Appendix C
Domino Setup for Writing Java Programs

Given that you have an installed and running Domino server or Notes Client, you're all set as far as *running* Java Applications or Agents that use the Notes Object Interface. But you'll need to do a bit of extra setup if you want to *write* any Java programs for NOI. Let's go over the basic system requirements.

System Requirements

You must use an operating system that supports Java, meaning one for which a Java Virtual Machine has been released. There are some Domino platforms for which no VM has yet been released, and there are others for which you can find a VM, but it has not been "certified" for Domino. The lack of certification shouldn't necessarily deter you, as I'll explain shortly.

Table C.1 summarizes the status of Java programming for each of the Domino platforms. Domino is compatible with Java version 1.1 and above only. Java 1.02 will not work with Domino NOI, although 1.02 Applets do work in the Notes Client and can be served up by the Domino HTTP server.

Table C.1 Java Availability For Domino 4.6 Platforms

Operating Platform	Java VM Available?	Java Certified for Domino?	Java Built for Domino?
Windows 32-bit (Win95 and NT; Intel only, not Alpha)	Yes	Yes	Yes
Windows 16-bit (Windows 3.1)	No	No	No
Mac	Yes	No	No
NLM	No	No	No
Solaris (SPARC)	Yes	Yes	Yes
Solaris (Intel)	Yes	No	No
HPUX	Yes	No	Yes
AIX	Yes	No	Yes

OS/2 v 2.2 and later	Yes	No	Yes
AS400	Yes	No	No
OS390	Yes	No	No

Just because there's no certified support for Java on a particular platform doesn't necessarily mean you can't use Java to program NOI on that platform. Usually certification of a feature on a platform simply means that Lotus has tested the feature on that platform and believes that it is working properly. It also means that Lotus provides technical support for that feature on that platform and will accept (and ultimately fix) problem reports.

Some of the UNIX platforms were built with all the Java support code included, but no formal testing was done. If you're willing to experiment and do without official tech support, there's no reason why you shouldn't try using NOI on a uncertified system. If, however, the Java support code was not built for a given platform (e.g., Macintosh, Win16, AS400), NOI won't be available on that platform, even if a Java VM is available from the system vendor. You won't kill your system by trying it; at worst, you'll get some error messages.

But, you ask, isn't Java a cross-platform language? Why can't I take the notes.jar file from a supported platform and use it on another one, as long as there's a Virtual Machine to run it in? You can try, but it won't work because the Java code in the NOI classes (distributed in a single jar file called notes.jar) all make use of calls in to C and C++ code in the Notes core. If the C/C++ interface wasn't built for a given platform, there's just no way to call into Notes from Java.

Development Tool Options

You can use all kinds of development tools to write Java programs that use the NOI classes, ranging from the free and basic to the expensive and sophisticated. The only tool that I've come across that I know *won't* work, for reasons explained elsewhere, is Microsoft's Visual J++. For purposes of illustration, I'll mention three tools that I have

used.. My mention of a particular tool does not constitute an endorsement of that tool or of that tool's vendor. I haven't made an exhaustive study of all the tools that are available, either.

The most basic, and in some sense the most authoritative, Java development tool is JavaSoft's Java Development Kit (JDK), available for free from the JavaSoft Web site (http://java.sun.com, see the References.nsf database on the CD-ROM for this and other links). It contains a compiler, an interpreter, the classes you need at runtime, and not much else. Additional tools, such as the Bean Development Kit (BDK), the Servlet Development Kit, and a kit for RMI are separately available as well, and they are also free.

If you use the JDK as the basis for your Java programming, you must supply your own text editor to type in the Java source code. Then you use the JDK tools (typically from a console window command line) to invoke the compiler and to run your program (unless it's an Agent, in which case you use Notes/Domino to run the program, as described in Chapters 8 and 9). A debugger comes with the JDK, but most people find it useless (I sure did). Still, the JDK is the authoritative, "reference" implementation of the language (on Win32 and Solaris, the platforms for which it's officially available from JavaSoft:), and code compiled using the JDK will (at least in theory) run in any other vendor's VM.

Two other development environments I've used are Borland's JBuilder and Symantec's Visual Cafe. Both provide approximately the same features: multiwindowed development and debugging environment; project orientation; code generation wizards to help you get going, particularly with Applet programming; and libraries of useful widgets. SunSoft also has an Integrated Development Environment (IDE) for Solaris, although I haven't tried it.

With a full-blown IDE, you typically use a wizard to generate the basic code for your program, or you can bring up an editor window and type some in yourself (or

take some existing code and add it to your current project). Then, entirely from within the tool's user interface, you can compile, run, and debug your Application or Applet. The debuggers are pretty nice. They allow you to step through your program line by line, set breakpoints and examine variables, and browse class hierarchies. Debugging Agents, of course, is another matter, as the Notes runtime has to be involved (see Chapter 9 for tips on how to debug Agents).

All of these tools require some tweaking before you can use them with Domino's NOI classes.

Path and Classpath Setup

Most people are familiar with the concept of a search path for executable programs: you can define an environment variable on most kinds of systems that contains a list of directories that the system will search for executable programs. This environment variable is usually called "PATH". For the Java interpreter a similar environment variable, called "CLASSPATH", tells it where to look for Java classes that it might need to find and load. Each tool has its own way tospecify these variables. The following sections describe how to set up each of the three tools I've mentioned to work with NOI. The exact representation of these environment variables is platform specific, so check the exact syntax ("CLASSPATH" vs. "classpath," for example) for the system you're using.

