

# LiteWebServer 3

## Base Module Reference Manual

2003-03-31, Version 1.0.2

<b>INTRODUCTION</b> .....	<b>3</b>
<b>LICENSES</b> .....	<b>3</b>
<b>CONTACT INFORMATION</b> .....	<b>3</b>
<b>FEATURES</b> .....	<b>3</b>
<b>INSTALLATION AND OPERATION</b> .....	<b>4</b>
Installing LiteWebServer.....	5
Starting and Stopping LiteWebServer .....	6
Script Environment Variables.....	7
<b>TESTING THE INSTALLATION</b> .....	<b>7</b>
<b>INSTALLING WEB APPLICATIONS</b> .....	<b>8</b>
Sharing Classes Between Web Applications.....	9
<b>JUSTGETIT MODULE MANAGER APPLICATION</b> .....	<b>10</b>
Proxy Settings .....	10
Creating an Account.....	11
Logging In .....	12
Listing Installed Modules.....	12

<b>Uninstalling Modules.....</b>	<b>13</b>
<b>Installing or Upgrading Modules .....</b>	<b>13</b>
<b>Checking Installation Status .....</b>	<b>15</b>
<b>Manually Restarting LiteWebServer.....</b>	<b>16</b>
<b>CONFIGURATION OPTIONS.....</b>	<b>17</b>
<b>Server Configuration.....</b>	<b>17</b>
<b>Context Configuration .....</b>	<b>19</b>
<b>Default web.xml Settings For All Applications .....</b>	<b>20</b>
<b>ENABLING HTTPS.....</b>	<b>20</b>
<b>Installing JSSE.....</b>	<b>20</b>
<b>Generating a Self-Signed Certificate.....</b>	<b>21</b>
<b>Configuring LiteWebServer for HTTPS .....</b>	<b>21</b>
<b>HTTPS web.xml Security Constraints Declarations .....</b>	<b>22</b>
<b>REMOVING OPTIONAL JAR FILES .....</b>	<b>23</b>
<b>SECURITY NOTES.....</b>	<b>23</b>
<b>EMBEDDING LITEWEBSERVER IN ANOTHER APPLICATION .....</b>	<b>24</b>
<b>APPENDIX A. COM.GEFIONSOFTWARE.LWS.STARTUP.BOOTSTRAP .....</b>	<b>25</b>

Thank you for choosing LiteWebServer™ (LWS), a small and easy-to-use Java web server with native support for the Servlet API. It's small size (roughly 1 MB or less, depending on Java version and optional features) makes it ideal for embedding in other Java applications, for bundling with web application demos sent to potential customers, and for running web applications distributed on CDs. It's also ideal for Java web application development, personal use or a small intranet.

This Reference Manual describes how to install, configure and use LiteWebServer. For information about how to develop servlets and JSP pages, please use the resources listed on Sun's Servlet and JSP pages, <http://java.sun.com/products/servlet> and <http://java.sun.com/products/jsp> respectively. We also recommend Jason Hunter's *Java Servlet Programming* (O'Reilly) and Hans Bergsten's *JavaServer Pages* (O'Reilly) if you want to learn more about these technologies.

## Introduction

LiteWebServer (LWS) is a pure Java web server with native support for the Java Servlet API. To keep the server as small and simple to use as possible, it's designed as a base module plus a number of add-on modules that can be installed separately to add features to the server. One such add-on module adds support for JavaServer Pages (JSP). Add-on modules can also add authentication, access logging, and other things that are not needed for all environments, for instance when LWS is used as an embedded server. If you need a feature that's currently not available as an add-on module, please let us know.

## Licenses

LWS is licensed under a BSD-style open source license. Briefly, this means that you can use the product any way you want as long as you keep all copyright notices and include a note about where it comes from if you redistribute LWS with other software. See the *license.txt* file in the installation directory for details.

The LWS Base Module includes software developed by the Apache Software Foundation (<http://www.apache.org/>). More specifically, it is based on the Apache Tomcat server. See the Apache Software License, Version 1.1 (*Tomcat-license.txt*) in the installation directory for licensing terms with respect to the bundled ASF code.

Bundled with LWS is another Gefion Software product, named LiteXMLParser™ (LXP). LXP is a small JAXP compliant XML SAX 2 parser, based on the MinML2 parser developed by John Wilson (<http://www.wilson.co.uk/>). See the *MinML2-license.txt* file in the installation directory for the MinML2 licensing terms.

## Contact information

You can reach Gefion Software through one of the following mail addresses.

[info@gefionsoftware.com](mailto:info@gefionsoftware.com) for general questions or comments about the company and the products.

[support@gefionsoftware.com](mailto:support@gefionsoftware.com) to get help or report a problem with a product. Ideas about new features or other improvements are of course also welcome.

[sales@gefionsoftware.com](mailto:sales@gefionsoftware.com) for questions about licenses.

Visit our web site at <http://www.gefionsoftware.com/> for up-to-date information.

## Features

The LiteWebServer Base Module offers the following main features:

- HTTP/1.1 compliant web server.

- HTTPS support with Sun's JSSE package.
- Java Servlet 2.3 compliant web container.
- Auto-deploy of web applications stored in a special directory.
- Runs web applications directly from Web Archive (WAR) files, reducing the disk space requirements.
- Automatic reload of modified application classes can be enabled per web application.
- GUI-based installation program.
- JustGetIt™ module manager web application for easy module installation and upgrade.
- Server configuration through Java properties files.
- Flexible API for embedding in a custom Java application.

Add-on modules provide additional features, such as JSP 1.2 support. For an up-to-date list of modules, see our web site.

## Installation and Operation

LiteWebServer (LWS) requires a Java 2 Standard Edition (J2SE, at least 1.3 but we recommend that you use the latest version available for your platform) compatible Java VM. If you don't have one already you can download a Java VM from <http://java.sun.com/j2se/downloads.html>. If you plan to install the JSP add-on module at some point, you need the SDK version (includes a Java compiler). Otherwise you can install the smaller JRE version.

When you have installed the Java VM, verify that *java* command is recognized. On a Windows platform you can do so by opening an MS-DOS Command Prompt window and type this command:

```
C:\> java -version
```

If the command results in the message "Bad command or file name", you must set the PATH environment variable to include the Java *bin* directory. See the *Setting Environment Variables in Windows* sidebar if you don't know how to do this.

