

## CHAPTER

## 1

# Installing JBuilder

Understanding the installation process, file locations, configuration files, and advanced settings can help a developer establish a stable and usable environment that will support the development effort. Borland JBuilder uses the InstallAnywhere installation program from Zero G to handle the install process; so many typical installation problems are minimized. The most daunting task removed was installation on multiple platforms. InstallAnywhere provides easy installation on the Mac, Windows, Linux, and Solaris platforms. Installation can be attempted for unsupported platforms, but it requires additional work and is not supported in any way by Borland.

The installation process itself is fairly painless, but developers installing the product should follow up with the following tasks:

1. Install everything first: Web services, Borland Enterprise Server, MobileSet, OptimizeIT.
2. Review what was installed and where it was installed.
3. Review the configuration files.
4. Set the user.home.
5. Review the licensing procedure.
6. Set additional switches.
7. Perform troubleshooting.

## 4 Chapter 1

---

### Installing JBuilder

---

Because JBuilder is a Java program, it is always recommended that it be installed on a noncompressed drive with ample free space. JBuilder should be installed and used on a machine with as much memory as can be afforded. A fast processor is important, but when using Java-based programs, memory is even more critical. The base machines that Borland recommends will allow JBuilder to run fairly decently; however, if at all possible, having a machine with 512MB more memory will really make the product perform like native development environments. Later in this chapter, we recommend the proper levels of JVM switches to get the most out of your environment.

The installation process itself is rather painless and will require the standard information about what components are to be installed and where the files should be located. The installation process normally will be started from a CD; the appropriate executable on Windows or the script on other platforms should be executed to begin the installer program. The install process will then proceed to load a JVM into memory and start a step-by-step wizard to complete the process. Once the install process is complete, the developer will have an opportunity to install additional programs from the installer, if necessary. Once all programs have been installed, the developer can exit the installer program.

Make sure to install all programs that may be part of the JBuilder system first before starting the program because of the licensing procedures that Borland is now using to register the products, which is covered later in the chapter. Additional features can be added to the JBuilder environment from the Borland Enterprise Server, Web Services pack, MobileSet, and OptimizeIT; however, they do not need to be installed for JBuilder to work. For example, if the developer were not going to be programming J2ME, the usefulness of JBuilder's MobileSet would be limited.

JBuilder was designed to be completely extensible, meaning that if a function or feature was missing inside JBuilder, then any Java developer could add an OpenTool without having to recreate the entire product. The concept of OpenTools is not new; it is nothing more than a standardized plug-in for the environment. JBuilder's implementation of an OpenTools framework is much more complete and documented than other OpenTools frameworks. In fact, the JBuilder program itself is only about 3K in size, with its major responsibility being to understand how to load and run OpenTools, meaning that everything in JBuilder is an OpenTool. Tools like MobileSet are nothing more than a set of OpenTools that give new functionality to JBuilder.

Borland uses this framework to introduce new products and features to JBuilder. Most of these new features can be found at [www.borland.com/jbuilder](http://www.borland.com/jbuilder) and are available for free download. Two examples given thus far have been MobileSet and the Web Services pack OpenTools, each giving distinct functionality. When it came to Web Services, Borland wanted to give the users of JBuilder a preview of what features it was planning to add to the product. The standards that were going to be used were not yet complete from a specification standpoint, though, so Borland could not add the features to JBuilder because it would have been introducing a nonstandard implementation that could have made JBuilder unstable. Borland still wanted the feedback from

developers to make sure that what was being developed was on target, so it made the Web Services pack available from the download site to let developers interested in Web Services to download it and try it out. Once the specifications were completed, the Web Services pack was then added to the standard build of JBuilder Enterprise. This gives Borland the best of both worlds, exposing new technologies that might not be ready for the real world or of interest to only a small portion of developers while still getting valuable information.

Other programs that may be installed include Borland's Enterprise Server (J2EE 1.3 compliant server) included in the Enterprise Edition, Together, StarTeam, and OptimizeIt, which is included in the Studio product. Because these are additional technologies, they may need to be registered with Borland, so installing all of them can save time in the registration process. Plus, the installer may want to control where licenses are located on the machine.

