Chapter 1

Domino/Notes Programmability Overview

Believe it or not, this is a big topic all by itself. If we use a broad definition of the word *programmabilty*, we have to mention LotusScript, cover the Notes Object Interface (NOI), and at least touch on each of the ways in which LotusScript and Java programs can be formulated (agents vs. applications vs. servlets vs. applets vs. CGI), triggered (foreground, HTTP, scheduled, new mail), where each kind of program can run (client, server, both), and how each can be edited and debugged.

Were we to cover all these topics in depth, you'd be holding an encyclopedia instead of a book. Still, we can at least categorize, compare, and contrast to give you an overview of which technology you might want to use when and where.

The Notes Object Interface (NOI)

We'll be talking a lot in this book about NOI: how to use it and what you can do with it. In fact, there are five whole chapters that document every call in the Java version of the API. For now, we'll say only that NOI is a class library, that is, a group of objects that you can program using LotusScript or Java (this book focuses on the Java interface) that manipulate the Domino/Notes groupware development platform.

If you are already a Notes/Domino user, you're familiar with such product concepts as servers, databases, views, and documents. NOI provides a programming interface to those objects. As of Domino Release 4.6, those objects are available in Java. Previously, they were available only through the embedded language known as LotusScript.

The term *Notes Object Interface* is not language specific. It refers to both the Java and LotusScript APIs. The objects underneath, however, are the same. The differences in the way you use them have mostly to do with syntactical differences in the two languages.

In the discussions that follow, some familiarity with object-oriented programming concepts is assumed. If you're already a LotusScript or C++ programmer, you'll have no trouble at all.

The rest of this chapter is about the kinds of programs people write for Domino and for Internet-based applications in general. Let's start with the different kinds of programs you can write for and in Notes/Domino and define them a bit.

Applications

For the purposes of this book, we'll use the term *application* to mean any program that you start from a system-level icon or from a command line. Applications are self-contained and reside somewhere on your computer's disk. The Domino server is an example of an application, as is a Solitaire card game. Applications almost always have some kind of user interface (UI) with which you interact. A UI can be a simple command line or a full-blown windowing environment.

Computer programs, or *applications*, can be developed in any programming language: C, COBOL, FORTRAN, Java, all kinds of assembly languages. Some programs are developed with the aid of other software tools, because developers sometimes don't know how to program in any of the traditional computer languages. For example, when you use the Notes client application to create a database and some forms that perform a business task for you (even if it's just a list of people you have to call tomorrow), that's an application. You created it using Notes, not C, but it's still an application.

Downloaded Applets

Applets, meaning programs downloaded over a network to your local machine and (usually) executed by a *browser* program, are very different from applications. As far as I know, applets can only be written in Java. A browser is an application that knows how to find its way around a network (a LAN or the Internet) and display *pages*. The pages are "programmed" with a language called HTML (HyperText Markup Language) and may contain references to Java applets. When a page contains a reference (an e-mail address, or Universal Resource Locator, or URL) to an embedded Java applet program, the browser fetches that program, and executes it in the context of the page it is rendering for you.

For security reasons, browsers restrict the activities that applets can perform. Applets can't, for example, load and execute programs from your local hard disk, although that is a capability of the Java language. They also can't communicate over the network with any server, except the one from which it was downloaded. If you were to write a Java applet that tried, for example, to erase the disk of the hapless user who downloaded your program, the browser would simply throw a runtime exception, and your applet would stop executing.

Applets are a lot of what makes the stuff you see on the Internet these days hop around and boogie. Most of the headache-producing jump and jive you encounter on most pages, where text flashes in different colors and icons spin around, is programmed with Java and downloaded to your machine by the browser as applets.

We won't discuss applets very much in this book, mainly because you can't use NOI from an applet. The reason is simple: the Java NOI was implemented in a very thin layer of Java code that mostly just packages some arguments and calls into the Notes DLLs (Dynamic Link Libraries) to perform the required work. One of the things that browsers prevent applets from doing is loading a DLL, so applets are incompatible with the Domino 4.6 NOI. That doesn't mean you can't develop a Web site using Domino and have applets in your pages. That works fine, because Domino simply serves up

your applets along with the rest of the stuff making up your site. What doesn't work is downloading an applet to a machine that doesn't have Domino installed on it and having that applet try to use a Notes object. The code just isn't on the machine, and even if it were, the browser wouldn't let your applet load it into memory.