### Setting Environment Variables in Windows

You can set an environment variable in an MS-DOS Command Prompt window with the *set* command:

```
C:\> set PATH=c:\j2sdk1.4.0\bin;%PATH%
```

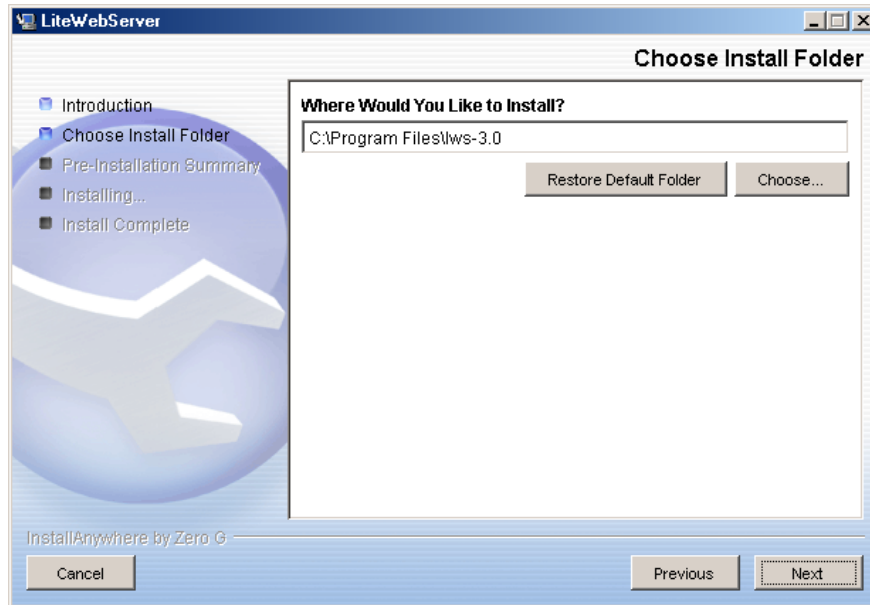
This value, however, is only available in the window where you run the command. How to set the variable so it's available to all windows and survives a reboot depends on the Windows version you use.

On a Windows 95/98/ME system, you add the *set* command with the syntax shown above to the *C:\AUTOEXEC.BAT* file. Use any text editor, such as Notepad, to add the line with the *set* command. The next time you boot the PC, the environment variables will be set automatically.

For Windows NT you can set them permanently from the Environment tab in the System Properties tool, and for Windows 2000 and Windows XP you can do the same with the System tool by first selecting the Advanced tab and then Environment Variables.

## Installing LiteWebServer

When you have installed Java and verified that the `java` command is recognized, download the LWS installer for your platform from <http://www.gefionsoftware.com/LiteWebServer/> and run it. It's a GUI-based installer for most platforms and it asks you for where to install LWS, which Java installation to use, and (on Windows only) where to place shortcuts to the LWS start and stop commands plus the uninstaller:



If you want to use JavaServer Pages (JSP), make sure you select a Java Software Development Kit (SDK) installation when asked for which Java to use, as opposed to a plain Java Runtime Environment (JRE). Otherwise LWS is not able to automatically compile your JSP pages.

The installer creates the LWS home directory (referred to as `$LWS_HOME` in some parts of this document) with these subdirectories:

*bin*

Contains scripts for starting and stopping LWS

*common*

Contains subdirectories for classes that are available to both LWS and all web applications.

*conf*

Contains configuration information files. You can edit these files to adjust how LWS works as described in the *Configuration* section.

*etc*

Contains files with information about the installed modules.

*logs*

The default LWS log directory. If something doesn't work as expected, always check the logs in this directory to see what's going on.

*server*

Contains subdirectories for the classes used internally by LWS. Web applications do not have access to these files and you must not modify its content.

*shared*

Contains subdirectories for classes that are available to all web applications but not to LWS.

*temp*

The default directory for Java temporary files.

*webapps*

The default directory for auto-deployed web applications.

*work*

The default directory for LWS and web application temporary files.

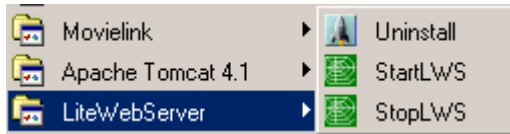
After a fresh install, the *webapps* directory contains a default application (the *ROOT* directory) with LWS information. You can replace the files in this directory with the web application you want to use as the default application. It also contains the JustGetIt module manager application (the *jgi.war* file). You can initially use this application to ensure that the installation was successful. We advice you to keep this application to manage your add-on modules, but if you don't need it you can simply remove the WAR file. When you have verified that the installation went smoothly, add your own applications as described in the *Installing Web Applications* section.

## Starting and Stopping LiteWebServer

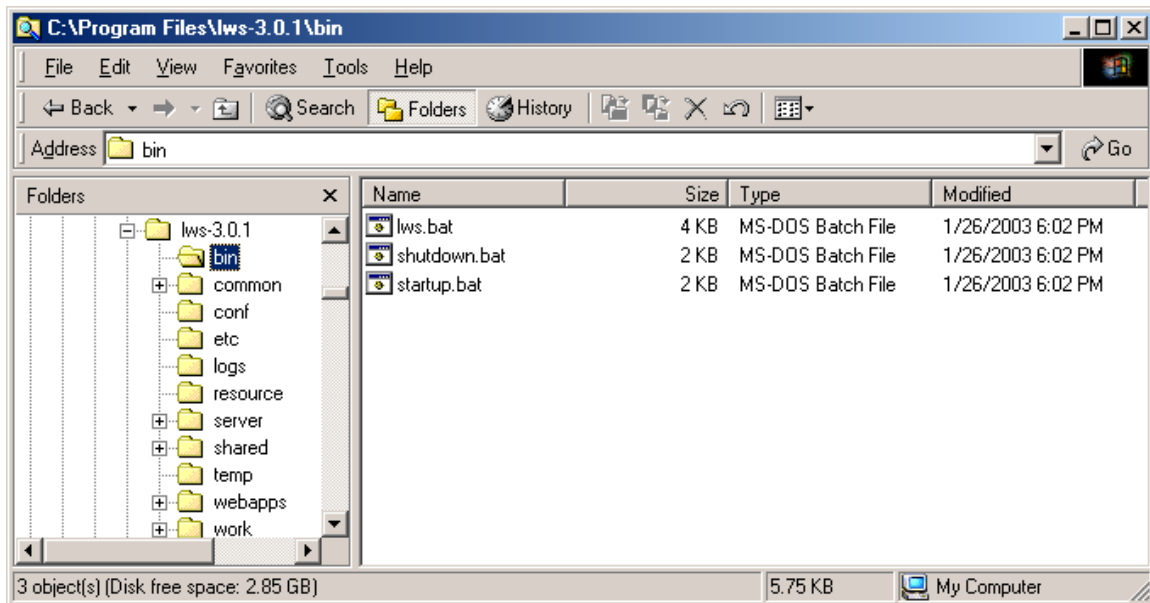
To start and stop LWS, you use the scripts in the *bin* directory or the shortcuts created by the installer.

### Windows Systems (Windows 98/ME/XP/NT/2000)

On a Windows system, you can use the two shortcuts named *StartLWS* and *StopLWS* created by the by the installer to start and stop LWS:



Alternative, just double-click the *startup.bat* and *shutdown.bat* files in the Windows Explorer:



When you start LWS, an MS-DOS Command Prompt window opens. This where LWS runs so you must not kill it, but you can minimize it if you want.

As an alternative to the shortcuts, you can run the scripts from the command line in an MS-DOS Command Prompt window. The syntax is exactly the same as for Unix (as described in the next section) with exception for the script names.

### Unix (Linux, Mac OS X, Solaris, FreeBSD, or any other Unix-like system)

To start LWS, run the *startup.sh* script:

```
[hans@frontier hans] lws-3.0.1/bin/startup.sh
```

This script starts LWS in the background and writes all startup info to *\$LWS\_HOME/logs/lws.out*.