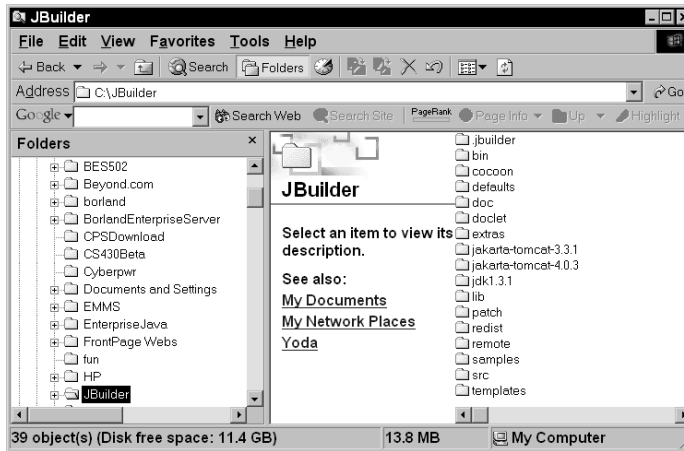
## Review the Installation

After everything has been installed, understanding file locations and their meanings helps the developer understand how JBuilder works. Understanding the directory structure or file extensions will help a JBuilder developer understand what is being held in each file and where. As the developer becomes more knowledgeable with the product, this underlying information will become more important. Projects will continue to get larger and more complex, so having a firm understanding of the environment will allow the developer to understand how to configure JBuilder to handle these situations better. Understanding the configuration files that JBuilder uses to load itself into memory can help developers create advanced configurations for resource-starved machines or set switches so that the painting mechanism that JBuilder uses works better with the underlying hardware and operating system.

Figure 1.1 shows the directories located under the JBuilder directory. This directory structure will be the same no matter what OS you install JBuilder on. All versions of JBuilder will follow this directory layout; some older versions of JBuilder may not include all of the ones listed in Figure 1.1. Most of the directory structure is explained throughout this book; however, this chapter focuses on the `bin` and `.jbuilder` directories because they have the most to do with installing JBuilder.

The `bin` directory is where the JBuilder executables are located. Notice that all the files are either executables or configuration files. The executables `.exe` on the Windows OS and a single name with a MOD of 777 on Unix are nothing more than bootstrap programs that use the configuration file to define what should be run. This type of executable is the same that Native Executable supports in the build system, which is fully covered in Chapter 10. Inside this directory, notice that most executables have a corresponding `.config` file; this is where program specifics like classpath and memory setting are located. Now, there are three files in this directory to make note of. The first is `config_readme.html`, which is nothing more than an HTML page that explains how the configuration files are defined and what each switch means.

## 6 Chapter 1



**Figure 1.1** Directories installed on the Windows OS.

The next file is the `jbuilder.config`; it is responsible for holding all the settings for launching JBuilder using the bootstrap program called `jbuilder.exe` or `jbuilderw.exe` on the Windows OS. The only difference between `jbuilder.exe` and `jbuilderw.exe` is that if you call the `jbuilder.exe`, it will launch from a command or terminal window, and if `jbuilderw.exe` is used, it will not spawn a window. This becomes important if you need to see what JBuilder is loading or doing when executing; this topic is covered later in this chapter.

## Review the Configuration Files

The last file of interest in the `bin` directory is the `jdk.config`. It is one of the most important files in the directory because it is called by all the other configuration files. It can be thought of as an include file, which means that `jdk.config` can be used to make global settings for all JBuilder executables.

The `jdk.config` file includes the two following lines:

```
vmparam -Xminf0.2
vmparam -Xmaxf0.2
```

The `-Xminf0.2` represents the minimum ratio of free memory used in heap after Garbage Collection (GC), and the `-Xmaxf0.2` represents the maximum ratio of free memory used in heap after GC. The two settings can be found in the documentation for the Hotspot JVM. They also have long names associated with them. `-Xminf` also can be denoted as `-XX:MinHeapFreeRatio=<minimum>` on the command line and `-XX:MaxHeapFreeRatio=<maximum>`, respectively. These two settings are bound by the `-Xms` and `-Xmx` VM switches. The `-Xms` represents the initial memory size (heap) available to the program's memory pool. The `-Xmx` represents the maximum size the memory pool can get for a given program. This then means that `-Xminf` and `-Xmaxf` are percentages of the values found in the `-Xms` and `-Xmx` switches. These two can be worked with, depending on the amount of memory a given machine has available to it.

