Using imake with Motif 2.0

Paul DuBois
dubois@primate.wisc.edu

Wisconsin Regional Primate Research Center
Revision date: 19 November 1996

Introduction

The primary purpose of this document is to discuss various issues involved in generating Makefiles for the Motif 2.0 distribution and for third-party Motif-based projects. As an incidental purpose, it also describes how to compile and install the Motif software. The document is not a replacement for the Motif release notes; indeed, it refers you to them occasionally. Rather, this document attempts to clarify or amplify certain aspects of those notes.

Background

Open Software Foundation (OSF) distributes Motif 2.0, a software release that is based on X11R5 and is configured using *imake*. The configuration files included in the *config* directory under the root of the Motif source tree are here referred to as "the Motif configuration files." These files consist of a set of X11R5 configuration files that have been modified by OSF and to which OSF has added two Motif-specific files *Motif.tmpl* and *Motif.rules*.

This document consists of notes that I collected while configuring Motif under various conditions:

- **Configuring Motif using the instructions in the Motif release notes**

  This is the control condition, using the configuration files that come with the Motif distribution in the *config* directory under the source tree root and using the *Makefile*-generation procedure described in the release notes.

- **Configuring Motif using *imboot* to direct *Makefile* generation**

  This condition uses the configuration files distributed with Motif, but generates Makefiles using *imboot*, a general-purpose *Makefile* bootstrapper. I wanted a process that (i) didn't require running *imake* commands manually; (ii) didn't need *xmkmf*, which I consider "dedicated" to X11; and (iii) would work for configuring Motif itself as well as for third-party Motif-based projects.

- **Configuring Motif using X11 configuration files other than those distributed with Motif**

  In this condition, I tested configuring Motif with sets of configuration files from the X11R5 and X11R6.1 distributions, to which *Motif.tmpl* and *Motif.rules* had been added, as opposed to using the configuration files that come with the Motif distribution. There are different reasons for doing something like this. You may want to use a set of X11R5 files other than those which OSF provides with Motif (e.g., if you've made local modifications to them that aren't present in the OSF-distributed files). Or you may want to use X11R6.1 files, so as to configure Motif for use with a more recent X11 release than R5. (When I refer to R6.1 here, you can substitute R6 without loss of generality. For the purposes of this document, there are no pertinent differences between releases that I know of.)

Getting Started

This section describes some of the issues you should consider as you're getting ready to build Motif:

- Several modifications to files in the Motif distribution are described in this document. It's prudent to make a backup copy of any file you modify, in case you make a mistake or want to undo your actions. This also allows you to *diff* the original file with your modified copy later to review what you did.

- Motif uses `XTop` and `MTop` to indicate the absolute pathnames to the tops of the X11 and Motif source trees. Define them in *site.def* like this:

  ```
  #define XTop /path/to/X11/source/root
  #define MTop /path/to/Motif/source/root
  ```

  You can build Motif without setting `XTop` if the X11 header files and libraries are installed in system directories.

- Motif uses the `UseInstalled` macro, like X11, but also uses an additional macro `Use-InstalledMotif`. The implications of defining `UseInstalled` and `UseInstalledMotif` versus leaving them undefined are discussed in "Finding Header Files, Libraries, and Configuration Files."

- Decide whether or not you want to compile the demonstration programs. It takes a lot of space to do so and the build process takes longer. I found it useful to speed up build testing by restricting the build to the main part of the Motif distribution. To exclude the demo subtree from the build, turn on `Dont-BuildMotifDemos` in *site.def*:

  ```
  #define DontBuildMotifDemos /* as nothing */
  ```

  I recommend that you define this symbol when you configure Motif initially. You can always remove the definition from *site.def* later and redo the build should you wish to include the demonstration programs.

- The X Pixmap Library (XPM) is used by some of the sources, but is not present in the Motif or X11 distributions. You can get it as part of the X11 *contrib* distribution. If you don't have the XPM library and don't want to get it, define `UseLocalXpm` in *site.def*:

  ```
  #define UseLocalXpm /* as nothing */
  ```

  This causes `-DUSE_LOCAL_XPM` to be passed to compiler operations so that some sort of Motif built-in equivalent will be used.