To stop LWS, run the *shutdown.sh* script:

```
[hans@frontier hans] lws-3.0.1/bin/shutdown.sh
```

Alternatively, you can run the *lws.sh* script. It supports the following command line options:

```
lws.sh [start | stop | run] [-debug]
```

**start**

Starts LWS in the background and captures `stdout` and `stderr` in `$LWS_HOME/logs/lws.out`.

**stops**

Stops LWS.

**run**

Starts LWS in the foreground. Use Ctrl-C to stop it.

**-debug**

Writes debug information to `stdout` for the start-up phase. Debug information while the server or a context runs can be controlled through configuration properties, as described in the *Configuration* section.

## Script Environment Variables

The scripts on all platforms can optionally be controlled through a number of environment variables:

Environment Variable	Description
LWS_HOME	Set this variable if you want to run the scripts without first setting current directory ( <code>cd</code> ) to the LWS installation directory or <i>bin</i> directory.
JAVA_HOME	Set this variable if you want to use a different Java installation than the one in the <code>PATH</code> .
JSSE_HOME	Set this variable if you're using a Java version prior to 1.4 and have not installed JSSE as a standard extension (see <i>Enabling HTTPS</i> below).
JAVA_OPTS	Set this variable to Java options that you want to use, for instance like this to specify a proxy for the JustGetIt module manager when you run behind a proxy: <code>-Dhttp.proxyHost=myproxyserver -Dhttp.proxyPort=8080</code> Scripts for other Java programs besides LWS sometimes use this variable name as well, so use it only for options that apply to all Java programs.
LWS_OPTS	This variable is used the same way as <code>JAVA_OPTS</code> but is unique for LWS, so you can use it for LWS-specific options.
CLASSPATH	The scripts do not change the <code>CLASSPATH</code> variable or specify an explicit classpath for the <i>java</i> command. All classes in your <code>CLASSPATH</code> are therefore available to LWS and all web application. We recommend that you leave it empty and use the <code>LWS_HOME/common</code> and <code>LWS_HOME/shared</code> directories for classes like this instead.

## Testing the Installation

To test that the installation is successful, enter this URL in a browser when you have started LWS:

```
http://localhost:9090/
```

The server name *localhost* only works if you run the browser on the same computer as LWS; replace it with the real server name if this is not the case. The port number 9090 is the default for LWS. You can change this in the configuration files as described in the Configuration section. If you set it to 80 (the HTTP default), you don't have to include it in the URL.

If the Java runtime and LWS are installed correctly, you get a welcome page with links to documentation and the JustGetIt module manager. Use the JustGetIt application to verify that everything works as expected before you install your own applications or change the configuration.

## Installing Web Applications

LWS supports the web application structure defined by the Servlet 2.3 specification. Briefly, a web application consists of any number of files and subdirectories containing web pages (HTML, XML or JSP files) and other resources such as images. It also contains a special directory named *WEB-INF*. Here's an example of a simple web application structure:

```
index.html
images/
  logo.gif
products/
  overview.html
lws.html
lwp.html
WEB-INF/
  web.xml
  classes/
    com/
      gefionsoftware/
        app/
          testServlet.class
lib/
  app.jar
```

For a servlet or JSP application, the *WEB-INF* directory can contain a *web.xml* file with application configuration information and the subdirectories *classes* and *lib* for Java classes used by the application. Class files that are packaged in JAR files go in the *lib* directory while all other class files go into subdirectories under *classes* that mirror the package structure for the class. The whole structure can be packaged in a JAR file with a *.war* file extension; this is called a Web Application Archive (a WAR file). A WAR file is compressed, so on platforms where disk space is an issue, a WAR file is a good choice.

Each web application is represented by a servlet context and is identified by a unique *context-path*. The context-path is used as a prefix in URLs for application resources like HTML pages, servlets and JSP pages. For instance, the URL for the *index.html* file in an application with the context-path */example* looks something like this:

```
http://www.mycompany.com/example/index.html
```

The easiest way to install a web application is to just create a subdirectory for it (or place a WAR file) in the LWS *webapps* directory. The context-path for an application installed this way is the name of the subdirectory (or the WAR file minus the *.war* extension). Note that you must create a *WEB-INF* directory for the application, but you can leave it empty if you don't need any special *web.xml* configuration or class files.

There's one exception to the rule about context-paths; one application, the default application, is assigned an empty string as its context-path, and URLs for resources in this application do not have any prefix. For instance, to get the *index.html* file in the default application, you use a URL like this:

*http://www.mycompany.com/index.html*

The default application is installed in the *webapps/ROOT* directory (or the *webapps/ROOT.war* file).

You can also tell LWS to load applications from any readable directory by editing the configuration files as described in the *Configuration* section.

## **Sharing Classes Between Web Applications**

If you have many web applications, you may want to install common class in one place instead of copies in the *WEB-INF* directory for each application. LWS supports two ways for doing this.

The recommended way is to store the shared classes in the LWS *shared* subdirectories. You can install class files into the *shared/classes* directory (or subdirectories mirroring the package structure) and JAR files into the *shared/lib* directory. If the same class is present in both the *classes* subdirectory and a JAR file in the *lib* subdirectory, the class in the *classes* subdirectory is used.

The second way is to set the `CLASSPATH` environment variable to include the shared classes. The LWS startup scripts makes all classes defined in the `CLASSPATH` available to all web applications.

# JustGetIt Module Manager Application

The JustGetIt module manager web application bundled with LiteWebServer (LWS) makes it a snap to add and remove add-on modules as well as upgrade the modules you have installed. It's installed with a context path of */jgi*, so to access it, type this URL in your browser's address field (assuming LWS is running on a server named frontier with the default port 9090):

```
http://frontier:9090/jgi/
```

The JustGetIt application communicates with a web service running on our server, so you must have an Internet connection to use it.

## Proxy Settings

If the computer where LWS runs communicates with Internet servers through a proxy, you must specify a couple of Java system properties to allow the connections to tunnel through the proxy. For a regular HTTP proxy, the properties you must set are named:

```
-Dhttp.proxyHost=ProxyHostName  
-Dhttp.proxyPort=ProxyPortNumber
```

If you're behind a SOCKS proxy, you must use these properties instead:

```
-DsocksProxyHost=ProxyHostName  
-DsocksProxyPort=ProxyPortNumber
```

For more information about these and other network properties you can set, see <http://java.sun.com/j2se/1.4.1/docs/guide/net/properties.html>.

If you use the scripts to start LWS, you can set either the JAVA\_OPTS or the LWS\_OPTS environment variable (see *Script Environment Variables*) to include these properties:

```
JAVA_OPTS=-Dhttp.proxyHost=myproxy.com -Dhttp.proxyPort=80
```

The Windows shortcuts do not read environment variables. If you start LWS using a shortcut, you must edit the shortcut command line instead. Select the *StartLWS* shortcut in the Settings, advanced Task Bar window and choose Properties from the right mouse button menu to edit the command:

```
"C:\Program Files\Java\j2sdk1.4.0\bin\java.exe" -Dhttp.proxyHost=myproxy.com  
-Dhttp.proxyPort=80-jar "server\lib\bootstrap.jar" -start
```

## Creating an Account

In order to use the application, you must log in with a username and password. This screen is displayed the first time you access the application.