The current settings are for 20 percent both min and max. Documentation states that the max can be as high as 70, but this has not been tested. They are currently set in a generic form and are good for most machines. Machines with more memory may want to set these settings higher to get better performance. If performance becomes an issue, this is one area to keep in mind. You should also keep in mind that these switches are located in the `jdk.config` and will be used by all the executables found inside the `jbuilder/bin` directory. Setting these switches to optimize JBuilder performance may have adverse effects on the other programs that use these settings.

The `jbuilder.config` file also has memory switches defined; they are the `-Xms` and the `-Xmx` switches, and these should be set according to the amount of memory on the particular machine on which JBuilder will be executing. The defaults are as follows:

```
vmparam -Xms32m
vmparam -Xmx256m
```

Reviewing the definitions defined previously, these `vmparams` represent the base starting memory pool, which will be started with 32 megabytes and has a maximum of 256 megabytes. Because the machine being used for this book includes 512 megabytes of memory, the `-Xms` switch could be set to 128 and the `-Xmx` switch could be set to 384. The last character is important when using these switches:

- “m” represents megabytes
- “k” is used for kilobytes
- “g” represents gigabytes

More information on memory management can be found at the following sites:

- <http://java.sun.com/docs/hotspot/gc/index.html>
- <http://java.sun.com/docs/hotspot/VMOptions.html>
- <http://community.borland.com/article/0,1410,23022,00.html>

These are just some of the more useful links. JDK versions noted in the links are still valid for JDK 1.3.x and beyond. These techniques can be used with all Java programs, not just JBuilder.

Another set of switches found in the `jdk.config` file that can be very important to the way JBuilder performs are these:

```
# vmparam -Dsun.java2d.noddraw
# vmparam -Dsun.java2d.d3d=false
```

On certain hardware, notably machines with ATI graphics cards or machines using the Windows XP operating system, the preceding switches can be uncommented to fix the problem. Do not change these settings unless problems are occurring with JBuilder. A line beginning with “#” is a commented line; removing the “#” will make it uncommented. Normally, the `-Dsun.java2d.noddraw` is the switch most used with the ATI graphics card. The `-Dsun.java2d.d3d=false` switch is the one used for the XP operating system. Uncomment only one line at a time, then test the configuration by restarting JBuilder; if the results do not change, recomment the line, uncomment the other line, and repeat the process.

## 8 Chapter 1

---

**NOTE** Always exit JBuilder when making changes to one of its configuration files, as JBuilder will rewrite them on exit. Always save the file, exit, and restart JBuilder for changes to take effect.

### Set the user.home

---

When you start up JBuilder, the first thing that will pop up is a request to enter the license key. The license key will allow the program to be properly registered, so Borland can refine the product and contact the user if issues arise. Starting with JBuilder 5, Borland has taken steps to ensure that, at a very minimum, a registration process is followed to activate the product.

The location of the license information on a machine can be set manually. This helps long-term maintenance and ensures that the installer has complete control over the process. This also allows the installer to have all the needed files in one defined location. JBuilder and the tools included with it use license files to validate users; some tools like JDataStore need to have a license file with a valid key before it will work in a deployed situation, so knowing the location of these files can eliminate the guesswork of where they are located. Then, by placing the license files on the classpath, JBuilder and its tools will be able to find and execute without asking for revalidation. The switch that should be added to the `jbuilder.config` file is the following:

```
vmparam -Duser.home=[location]
```

The `[location]` part is important because this is where all the default files will be located for JBuilder, including the licensing file. The default project file and the internal configuration files are also held there after the initial install. The `[location]` could look like the following:

```
vmparam -Duser.home=\JBuilder\jbuilder
```

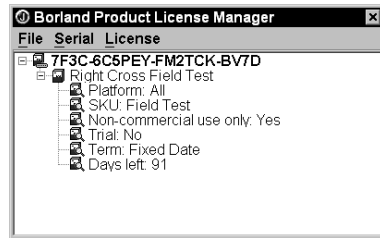
This means that all files would be located in the `.jbuilder` directory under the JBuilder directory. Once this location has been set, it cannot be moved or changed unless the JBuilder licensing process is followed again. This is another reason for making sure that the directory is where the installer needs it to be, and this is also why it is better to change the configuration files first before running and then having to reregister after the directory has been set.