Note that Domino usually does not require you to add the Notes executable directory to your PATH, but you should do so for all systems on which you will be running Java Applications. The reason is that Java, when running a Domino NOI program, needs to find and load the Notes DLL (or platform equivalent) into which the Java classes want to call. If your Java program is not located in the Notes executable directory (the directory where most of the Notes files are installed), Java will be unable to run your program.

JDK

For the JDK, you simply set the CLASSPATH variable to include the notes.jar file. The installation instructions for the JDK will tell you how to set up the initial CLASSPATH. For example, on Windows 95 systems, you might add the following line to your autoexec.bat file:

```
set classpath=.;c:\jdk11\lib\classes.zip;c:\notes\notes.jar
```

The "classes.zip" file is a container file that has in it all the standard Java runtime packages (java.lang, java.util, java.io, etc.). You simply append the Notes container file, which includes lotus.notes package.

For Windows NT you could accomplish the same result by going to the control panel, bringing up the System tool, and selecting the **Environment** tab. Then you would enter the above line into either the System Variables section (if you wanted it to apply to all users of the machine) or in the personal section for the current user.

The initial "." in the example classpath above tells the VM to search the current directory for classes that need to be loaded. If you use this technique, you can use any directory on your system to develop your Java code. Or, you can name your Java source directory explicitly.

JBuilder

When you install the JBuilder tool, it sets up its own CLASSPATH setting and will ignore your CLASSPATH environment variable, if you have one. That's because JBuilder, like most Java development tools, comes with its own copy of the Java VM and runtime classes. Therefore it wants to maintain its own CLASSPATH variable.

To use JBuilder successfully with NOI, you have to tell it where the Notes classes are. To do that, you can just edit the Jbuilder.ini file that lives in the JBuilder\bin subdirectory. There's a line in there for ClassPath and also one for IDEClassPath. I couldn't find any documentation of how the two differ, but I just appended my notes.jar

path to both. Also, because I stored all the Java source files that I used for my samples (they're all on the CD) in a single directory on my system, I appended that directory as well. That way I could "add" my source files to a JBuilder project and compile and debug them without having to store them in a JBuilder subdirectory. For the IDE to find my source files, I also had to modify the SourcePath entry.

The modified IDEClassPath and Classpath entries in my JBuilder.ini file look like this:

```
IDEClassPath=..\lib\jbuilder.zip;..\lib\jbcl.zip;..\lib\jgl.
zip;..\java\lib\classes.zip;\notes\notes.jar;\lotus\work\wor
dpro\book\examples
SourcePath=..\myprojects;..\java\src.zip;..\src\jgl-
src.zip;..\src\jbcl-
src.zip;\lotus\work\wordpro\book\examples
ClassPath=..\lib\jbcl.zip;..\lib\jgl.zip;..\java\lib\classes
.zip;\notes\notes.jar;\lotus\work\wordpro\book\examples
```

Visual Cafe

Visual Cafe, like JBuilder, installs its own Java VM and runtime class library. Also like JBuilder, it has its own .ini file that keeps track of its own set of environment variables. You'll have to edit the file bin\sc.ini (where bin is a subdirectory in the Visual Cafe installation tree), and append your notes.jar path to the CLASSPATH entry. You should also add your Notes directory to the PATH entry. The two modified lines on my system look like this:

```
PATH=%@P%\..\BIN;%@P%\..\Java\Bin;%PATH%;\notes
CLASSPATH=.;%@P%\COMPONENTS\SYMBEANS.JAR;%@P%\..\JAVA\LIB;%@
P%\..\JAVA\LIB\SYMCLASS.ZIP;%@P%\..\JAVA\LIB\CLASSES.ZIP;\no
tes\notes.jar;\lotus\work\wordpro\book\examples
```

Developing Domino Agents and Servlets

Although Notes does not (yet) include a Java development environment, it does (at runtime) use its own Java VM and, like most development tools, it will ignore a CLASSPATH set up in your system's environment in favor of its own. Notes/Domino, of course, will already have included the notes.jar file on its internal CLASSPATH. If you want to make sure Domino's VM will also find your Servlet or other Java class files, you need to append the appropriate directories to the JavaUserClasses variable in your notes.ini file.

You can also set the size of the Java VM's internal heap and stack using the JavaMaxHeapSize and JavaStackSize variables (see Chapter 8 for details).

Supported Java Versions

The Domino Notes Object Interface in release 4.6 was developed using the 1.1.1 release of the Java Development Kit. By the time 4.6 shipped (September 1997), JavaSoft had posted a couple of additional maintenance (bug-fixing) releases, named 1.1.2, 1.1.3, and 1.1.4. Programs you build with any of these releases should work just fine with Domino. If you use version 1.0.2 of the JDK to compile a program it may or may not work for NOI programs. Version 1.0.2 is probably safe for developing Applets that you want Domino to serve up to browsers.

You should keep in mind that JavaSoft will typically post versions of Java only for Win32 and Solaris. Other platform versions of the VM are supplied by individual vendors. IBM, for instance, licenses Java source code from JavaSoft and supplies Lotus with all the VM implementations that ship with Domino. Thus implementation of a particular release of Java for a particular platform might lag a bit, depending on how quickly the vendor can build, test, and release it after receiving the code from JavaSoft.