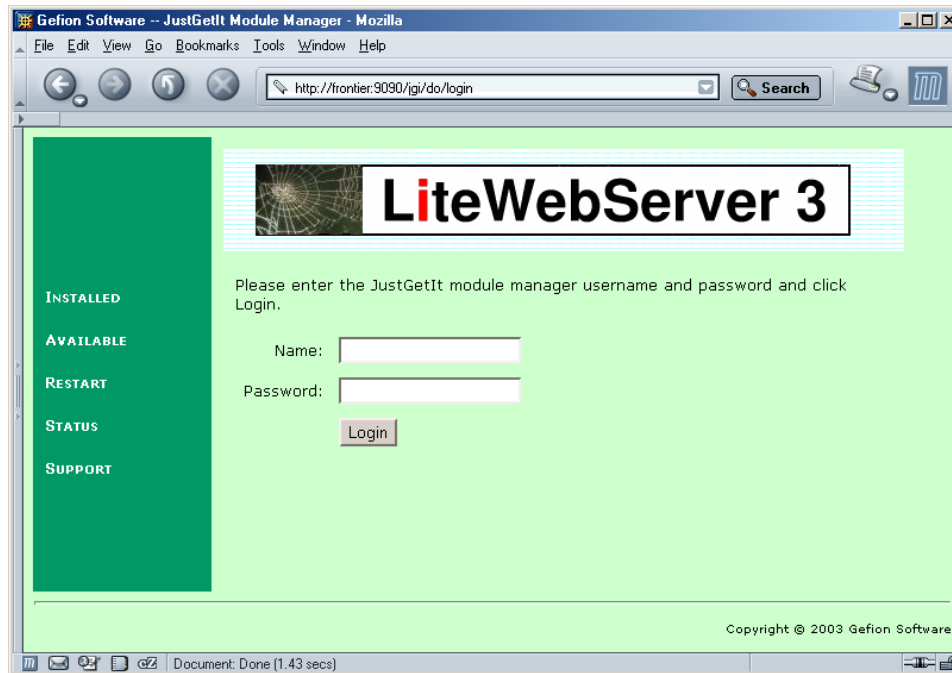


Enter the account username and password you want to use for the application and click the Create button to create it. The username and password is saved in an encrypted format in the file *conf/jgiacc.properties*. If you ever want to replace the account information, just remove this file, restart LWS and access the application again.

When the account is created, you're automatically logged in to the application. After 15 minutes of inactivity, you're automatically logged out.

## Logging In

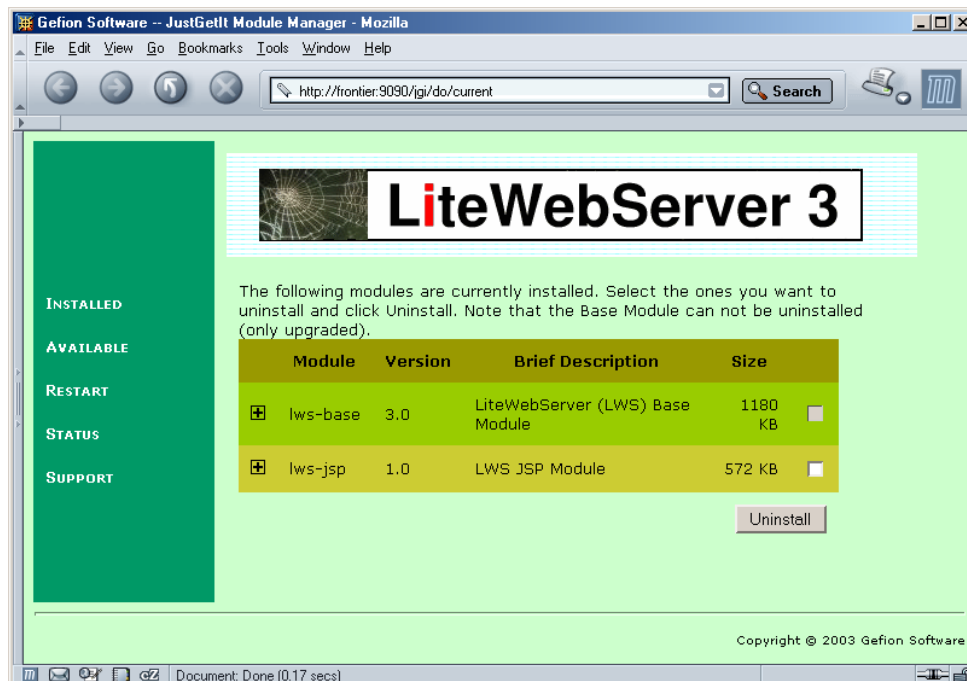
When you try to access the application without being logged in, you're prompted for the account username and password.



Enter the username and password for the account you created earlier and click the Login button to login.

## Listing Installed Modules

The main screen is shown after you've logged in, or when you click Installed in the menu. It shows a list of all currently installed modules.



For each module, the module ID, the version number, a brief description and the amount of space the module takes up on the disk are listed. Clicking on the plus sign adds a more detailed description and a link to the page on our web site with additional information, such as the complete documentation and release notes.

## Uninstalling Modules

To uninstall one or more modules, display the list of installed modules (by clicking Installed in the menu) and click the checkboxes to the right to select the ones to be uninstalled. When you click Uninstall, a confirmation screen is displayed. Clicking Uninstall on the confirmation screen uninstalls all selected modules and restarts LWS to unload all classes for these modules. When the restart is complete, an updated list of installed modules is displayed.

