames="org.eclipse.virgo.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="org.eclipse.virgo.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, VWS 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 `org.eclipse.virgo.server.web.tomcat.ServerLifecycleLoggingListener` class is specific to Virgo Web 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 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 define 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. Virgo Web 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 Virgo Web 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 Virgo Web 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

Virgo Web 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 Virgo Web 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 Virgo Web Server searches the searchpaths when it looks for dependencies; the first path listed specifies the first actual directory that Virgo Web 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 Virgo Web 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 VWS: `ext` and `usr`. Virgo Web Server searches each of these searchpaths when locating entries for inclusion in the repository.

The `chain` property shows the order in which Virgo Web 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
<code>repository-name.type</code>	<p>Specifies the type of path. You can set this property to one of the following three valid values:</p> <ul style="list-style-type: none"> <li>• <code>external</code>: Specifies that this path points to a number of directories that satisfy a given search pattern and are local to the current Virgo Web Server instance. Use the <code>searchPattern</code> property to specify the</li> </ul>



Property	Description
	<p>directory search pattern.</p> <ul style="list-style-type: none"> <li>• <b>watched:</b> Specifies that this path points to a single directory, local to the current Virgo Web Server instance. Virgo Web Server regularly scans watched repositories so it automatically picks up any changes to the artifacts in the directory at runtime. Use the <code>watchDirectory</code> property to specify the watched directory and the <code>watchInterval</code> property to specify how often VWS checks the directory.</li> <li>• <b>remote:</b> Specifies that the path points to a remotely-hosted repository, hosted by a remote instance of Virgo Web Server. Use the <code>uri</code> property to specify the full URI of the remote repository. You can also specify the optional <code>indexRefreshInterval</code> property.</li> </ul> <p>See <a href="#">Watched or External Repository?</a> for additional information about when to configure watched or external repositories for your particular environment.</p>
<code>repository-name.searchPattern</code>	<p>Specifies the pattern that an external repository uses when deciding which local directories it should search when identifying artifacts. Use this property together with <code>repository-name.type=external</code>. See <a href="#">Search Paths: Additional Information</a> for detailed information about specifying a search pattern.</p>
<code>repository-name.watchDirectory</code>	<p>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 VWS installation (<code>SERVER_HOME</code>). Use this property together with <code>repository-name.type=watched</code>.</p>
<code>repository-name.watchInterval</code>	<p>Specifies the interval in seconds between</p>

Property	Description
	<p>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 <code>repository-name.type=watched</code>.</p>
<code>repository-name.uri</code>	<p>Specifies the URI of the hosted repository to which a remote repository connects. The value of this property takes the following format:</p> <pre>http://host:port/org.eclipse.virgo.server.repo</pre> <p>where:</p> <ul style="list-style-type: none"> <li>• <i>host</i> refers to the computer on which the remote VWS instance hosts the remote repository.</li> <li>• <i>port</i> refers to Tomcat listen port of the remote VWS instance which hosts the remote repository.</li> <li>• <i>remote-repository-name</i> refers to the name of the remote repository, as specified in the <code>hostedRepository.properties</code> file of the remote VWS instance.</li> </ul> <p>Use this property together with <code>repository-name.type=remote</code>.</p>
<code>repository-name.indexRefreshInterval</code>	<p>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.</p> <p>Use this property together with <code>repository-name.type=remote</code>.</p>

## Should I Configure a Watched or External Repository?

The main difference between a watched and an external repository is that Virgo Web Server regularly scans watched directories and automatically picks up any changed artifacts, while VWS scans external directories only at startup, and then only if there is no cached index available. This means that VWS 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 VWS 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, we recommend 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 VWS 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 VWS 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, we recommend 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 Virgo Web 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, Virgo Web 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 Virgo Web 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 VWS, 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/org.eclipse.virgo.server.repository/my-hosted-repo`; this means that there is a VWS 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 VWS 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/org.eclipse.virgo.server.repository/my-hosted-repo
remote-repo.indexRefreshInterval=30

watched-repo.type=watched
watched-repo.watchDirectory=repository/watched
watched-repo.watchInterval=5

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

## 9.5 Configuring a Hosted Repository

You configure a VWS 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-repo` 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
<code>repository-name.type</code>	<p>Specifies the type of path of the hosted repository. All paths are local to the current VWS instance. You can set this property to one of the following valid values:</p> <ul style="list-style-type: none"> <li><code>external</code>: Specifies that this path points to a number of directories that satisfy a given search pattern. Use the <code>searchPattern</code> property to specify the directory search pattern.</li> <li><code>watched</code>: Specifies that this path points to a single directory. Virgo Web Server regularly scans watched repositories so it automatically picks up any changes to the artifacts in the directory at runtime. Use the <code>watchDirectory</code> property to specify the actual watched directory and the <code>watchInterval</code> property to specify how often VWS checks the directory.</li> </ul> <p>See <a href="#">Watched or External Repository?</a> for additional information about when to configure watched or external repositories for your particular environment.</p>
<code>repository-name.searchPattern</code>	<p>Specifies the pattern that an external hosted repository uses when deciding which local directories it should search when identifying artifacts. Use this property when <code>repository-name.type=external</code>. See <a href="#">Search Paths: Additional Information</a> for detailed information about specifying a search pattern.</p>
<code>repository-name.watchDirectory</code>	<p>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 VWS installation (<code>SERVER_HOME</code>). Use this property when <code>repository-name.type=watched</code>.</p>
<code>repository-name.watchInterval</code>	<p>Specifies the interval in seconds between checks of a watched directory by a watched hosted repository. This property is optional.</p>

Property	Description
	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 VWS have the general syntax `<XXnnnnL>` where:

XX	is a two-letter code (upper-case) identifying the region of the VWS 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):

AG	<code>org.eclipse.virgo.kernel.agent.dm</code>
CC	<code>org.eclipse.virgo.kernel.services.concurrent</code>
DE	<code>org.eclipse.virgo.kernel.deployer.core</code>
HD	<code>org.eclipse.virgo.kernel.deployer.hot</code>
KD	<code>org.eclipse.virgo.kernel.dm</code>
KE	<code>org.eclipse.virgo.kernel.core</code>
KS	<code>org.eclipse.virgo.kernel.services</code>
OF	<code>org.eclipse.virgo.kernel.osgi</code>
RP	<code>com.springsource.repository</code>
SH	<code>org.eclipse.virgo.kernel.shell</code>
UR	<code>org.eclipse.virgo.kernel.userregion</code>

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 [Eclipse Issue Tracker for the Virgo Runtime Environment](#). The issues are organised 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 VWS 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 VWS 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 Virgo Web 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

[Virgo Web Server Programmer Guide](#)

[Spring Framework Reference Guide](#)

[Spring Dynamic Modules Reference Guide](#)

[The Logback Manual](#)

