Appendix E
Creating LSXs

The Notes Object Interface in Notes release 4.0 was implemented using an architecture originally called LotusScript Extension, or LSX. This architecture is a specification that tells you how to implement a dynamically loaded code library ("DLL" in Windows, "shared library" in UNIX, "NLM" for Netware, and so on for all the supported Notes server platforms), using C++, which implements a set of LotusScript classes. The LSX library can be loaded and used from any of the Lotus products that support LotusScript (Notes, but also 1-2-3, Freelance, WordPro and Approach).

A while after release 4.0 shipped, Lotus released a toolkit to assist developers in creating LSXs. Release 2.0 of the LSX Toolkit now supports an enhancement to the LSX architecture to support= Java. The new version of the toolkit adds a layer of C-callable code that is compatible with JNI, the Java Native Interface spec from JavaSoft. You can now implement your classes so that they can be called from LotusScript or Java (in fact, the LSX architecture also supports method and property invocation from OLE Automation).

With the addition of Java support, the name *LotusScript Extension* is somewhat misleading, but it will probably be with us for a while. Within Lotus there has been support for referring to this technology as *plug-ins* instead, but the new name doesn't seem to have taken off yet.

The LSX Toolkit is available for free on the Lotus Web site (the URL is complex, see the entry in the References.nsf links database on the CD-ROM, or go to http://www.lotus.com and wend your way there yourself). It includes template source code and header files, plus some documentation on how to build your own LSX. Code

developed using the toolkit can be portable (if you follow the directions) across almost all the supported Notes platforms, client and server (NLM has not been certified as yet).

There's also an LSX generation wizard, implemented as a Notes application, of course. The wizard lets you fill out a form describing your classes, methods, and properties, and then it generates a skeleton C++ implementation for you. All you have to do then is fill in the logic for all the calls. If you're writing a Java interface, you also need to write a thin layer of Java code to define your classes and methods. Each Java method calls into one of the predefined JNI entry points generated by the wizard. This is a huge time saver, believe me.

The great thing about the LSX Toolkit and about the LSX architecture in general is that you can wrapper existing classes (or C APIs that you would like to expose in an object-oriented way) pretty easily and create modules that are callable from inside any of the Lotus products or directly from any executable via Java. Yes, you have to program in C++ to take advantage of this feature, but if you you want to hook legacy code into the Domino environment via LotusScript or Java (Agents, buttons, Web server interface, the whole nine yards), it's a huge win.

One new development (not yet released as of this writing, but check the Lotus Web site for updates) is an upcoming version of the toolkit that will support C++ invocation of Notes objects via the LSX interface. Previously, this interface was available only to the LotusScript runtime environment. The next toolkit release will document how to pass a Notes NOI object to one of your own class's methods as an argument, and from C++, invoke any of the publicly available methods or properties on that object.