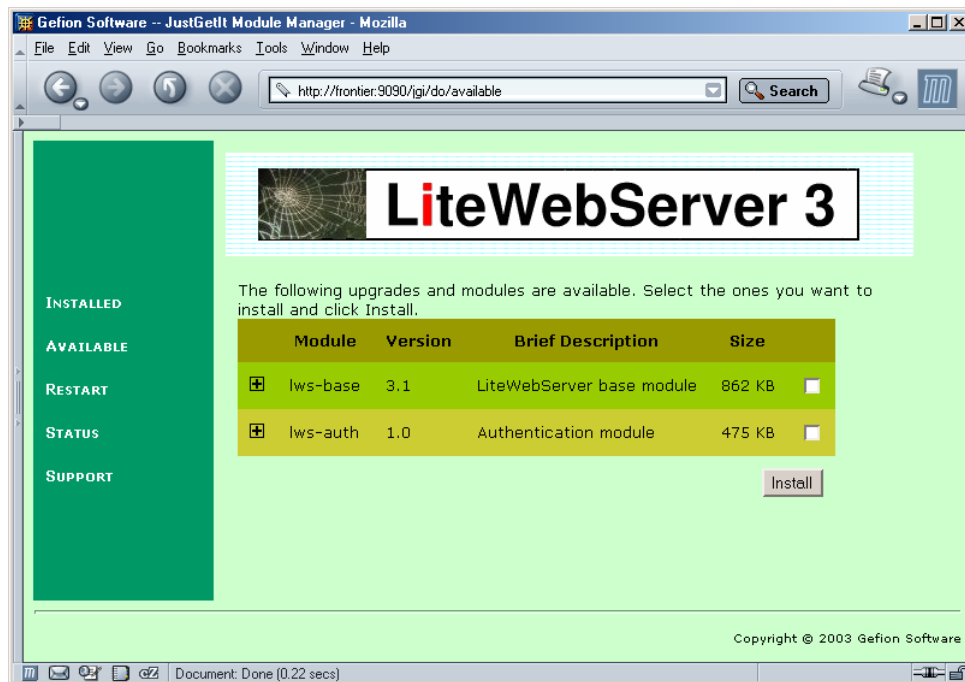
---

**Note!** Uninstalling a module on a Windows platform doesn't delete the module's JAR files due to file locking problems (see Sun's Java bug database, bug ID 4755278, for details. As a work-around, the content of each JAR file is replaced with a dummy entry. This reduces the size and ensures that the uninstalled classes disappear, but to really get rid of the files you need to delete the JAR files manually. The *etc* directory contains a *properties* file for each module with a list of all files to delete. Look at this file *before* you uninstall the module, since uninstall removes it.

---

## Installing or Upgrading Modules

When you click Available in the menu, the application contacts our JustGetIt web service to get a list of all available modules and lists all add-on modules and upgrades that you do not have installed.

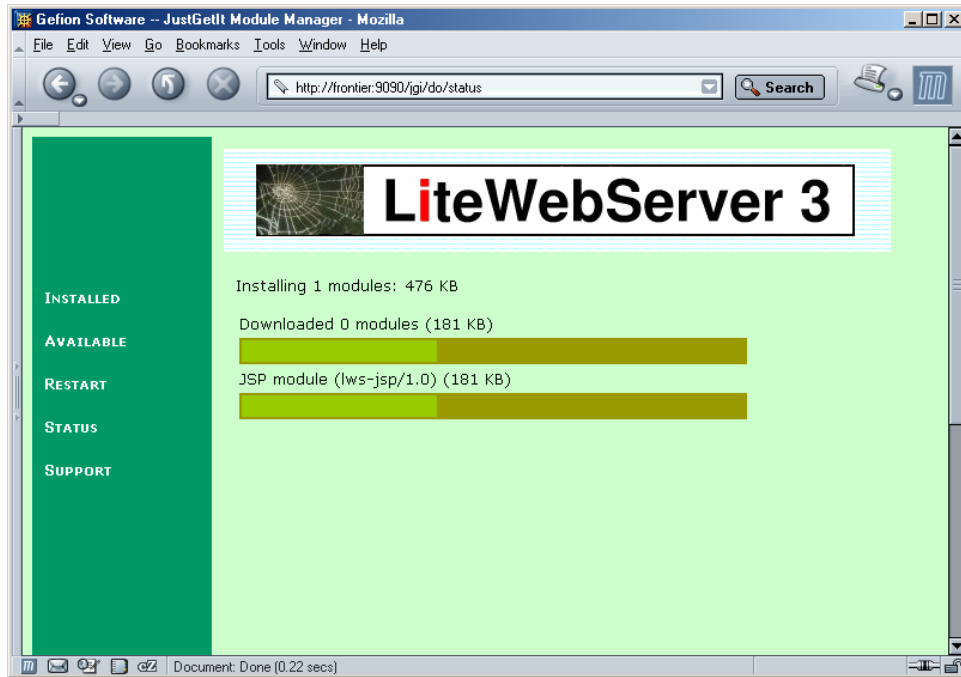


For each module, the module ID, the version number, a brief description and the size of the module download file. Clicking on the plus sign reveals more detailed description and a link to the page on our web site with additional information, such as the complete documentation and release notes.

To install one or more modules, click the checkboxes to the right to select the ones you want. When you click Install, a confirmation screen is displayed. It lists all modules you selected plus modules the selected

modules depend on, if any. Click Install on the confirmation screen to start downloading and installing all modules.

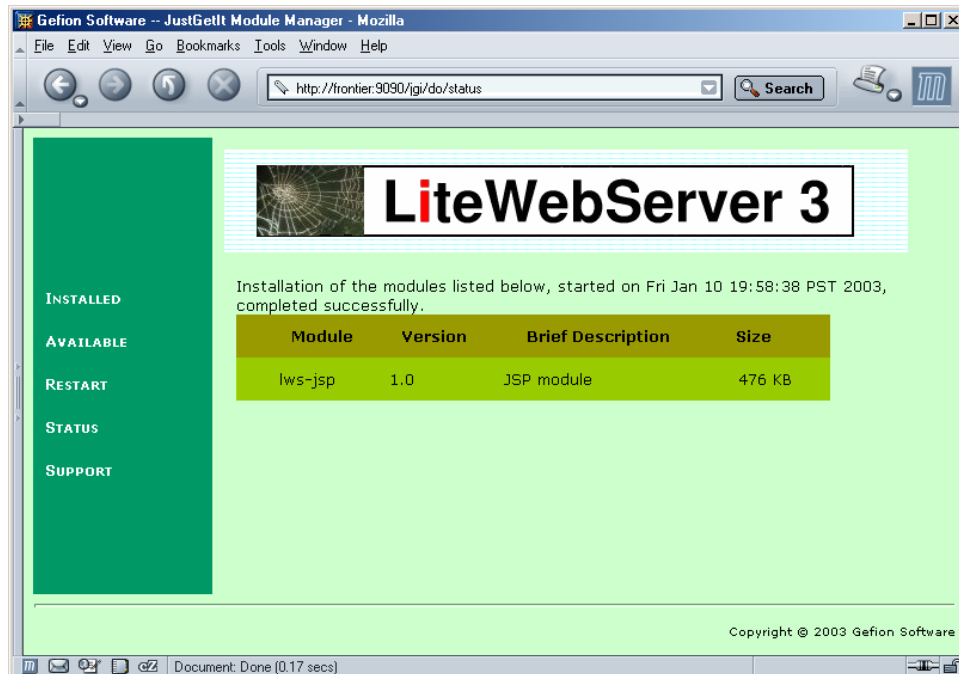
During the installation process, a status screen with progress bars is displayed (refreshed every 2 seconds).



When the installation is completed, LWS is automatically restarted to load all classes for the new modules and an installation result screen, shown in the next section, is displayed.

## Checking Installation Status

Clicking Status in the menu brings up the installation status screen. If an installation is in progress, the same screen as above is displayed until it's done. Otherwise a screen with information about the latest install is displayed.

