Appendix A

Extending Tcl

For information on writing C programs and binding them with Tcl, see the third part of the book by John Ousterhout, *An Introduction to Tcl and Tk*. (See "Additional Reading" on page xiv for bibliographic information.)

Installing Header Files

In order to extend Tcl by writing your own C routines, you must install the *sgitcl_dev* option in addition to the *sgitcl.eoe* product. The *sgitcl.dev* product image includes C and C++ header files for program development. The <*tcl.h*> include file is the most important of the many C include files.

C++ Classes and Tcl

The <*tcl++.h*> include file defines a set of C++ classes that can be used to access a Tcl interpreter.

A C++ class may be instantiated in one of two ways. In one method, an interpreter is created and owned by the object. When the object is deleted, the interpreter is deleted. In the other method, the interpreter is passed to the constructor and is referenced by the object, but not owned by it. When the object is deleted, the interpreter is not deleted.

Finding Existing Extensions

Places on the Internet where you can find archives of Tcl extensions and releases, or pointers to information about Tcl, are listed below (as Web sites in URL format):

```
ftp://ftp.aud.alcatel.com/tcl
http://www.sunlabs.com/research/tcl
http://www.sco.com/Technology/tcl/Tcl.html
```

Creating a Shared Library

In order to make C program procedures available as Tcl procedures, you need to build and link them as a dynamic shared object for Tcl to invoke with the **dlopen()** library call. See the dlopen(3) and DSO(5) reference pages for details.

Existing Tcl Extension Packages

Here are the steps to follow for porting an existing Tcl extension package:

1. Build the *.so* file using the **-shared** option to the compiler or loader. Make sure that the build line references all libraries needed by the extension library. For example, Tk requires **-lX11 -lc -lm** as libraries. It is a good idea to add the **-no_unresolved** flag to ensure that there are no unresolvable symbols.

   Many Tcl packages come with a *Makefile* that builds a library archive (*.a*) from which *ld* can produce a shared object. If not, you need to identify what object files need to be included in the build. For extensions that follow normal conventions, this usually means everything except the file containing the **Tcl_AppInit** routine, since your library will be initialized by a Tcl command.

2. Install the *.so* file in */usr/sgitcl/lib*. Inside *sgitcl*, call the **dlopen** procedure:

   ```
   sgitcl>dlopen libname.so init init-routine
   ```

   In this example, *libname* is the library name (for example, *libdb.so*), and *init-routine* is the initialization routine called by **Tcl_AppInit** (for example, **Db_Init**).

The **dlopen** procedure prints out the return value of the initialization routine, which may be an empty string. At this point the library should be loaded and initialized, and all the new extensions should be available.

If **dlopen** returns the "unable to open library" message, make sure you have placed the *.so* file in */usr/sgitcl/lib*. If so, there are probably unresolvable symbols in your library; relink the library specifying **-no_unresolved** to check if there are unresolvable symbols.

If **dlopen** returns the "no such routine *init-routine*" message, it indicates that the initialization routine you specified is not defined in the library. Make sure the routine was included in the object list. Some packages fold this routine into the same file as **Tcl_AppInit**. If this is the case you will need to edit this file to remove **Tcl_AppInit** (comment it out or use #ifdef's). Then add this file to the list of linked files and rebuild the library.

Developing a New Library

After you write code for new procedures as described in Ousterhout's Tcl book, you then need to write an initialization routine that adds the new commands to the interpreter using **Tcl_CreateCommand**. By convention this routine is named ***module-name*_Init** where *module-name* is a short acronym for your extension library, such as Tk, TclX, or Tm. This routine takes a *Tcl_Interp* pointer as its only argument.

Build and install your new shared object as described in the section above. To test your library, run this command:

```
sgitcl>dlopen libname.so init init-routine
```

This should print the string returned by your initialization routine. Your new commands should be added to the interpreter and you can begin testing them.

Other Features of dlopen

As a side effect of **dlopen**, a new command is automatically added to the interpreter—the name of the library. For example, the tclMotif library is opened by the command:

```
sgitcl>dlopen libtclMotif.so
```

This adds a new command **libtclMotif.so**, the sole function of which is to permit the calling of routines defined in the library, as shown in the example below.

```
sgitcl>libtclMotif.so call procedure args...
```

There are two types of routines that may be called: **init** routines and **call** routines.

The **init** routine has the following prototype:

```
int Init_Routine(Tcl_Interp* interp)
```

The *interp* argument is set to the interpreter pointer created by Tcl. To reference an **init** routine, type:

```
sgitcl>library-name init Init-routine
```

The **call** routine has the following prototype:

```
int Call_Routine(Tcl_Interp* interp, int argc, char* argv[])
```

The *interp* argument is the Tcl interpreter; *argc* is a count of arguments and *argv* is an array of argument strings with *argc* elements in the array. To call such a routine, type:

```
sgitcl>library-name call routine-name args...
```

Tcl will parse the *args* in the same way that command arguments are parsed, and pass them to the routine in *argc* and *argv*.

Both routine types should return TCL_OK if successful or TCL_ERROR if something went wrong. Additionally, your procedure should set *interp->result* to a descriptive error string. The return value of an **init** or **call** routine will be the value that your procedure places in *interp->result*, or the empty string if you fail to set a return value.

Note that **dlopen** allows an optional **init** or **call** to follow the library name:

```
sgitcl>dlopen libname.so init init-routine
```