How does a Java program become an applet? That's easy. It just has to be a class that extends (inherits from) the standard class java.applet.Applet. Well, maybe it's not quite that easy. It also has to implement a UI of some kind, or it will be invisible in the HTML page.

Servlets

If an applet is a Java program that gets downloaded to a browser running on a client and is hooked into the HTML page with a special tag, then I guess you'd expect a Java program that runs on the server when it is referenced by an HTML tag to be called a *servlet*. A servlet acts as a sort of plug-in to the server. It's a module that is invoked before an HTML page is served up to a browser and typically inserts HTML into the page on the fly. Servlets can also be explicitly invoked by name in a URL.

What would you do with a servlet? Typically, servlets are used to examine the URL that triggered the retrieval of the current page, do some application-specific processing (a relational database lookup, for example), and insert stuff into the page on the way to the client. As a customization tool for Web pages, servlets allow application designers to have more of the processing occur on the server, offloading the slower client machine. System administrators also like them because servlets allow them to centralize access to system resources, such as back-end databases, and simplify deployment issues.

JavaSoft (the subsidiary of Sun Microsystems that owns Java) has developed a standard servlet API, which you can download from their Web site. In Chapter 11, we'll see how you can use JavaSoft's classes to write a servlet, and then you'll see how to

convert any servlet into a Domino Agent. I'll also describe in detail how to set up your Domino 4.6 HTTP server to work with servlets.

Agents

As with applets and servlets, a Java class becomes an Agent by extending a specified base class, in this case lotus.notes.AgentBase. Although there was an Agent-like facility in Notes Release 3.0 called *macros*, real Agents first appeared in Notes Release 4.0. You could program them using a Notes @function formula (as in Release 3), with a pick list of *simple actions*, or with LotusScript. Notes/Domino 4.6 adds the ability to write agents using Java. In Chapter 8, we'll go into great and gory detail about how to create Java Agents for Domino. For now, though, I'll just mention some of the advantages that Domino Agents give you over servlets and applications:

- Transportability. Agents live in Notes databases and are design elements, just as forms and views are. That means they travel with the database contents when the database is moved, copied, or replicated. Once created, you don't have to keep track of the Java code separately from your application.
- Security. Agents are digitally signed when they are created or modified. This lets you know who touched an Agent last. Also, the Agent, when it runs on a server, will have no greater access to any databases than the signer of the Agent does. Furthermore, users must have designer access to a database to modify Agents. Agents can be *personal* (available for editing and execution only to the person who created it) or *shared* (available for execution to anyone with access to the database, and for editing to anyone with designer access to the database).
- Tracking. Whenever an Agent is run on a server machine, the server logs the start and end execution times, custom output messages, and any errors in the server log.

Like servlets, Agents can be triggered from a URL, a form of network address that tells you where something (an HTML page, an applet, a picture, a video) lives out there in

the universe. Unlike servlets, Agents can also be triggered on Domino servers in a number of other ways, as we'll see shortly. Agents run in the foreground can have a user interface, but Agents run on servers cannot.

CGI (Common Gateway Interface)

CGI programs are sometimes called *scripts*. They are programs written in any programming language supported on the server (Perl and C seem to be the most commonly used languages, although you can use BASIC and even UNIX shell scripts as well), and are triggered either with a URL or an HTML tag. Like background Agents, they exhibit no UI.

We won't be exploring CGI programming in any detail, as there are already numerous books available on that topic. In addition, my personal opinion is that CGI scripting will gradually be replaced by other forms of server programmability, primarily for performance reasons. When a request comes in to an HTTP server to run a CGI script, the server must first locate the file referenced in the URL (or HTML tag) in the file system. Then it must start a new process to execute the CGI program (regardless of whether it's a C executable, a BASIC program or a Perl or shell script). Arguments to the program are passed on the command line. When the program is finished, its process is shut down. The next time a request for that same CGI program comes in, the server again starts a new process for it. There is no way to cache CGI programs or to run them in the server process.