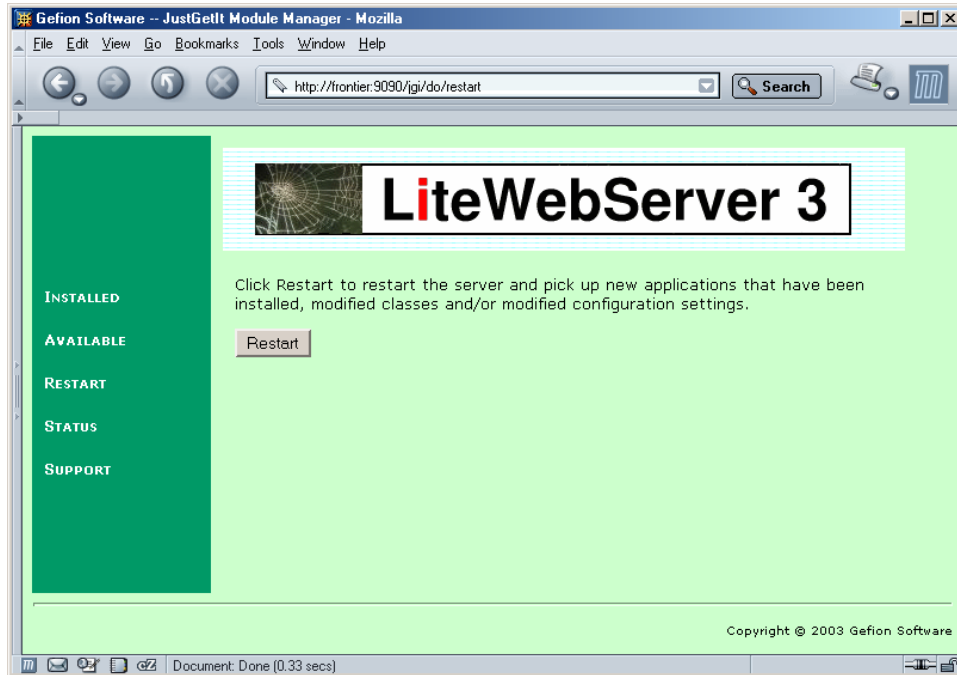


If the installation was successful, the screen contains a list of all installed modules. After the list, a number of messages may be displayed, alerting you to things like new configuration options or other things you need to be aware of.

If the installation failed, a message that described what went wrong is displayed before the list of modules. In most cases, the problem is temporary, so try again.

## Manually Restarting LiteWebServer

Installing and uninstalling modules automatically restarts LWS, but sometimes you may need to manually restart LWS. Examples are to get configuration changes to take effect, loading modified web application classes (when you haven't configured them for auto-reload), or to clear the application scope data. To restart LWS manually, click Restart in the menu.



A confirmation screen is displayed. Click the Restart button to start the restart. When the restart is completed, the list of installed modules is displayed.

# Configuration Options

Configuration options can be specified for the LWS server and for individual web applications (contexts). The configuration files are Java properties files; text files with a key and a value on each line, separated by an equal sign. Comment lines start with a hash mark (#) and you can break long lines by escaping the line feed character with a back slash (\) in all files.

To set a property, make a copy of the property comment, remove the comment hash mark (#) and add a value:

```
# server.webapps.unpackwars=<true-or-false>
server.webapps.unpackwars=true
# If set to true, any WAR file in the directory specified by
# server.webapps.root are unpacked in the same directory at startup.
# If set to false, only the class files (WEB-INF/classes and WEB-INF/lib)
# are extracted to a subdirectory under the directory specified by
# server.workdir.
# Default is false.
```

You must restart LWS after changing the configuration file for the changes to take effect.

## Server Configuration

In most cases, the default configuration is appropriate, but you can adjust a number of features for LWS by editing the `conf/server.properties` file. The following table describes all properties you can set:

Property	Default	Description
<code>server.backlog</code>	100	The maximum number of incoming HTTP connection requests that should be queued. If a connection request arrives when the queue is full, the connection is refused.
<code>server.conntimeout</code>	20000 (20 seconds)	The number of milliseconds to wait for data on a connection.
<code>server.debug</code>	0	A positive number, indicating the amount of debug output to produce (the higher the number, the more output).
<code>server.dnslookups</code>	true	If true, the server attempts to find the DNS host name when a servlet calls <code>request.getRemoteHost()</code> . If false, this call returns the string representation of the remote host's IP address instead, for better performance.
<code>server.host</code>	All IP addresses for the host where LWS runs	The host name or IP address that receives incoming HTTP/HTTPS requests. If you specify a host name, the first IP address returned for this name by the OS is used.
<code>server.https.enable</code>	false	Set to true to enable HTTPS connections. See the <i>Setting up HTTPS Support</i> section for additional setup requirements.
<code>server.https.keystore</code>	<code>\$HOME/.keystore,</code>	The file path to the keystore file that

Property	Default	Description
	where \$HOME is the home directory of the user account running LWS	contains the Certificate used to identify the server on an HTTPS connection. An absolute path or a path relative to \$LWS_HOME can be used. See the <i>Setting up HTTPS Support</i> section for details.
server.https.password	changeit	The HTTPS keystore and Certificate password (must be the same for both). See the <i>Setting up HTTPS Support</i> section for details.
server.https.port	9443	The port number the server shall listen to for HTTPS requests. See the <i>Setting up HTTPS Support</i> section for details.
server.logdir	\$LWS_HOME/logs	The file path to the directory for log files. An absolute path or a path relative to \$LWS_HOME can be used. Set to the empty string to disable logging.
server.port	9090	The port the server listens for HTTP requests on.
server.shutdown.command	SHUTDOWN	The string used as the shutdown command issued by the shutdown script.
server.shutdown.port	9005	The port for shutdown requests.
server.threadmax	100	The max number of threads that shall be created. When requests are received that cannot be immediately served by an available thread, new threads are added up to this max number.
server.threadadmin	10	The number of request handling threads to create when the server is started.
server.webapps.autodeploy	true	If true, web applications found in the server.webapps.root directory are deployed automatically at start-up.
server.webapps.root	\$LWS_HOME/webapps	The file path to the directory where web apps (WAR files or directories with WAR structure) can be placed for auto-deploy. An absolute path or a path relative to \$LWS_HOME can be used.
server.webapps.unpackwars	false	If set to true, any WAR file in the directory specified by server.webapps.root are unpacked in the same directory at startup. If set to false, only the class files ( <i>WEB-INF/classes</i> and <i>WEB-INF/lib</i> ) are extracted to a subdirectory under the directory specified by server.workdir.

Property	Default	Description
<code>server.workdir</code>	<code>\$LWS_HOME/work</code>	The file path to the root work directory for all contexts. A sub directory to this directory is made available to servlets in each context through the <code>javax.servlet.context.tempdir</code> context attribute.  An absolute path or a path relative to <code>\$LWS_HOME</code> can be used. The context's work directory can be used for temporary files, such as servlets generated from JSP pages.
<code>context.context-name.confdir</code>	Default is to automatically create contexts with default options for web applications in the <code>server.webapproot</code> directory.  See the <i>Context Configuration</i> section for details about the context options.	The file path to the configuration directory for the context named by <i>context-name</i> .  An absolute path or a path relative to <code>\$LWS_HOME</code> can be used. This directory must contain a <i>context.properties</i> file with special configuration options for the context.

You must restart LWS after changing the configuration for the changes to take effect.