### Review the License

---

JBuilder has a fairly simple licensing process. It can be changed or updated anytime to show additional programs that have been registered. The Licensing Information program has an icon on the Start menu in Windows, or its icon can be found in the JBuilder shortcut list in Unix. Typing the following line also can start it:

```
C:\JBuilder8\bin\JBuilderW.exe -license
```



**Figure 1.2** Licensing manager.

This can be done at the operating system command prompt, so make sure to have a defined JDK in the path. Figure 1.2 shows a license file for a Field Test product, which explains exactly what the program is supposed to be used for. Licenses come in many forms; normally the license is found on the CD case that holds the JBuilder product. License strings can be emailed or loaded from a file, so remember that anytime a license is changed, a connection to Borland through the Internet will most likely be needed. If an Internet connection is not available, then other licensing schemes can be used like email or even postal mail; these options can be discussed with Borland Technical Support if issues arise. Borland Product License Manager can be used to change or update licenses of Borland products. It is available in the JBuilder Program menu list.

## Set Other Switches

Most Java and VM switches can be used with JBuilder. Another often used switch is this one:

```
vmparam -Djava.io.tmpdir=\jbuilder\tmp
```

This is useful if a lot of IO is going to be done. This also helps JBuilder put in a Java caching area when it needs to find space. This is normally added to the `jbuilder.config` file; however, it could again be added to the `jdk.config` if the setting was going to be used by all the executable programs found in bin directory.

Another reported problem is that sometimes developers use multiple versions of a program. This is especially true when dealing with the Borland Enterprise Server. There are many ways to solve the problem; one is to change the classpath found inside the `jbuilder.config` file and use the `addjars` command to add any specific jars needed. This will mean, though, that each JBuilder configuration will have to be modified for each version of the software being used. In the older editions of JBuilder, before JBuilder 6 (that is, JBuilder 5 and below), the quickest and easiest way is to create a `.bat`, `.cmd`, or script file that will set all the settings for that particular version. The following is a script file used to point JBuilder to the latest `.jar` files to be used when using the Borland Enterprise Server:

## 10 Chapter 1

---

```

set path=
set classpath=
set path=c:\JBuilder\jdk1.4\bin;c:\JBuilder\jdk1.4\jre\bin;C:\
BorlandEnterpriseServer\bin
set classpath=%classpath%;c:\JBuilder\jdk1.4\jre\lib\rt.jar;c:\
JBuilder\jdk1.4\lib\tools.jar;C:\BorlandEnterpriseServer\lib\
vbjorb.jar;C:\BorlandEnterpriseServer\lib\vbjeb.jar
jbuilderw

```

This code sets the current path and classpath so that JBuilder uses the latest Borland Enterprise Server. This is critical when you want to point to one version of the Application Server and at another time point to a different version of the VisiBroker technology, as it ensures that JBuilder will not get confused.

**WARNING** The preceding technique works only with JBuilder 5 and below. Starting with JBuilder 6, these limitations were solved using the configuration files.

Two other switches that can be very helpful when trying to figure out why a problem is occurring when the JBuilder program loads are these:

```

-verbose
-info

```

These switches can be placed on the JBuilder command -line, and they will give additional information on configuration, classpath, and OpenTools being loaded during the load process.

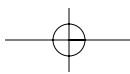
## Perform Troubleshooting

---

Whenever you make changes to any files, be sure to create a backup and exit JBuilder first. Any changes made while the JBuilder program is open most likely will not be saved.

When changing memory switches, make sure to test the configuration before beginning real programming. Nothing is more irritating than getting knee-deep in a problem and having JBuilder just crash or lock up with no warning.

Also keep in mind that custom configurations are probably not supported by Borland. It is good practice to make copies of the standard files so that the system can be reset to a default setting if problems arise.



## Summary

---

This chapter gave an overview of the process for installing the JBuilder product. It covered the proper steps to ensure a good install and highlighted some of the areas that can cause long-term problems with license files. The chapter also covered memory allocation, switches, and startup parameters to ensure that JBuilder works the best it can from the install point forward.

The next chapter discusses customizing the AppBrowser, which is JBuilder's main interface to the developer.