More up-to-date servers support higher throughput (and therefore more scalable) technologies such as servlets and Agents, which usually avoid costly process overhead. In addition, using Java as your programming language is an advantage, because once a Java class has been loaded into the Java Virtual Machine (VM), the Java byte code interpreter and execution environment, it can stay there for awhile, so that successive

invocations of the same Agent or servlet do not cause the class file to be reloaded from disk.

Location and Triggering

Tables 1.1 and 1.2 summarize what's been said about where each kind of program runs and how each is triggered.

Table 1.1 Program Location

Program	Client Browser		Notes Client	Domino Server	Other HTTP Server
Applet	yes	yes	no	no	
Agent	no	yes	yes	no	
Application	no	yes	yes	yes	
Servlet	no	no	yes	yes	
CGI	no	no	yes	yes	

Table 1.2 Program Triggering

Program	Manual	Scheduled	URL	HTML Tag	Other Event
Applet	no	no	yes	yes	no
Agent	yes	yes	yes	no	yes
Application	yes	no	no	no	no
Servlet	no	no	yes	yes	no
CGI	yes	no	yes	yes	no

Manual triggering refers to the ability to cause a program to run synchronously when you want, either from a command-line interface or by clicking on an icon or other command button. In the Notes client, for example, you can cause an Agent to run by selecting its name from the Actions menu or by selecting it in the Agent View and clicking on **Actions/Run** in the menubar. For example, applications and CGI programs can be run from a DOS command prompt.

The Other Event column in Table 1.2 refers to the fact that Domino Agents can be triggered by events such as new mail arriving in a database or by the modification of an existing document.

Both URL and HTML triggering are available anytime you have an HTTP server. Typically, to trigger an Agent or servlet or CGI program from a URL, you simply tack the program name onto the end of the URL (following the server's name, plus any subdirectory). To pass arguments or program-specific commands, you normally use the HTTP query syntax convention, of a question mark followed by more words. The HTTP server sees the program name and will run it if it can be found.

Notes Object Interface

The Notes Object Interface (NOI) is a set of classes used to manipulate the functionality of Notes workstations and Domino servers. As of Domino Release 4.6, NOI is available through LotusScript and Java bindings, as well as through OLE Automation (from Visual Basic, for example). NOI implements classes such as Session, Database and Document, and by using the methods and properties of these classes, you can build extraordinarily sophisticated collaborative and workflow applications.

We'll cover the details of the NOI classes in Chapters 2, 3, 4, 5, and 6, but it's worth summarizing here what kinds of programs can make use of the NOI classes. NOI is available only on machines where Notes/Domino is installed.

Table 1.3 NOI Availability

Program	Java Client	LotusScript Client	Java Server	LotusScript Server
Applet	no	no	no	no
Agent	yes	yes	yes	yes
Application	yes	no	yes	no
Servlet	no	no	yes	no

Java vs. LotusScript vs. JavaScript vs. VBScript

Having dissected the world of Domino programmability into the various kinds of programs you can write, you might wonder why this book focuses so heavily on Java, to the virtual exclusion of all the other Internet programming languages. Well, since you asked, I'll briefly describe the major differences among Java, JavaScript, LotusScript and VBScript, and then I'll tell you why I'm ignoring almost everything except Java.

LotusScript

LotusScript is a language based on Microsoft's Visual Basic (VB). Most of its components actually come from the original BASIC language, now in the public domain. Some VB constructs are supported for compatibility. The three major benefits of LotusScript, however, beyond being familiar to VB programmers, are the object-oriented extensions to the language that provide for classes, methods, and properties; the fact that LotusScript is embedded in the Lotus software products (desktop as well as Domino/Notes); and the LotusScript eXtension (LSX) architecture.

Unlike VB, LotusScript is a fully object-oriented language. You can write your own classes with full inheritance (single), encapsulation, and data hiding abilities (there is no polymorphism, at least not yet). Furthermore, each of the LotusScript host products has its own set of built-in classes that allow you to manipulate the product's objects from LotusScript. Thus, 1-2-3 provides a whole set of Range, Sheet and Workbook object manipulation abilities, and Notes has a set of classes like Session, Database, View, and DateTime.

The LSX architecture is simply a way of implementing LotusScript classes as dynamically loadable libraries (*DLL* in Windows parlance, *shared library* in UNIX, and so on), so that any LotusScript hosting product can use it. Thus, 1-2-3 can load the Notes LSX (provided that Notes is installed on the machine), and the Notes classes appear in the 1-2-3 class browser as if they were native to the spreadsheet world.