## Context Configuration

Typically, web applications are installed in the directory specified by the `server.webapps.root` property in *server.properties* and runs with default values for all properties. To customize the configuration for a context, or load it from any readable directory, you can create a *context.properties* file in a directory defined by the `context.context-name.confdir` property in the *server.properties* file. For instance, with this property in *server.properties*:

```
context.myapp.confdir=conf/myapp
```

you define the context configuration in the *conf/myapp/context.properties* file.

The following table describes all LWS context configuration properties:

Property	Default	Description
<code>context.approot</code>	A directory with the same name as the context under <code>\$LWS_HOME/webapps</code>	The file path to the directory with the web application files. An absolute path or a path relative to <code>\$LWS_HOME</code> can be used.
<code>context.autoreload</code>	<code>false</code>	If <code>true</code> , the context is restarted when modified class files are detected (or <i>web.xml</i> is modified), automatically loading the new version.
<code>context.crosscontext</code>	<code>false</code>	If <code>true</code> , <code>getServletContext()</code> returns a reference to another context in the same server. If <code>false</code> , the method

Property	Default	Description
		always returns null.
<code>context.debug</code>	0 (no debug output)	A positive number, indicating the amount of debug output to produce (the higher the number, the more output).
<code>context.path</code>	The name of the context with a slash prepended.	The context path for the context (the URL prefix used for all resources in the application). Must start with a slash.
<code>context.usecookies</code>	true	If true, cookies are used for session tracking by default, falling back to URL rewriting if no cookies are received from the client. Setting this property to false forces URL rewriting to always be used.

You must restart LWS after changing the configuration file for the changes to take effect.

## Default web.xml Settings For All Applications

The LWS `conf/web.xml` file contains standard context settings that are applied to all web applications, such as definitions of MIME types. You can modify the defaults by editing this file. Settings in the local `web.xml` file are added to the defaults. You must restart LWS after changing the configuration for the changes to take effect. See the Servlet 2.3 specification or a servlet or JSP book for details about the `web.xml` file and all things you can configure with elements in this file.

## Enabling HTTPS

HTTPS means HTTP over a more secure connection using a low-level protocol named Transport Layer Security (TLS), formally known as Secure Socket Layer (SSL). TLS (and therefore HTTPS) provides encryption and message integrity checks for the data sent over the connection to ensure that a third party cannot easily record or modify the data.

To use HTTPS, two things are needed: software that handles the encryption and integrity checks, and a certificate that identifies the server and contains the encryption keys. LWS uses Sun's JSSE package to provide HTTPS connection support. JSSE also includes command line tools for generating or importing Certificates. The following sections describe all you need to do to enable HTTPS in LWS.

## Installing JSSE

The Java 2 Standard Edition SDK 1.4 and later includes JSSE, so if you use a recent SDK version, you can move on to the *Generating a Self-Signed Certificate* section. For SDK versions prior to 1.4, you must download JSEE separately from <http://java.sun.com/products/jsse/>. Follow the instructions on Sun's site.

The JSSE JAR files can be installed as a so-called *installed extension* by placing them in the `$JAVA_HOME/lib/ext` directory (see the JSSE documentation for details), but can also be installed in any directory you like. If you do not install them as an installed extension, you must set the `JSSE_HOME` environment variable to the JSSE installation directory path. On a Windows platform, use the `set` command like this:

```
C:\> set JSSE_HOME=C:\jsse1.0.3_01
```

Look at the description of how to permanently set the `JAVA_HOME` environment variable in the *Installation* section if you're not sure how to set an environment variable permanently.

## Generating a Self-Signed Certificate

In order to establish HTTPS connections, you need a certificate for LWS. You can use either a self-signed certificate or a certificate signed by a trusted third party. A self-signed certificate is usually good enough if you only intend to use HTTPS to provide encrypted communication for sensitive data. If it's critical that the client can authenticate the server (e.g. in a b2b application), you may instead want to use a certificate signed by a trusted third party instead, see the Sun's JSSE documentation for how to generate a .

The `keytool` bundled with the Java SDK is used to generate a self-signed certificate. Use the command with these options (Windows syntax, adjust as needed for Unix platforms):

```
C:\> %JAVA_HOME%\bin\keytool -genkey -alias lws -keyalg RSA
```

The command prompts you for various bits of information.

When asked for a keystore password, enter `changeit` if you want to use the LWS default value. If you enter some other password, you will have to tell LWS about it as described later.

When asked for your first and last name, enter the server name used in the URLs when accessing LWS, e.g. `www.mycompany.com`. If you enter some other value, users will be warned that the server name doesn't match the certificate owner when trying to access your LWS server through HTTPS.

Enter any appropriate values for the organizational unit and name, city, state and country code.

Finally, when asked for the key password, you *must* use the same password as for the keystore. This is the default, so just hit Return. If you enter a password that is different than the one for the keystore, LWS will be unable to use the certificate.

If you entered the command exactly as shown earlier, the self-signed certificate is stored in a file named `.keystore` in the current user's home directory with a default expiration date. LWS looks for the certificate in a file with this name in the home directory of the account that it runs as by default, so if you generate the certificate with the same account as you use for LWS, you're all set. You can optionally use the `-keystore` option to specify another file name and the `-validity` option to adjust how long the certificate is valid, see the Java SDK tools documentation for details. If you use a different name, you must tell LWS about as described in the next section.

## Configuring LiteWebServer for HTTPS

When you have created the keystore, you must enable HTTPS in LWS. All HTTPS configuration is done by specifying properties in the `conf/server.properties` file.

Enable HTTPS by setting the `server.https.enable` property to `true`:

```
server.https.enable=true
```

If you created the keystore in a different file than the default (`.keystore` in the home directory for the LWS account) or specified another password than `changeit`, you must specify the values you used with the following properties:

```
server.https.keystore=/home/lws/mykeystore
server.https.password=secret
```

By default, LWS uses port 9443 for HTTPS. If you want to use a different port, such as the standard HTTPS port (443), you can set the following property:

```
server.https.port=443
```

After you have made these changes, you must restart LWS.

## HTTPS web.xml Security Constraints Declarations

With HTTPS enabled, you can declare that access to certain resources in a web application must only be access through HTTPS. You do this with the `<security-constraint>` element in the `web.xml` file for the application:

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!DOCTYPE web-app
  PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
  "http://java.sun.com/dtd/web-app_2_3.dtd">

<web-app>
  ...
  <security-constraint>
    <web-resource-collection>
      <web-resource-name>secured</web-resource-name>
      <url-pattern>/protected/*</url-pattern>
    </web-resource-collection>
    <user-data-constraint>
      <transport-guarantee>CONFIDENTIAL</transport-guarantee>
    </user-data-constraint>
  </security-constraint>
  ...
</web-app>
```

The `<security-constraint>` element contains one or more `<web-resource-collection>` elements that define the resources the constraint applies to. The nested `<web-resource-name>` element gives the collection a name and one or more `<url-pattern>` elements specify the resource paths. The `<user-data-constraint>` element and its nested `<transport-guarantee>` element define the type of connection can be used to access the resources in the collection. The transport guarantee value must be either `CONFIDENTIAL`, `INTEGRAL` or `NONE`. Either one of `CONFIDENTIAL` or `INTEGRAL` means they must be accessed through HTTPS; `NONE` means both HTTP and HTTPS are accepted.

In addition to a URL pattern, you can also specify that the security constraint only applies to resources accessed with one or more HTTP(S) methods. This example shows how to declare that all resources accessed with a path starting with `/protected` and the `POST` method must use HTTPS:

```
...
<security-constraint>
  <web-resource-collection>
    <web-resource-name>secured</web-resource-name>
    <url-pattern>/protected/*</url-pattern>
    <http-method>POST</http-method>
  </web-resource-collection>
  <user-data-constraint>
    <transport-guarantee>CONFIDENTIAL</transport-guarantee>
  </user-data-constraint>
</security-constraint>
...
```

With HTTPS enabled, when a user tries to access a resource matching a security constraint like this using HTTP, LWS sends a redirect response to the browser telling it to try again using HTTPS. If HTTPS is not enabled in :LWS, a response with status code 403 (Unauthorized) is returned instead.

## Removing Optional JAR Files

If you're running on a J2SE 1.4 platform, you don't need the LiteXMPParser, because this version of the Java platform includes a JAXP enabled parser. You can therefore remove the following JAR files from the LWS *common/lib* directory: *jaxp-api.jar*, *lpx.jar*, *MinML2.jar* and *sax.jar*.

As described in the *Security Notes* section, the generic servlet invoker feature is disabled by default. If you don't need to enable it, you can remove the *servlet-invoker.jar* file from the LWS *server/lib* directory.

Static files are served by a default servlet, but if your applications don't have any static files or define their own default servlets, you can also remove the *servlet-default.jar* file from the LWS *server/lib* directory.

If you remove both the *servlet-invoker.jar* file and the *servlet-default.jar* file, you can also remove the *servlet-common.jar* file.

## Security Notes

The LiteWebServer (LWS) base module does not provide any means for restricting access to a web resource. Anyone who can access the server port where LWS is running can access all web pages and other resources served by LWS unless you implement your own security mechanism, for instance with special servlets or filters. If you want to let LWS handle authentication, you can install the authentication add-on module and define the authentication requirements in the *web.xml* file for the application, as defined by the Servlet specification.

If you've used other web containers, you may be used to invoking servlets with a URL that includes a */servlet* path followed by the name of the servlet, like */servlet/HelloWorld*. This behavior is disabled by default in LWS because of security concerns (it's a feature that has proved to be vulnerable in the past and it's possible that new ways to abuse it are found in the future). The safest way to invoke a servlet is by defining a URL mapping for the servlet in the *web.xml* file:

```
...
    <servlet>
        <servlet-name>HelloWorld</servlet-name>
        <servlet-class>com.mycompany.HelloWorldServlet</servlet-class>
    </servlet>
    ...
    <servlet-mapping>
        <servlet-name>HelloWorld</servlet-name>
        <url-pattern>/hello/*</url-pattern>
    </servlet-mapping>
    ...
```

With this mapping in place, you use a URL like */hello* to invoke the servlet.

If you must use */servlet* type URLs, you can enable the feature by adding this mapping in the *web.xml* file for the application, or in the LWS *conf/web.xml* file to enable it for all applications (not recommended):

```
...
    <servlet-mapping>
        <servlet-name>invoker</servlet-name>
        <url-pattern>/servlet/*</url-pattern>
    </servlet-mapping>
    ...
```

## Embedding LiteWebServer in Another Application

LWS can be embedded within another Java application. You just need to create an instance of the server bootstrap class and start, restart and stop it as needed. If you've embedded a previous version of LWS, you must modify your code to use the new API since the internals of the server has changed dramatically in 3.0.

This is an example of the code you put in your application:

```
import com.gefionsoftware.lws.startup.Bootstrap;
import java.net.*;
import java.io.*;
import java.util.*;

public class MyApp implements Runnable {
    private Bootstrap server;

    public static void main(String[] args) throws Exception {
        MyApp myApp = new MyApp();
        myApp.startServer();
    }

    public void startServer() {
        // Create an instance of LWS and run in a new Thread
        server = Bootstrap.getInstance();
        Thread runner = new Thread(this);
        runner.start();

        // Do whatever the app is doing
        ...

        // Restart LWS (in response some external event, typically)
        server.restart();
        ...

        // Stop LWS
        server.stop();
    }

    public void run() {
        // Start LWS. Does not return until stopped
        server.start();
    }
}
```

The `com.gefionsoftware.lws.startup.Bootstrap` class handles creation of the LWS server instance, and starting, restarting and stopping the instance. The `getInstance()` method returns the single instance of the `Bootstrap` class. You then call the `start()`, `restart()` and `stop()` methods on this instance to control the server.

By default, the LWS instance is configured based on the `conf/server.properties` file. If you want to configure it based on information from another source, add the properties described in the *Server Configuration* section to a `java.util.Properties` instance and call the `setCustomServerProperties()` method with this instance:

```
Properties conf = new Properties();
conf.setProperty("server.port", "7070");
conf.setProperty("server.debug", "10");
server.setCustomServerProperties(conf);
server.start();
```

Properties set this way survive a restart of the server. If you change them and restart the server, the new properties are used. See *Appendix A* for a description of all the public `Bootstrap` methods.

All you need to include in the classpath from LWS when you compile your application is the *server/lib/bootstrap.jar* file:

```
javac -classpath $LWS_HOME/server/lib/bootstrap.jar\:$CLASSPATH MyApp.java
```

You must also include the *bootstrap.jar* file in the classpath and specify the LWS home directory with the `-Dlws.home` flag when you start your application, like this:

```
java -classpath $LWS_HOME/server/lib/bootstrap.jar\:$CLASSPATH -Dlws.home=$LWS_HOME MyApp
```

The `Bootstrap` class creates classloaders and loads all other LWS classes from the *server*, *shared*, and *common* subdirectories of the LWS home directory, plus the *WEB-INF/classes* and *WEB-INF/lib* directories for each web application. If you change the content of any of these directories and call `restart()`, the old classloaders are dumped and new ones created, with the result that the new classes are used by the restarted LWS instance.

## Appendix A. `com.gefionsoftware.lws.startup.Bootstrap`

The following methods are used to create and control an LWS instance embedded in another application.

### `getInstance`

```
public static com.gefionsoftware.lws.startup.Bootstrap getInstance()
```

Returns the single instance of this class.

### `setCustomServerProperties`

```
public void  
setCustomServerProperties(java.util.Properties customServerProperties)
```

Set the server properties to use instead of the ones defined in the server configuration file.

### `start`

```
public void start()
```

Start as an embedded server. This method does not return until the server is stopped.

### `start`

```
public void start(boolean debug)
```

Start as an embedded server and get debug messages for the start-up phase. This method does not return until the server is stopped.

### `stop`

```
public void stop()
```

Stop the embedded server.

### `restart`

```
public void restart()
```

Restart the server. Use by either an embedded server or the server itself. First a flag is set to indicate a restart is in progress. The server is then shutdown, causing its `process()` method to return to the `startServer()` method in this class. The flag is checked, and if set, all class loaders are replaced and the server is started again.