In Notes/Domino, LotusScript can be used for both back-end and front-end scripting. The distinction is simply that front-end scripts manipulate the Notes Client user interface, and can only run in the context of the Client UI. Back-end scripts, by contrast, can be run in the background on a server as well as in the Client UI. Back-end classes (the ones we're interested in here) can be used anywhere, while front-end classes (NotesUIWorkspace, NotesUIDatabase, NotesUIView, NotesUIDocument) can be used only in front-end scripts (on a form button or attached to a form event, for example). Agents meant to be triggered in the context of a UI operation, such as from the Actions menu or from a form button, may use front-end classes. However, if you write an Agent using any front-end classes and then try to run that agent in the background (on a scheduled basis, for example) where there is no UI context, the agent will fail to run (the referenced UI classes will not be found). It's the back-end classes that contain all the data manipulation functionality.

LotusScript Agents can be triggered in all the ways we've already discussed: via an URL, from the Notes Client, and in the background by various events (schedule, new mail, etc.).

The big drawback of LotusScript as a programming language is that it isn't supported by any of the popular Web browsers. That means there's no way you can write an applet in LotusScript and have it be downloaded to some random machine and execute there. Still, LotusScript remains a great Notes application programming language.

JavaScript

JavaScript and Java are two completely separate languages. They have nothing in common beyond the word *Java* in their names. Whereas Java is aimed at professional programmers and runs on any machine that has an appropriate VM installed, JavaScript is meant to be used by casual developers, and it works only in the various

browser products supplied by Netscape Corp. While Java programs your computer (via the VM), JavaScript can program only your browser (or whatever browser loads the Web page containing the script). So far, only Netscape's browser product fully supports JavaScript, as there's no published standard for the language. Microsoft's Internet Explorer supports a language that is very similar to JavaScript, called JScript, but you'll find that there are subtle (and important) differences.

Make no mistake, JavaScript fills a real need. People authoring interesting Web pages need a way to "wire" the different elements of the page together and to provide a way for the browser to handle certain UI events (mouse clicks, typing, and so on) properly. JavaScript can be used, for example, to catch a mouse click and route the event to an embedded Java applet.

Is JavaScript a real programming language? Yes and no. It is interpreted, like Java and BASIC, and it has some of the common programming constructs like arrays and for loops. Its drawbacks have to do more with its lack of portability (it runs only with Netscape's browser) and with the fact that when you write JavaScript code, you give away your source code every time your page is downloaded. The JavaScript sources are encoded directly into the HTML stream of the page and are interpreted from source (unlike Java, which is compiled down to a cross-platform interpretable byte stream) directly by the browser. This latter fact drives most professional developers to do as little as possible in JavaScript and to use it to simply connect the dots on the page.

VBScript

VBScript is Microsoft's alternative to JavaScript (though as I mentioned previously, Microsoft also supports a JavaScript-like language called JScript). JavaScript's syntax is not based on Java, but VBScript *is* based on Visual Basic. Apart from syntax differences, it is used in exactly the same way as JavaScript, and it has all the same advantages and disadvantages.

Summary

To sum up, the world of Internet programmability is quite large. Not only do you have to figure out what kind of HTTP server you might want to use but what language(s) you want to program it in, and what kind of functionality you want to expose. Are you looking primarily at downloadable applets? Or are you more interested in server-side functionality, such as workflow and relational DBMS connectivity?

I've attempted to segment the overall space for you and narrow the focus of this book to the following main points: Domino is an incredibly powerful application development platform that incorporates an HTTP server, a couple of cool programming languages, and a great object hierarchy for you to work with. If you've settled on using Domino for your server (or you're thinking about it), then read on to learn more about Java and NOI and about the different ways you can use the tools provided by Lotus to develop your applications.

To this point we've covered the differences among applications, applets, servlets, and Agents. Chapter 2 is an overview of the role played by NOI in Domino programmability and the beginning of our in-depth look at the Notes classes. Later chapters go into much greater detail about how to write Java applications and Agents for Domino (both single and multithreaded). There are also a few chapters on advanced topics, such as how to run existing servlets as Domino Agents, and how to use JDBC with NOI.