  A problem with the `UseLocalXpm` mechanism is that it isn't respected everywhere in the distribution. For instance, source in *demos/program/piano* includes *xpm.h* whether or not `UseLocalXpm` is defined. To work around this, you can exclude the demo programs from the Motif build by defining `Dont-BuildMotifDemos` as described above.

  If you leave `UseLocalXpm` undefined, the build process must be able to find *xpm.h* and *libXpm.a*. One way to specify these locations is by defining `XpmIncludes` as an `-I` argument for the directory in which *xpm.h* is located, and `XpmLibs` as the full pathname to the *libXpm.a* library. Here's an example of what you might put in *site.def*:

  ```
  #define XpmIncludes -I/usr/include/X11
  #define XpmLibs /usr/lib/libXpm.a
  ```

- The `_NO_PROTO` macro is supposed to be defined in *lib/Xm/Xm.h* if your system doesn't support function prototypes. However, some source files (e.g., *XmIm.c* and *XmString.c* in the *lib/Xm* directory) also use `_NO_PROTO` to determine whether to include *stdarg.h* or *varargs.h*. If you don't have *stdarg.h*, you should turn on `_NO_PROTO` if *Xm.h* doesn't define it for you. You can do that by defining

`MotifDefines` in your platform file:

```
#define MotifDefines -D_NO_PROTO
```

If your platform file already has a definition of `MotifDefines`, just add `-D_NO_PROTO` to the end of the value that is already there.

- Remove `WorldP.h` from the value of `INCLUDES` in *lib/Xm/Imakefile* or you may find that *make install* blows up in *lib/Xm*. I'm not sure what *WorldP.h* is for anyway; it's not present in the distribution anywhere, it's not included by any other file, and removing it from the *Imakefile* doesn't cause any problem. (Leaving it listed in `INCLUDES` causes the build process to create *World.P* as a symlink to itself (!), which is why the attempt to install it fails.)
- Be sure to read Chapter 6 of the Motif release notes (especially section 6.3.9) for information about other system dependencies you may need to be aware of.

Finding Header Files, Libraries, and Configuration Files

When you generate Motif Makefiles, location parameters are written into those Makefiles to instruct the build where to look for X11 and Motif header files and libraries. The locations are determined by the macros `UseInstalled` and `UseInstalledMotif`. These two macros also determine where *imake* looks for configuration files when you generate Makefiles.

Finding Header Files and Libraries

Motif is based on X11R5 and the normal build process uses X11R5 header files and libraries. You can direct the Motif build process to search for those files either in the X11 source tree or installed in public directories. If the `UseInstalled` macro is undefined when you generate the Motif Makefiles, the build looks in the X11 source tree. If `UseInstalled` is defined, the build looks for the files in public directories.

If you want to look for X11 files in the X11 source tree (i.e., if you leave `UseInstalled` undefined when you generate the Makefiles), the X11 header files must be present and the libraries must already be built there. Otherwise the Motif build will fail. You don't need to build the entire X11 distribution, but you must at least generate the X11 Makefiles and then run the following commands, starting in the root of the X11 distribution:

```
% make includes                    Create header file links
% cd lib ; make                    Build standard libraries
% cd ../extensions/lib ; make      Build extension libraries
```

Also, you must not subsequently run *make clean* in the X11 source tree. If you do, you'll remove the requisite header files and libraries, causing any later Motif build to fail.

The Motif build also requires (not surprisingly) Motif header files and libraries. You can direct the Motif build process to search for those files either in the Motif source tree or installed in public directories. If the `UseInstalledMotif` macro is undefined when you generate the Motif Makefiles, the build looks in the Motif source tree. If `UseInstalledMotif` is defined, the build looks for the files in public directories.

Normally you use the Motif header files and libraries located in the Motif source tree when you build Motif itself. On the other hand, it's more convenient if those files are installed when you build other Motif-based projects located outside the Motif distribution. This means that `UseInstalledMotif` is typically undefined for building Motif itself and defined for building other Motif-based projects.

Finding Configuration Files

The `UseInstalled` and `UseInstalledMotif` macros also have an effect on *Makefile* generation in that they determine where *imake* looks for configuration files. The way this works is similar to but not quite the same as in X11. In X11, Makefiles are generated using configuration files located in the X11 source tree if `UseInstalled` is undefined, and using publicly installed configuration files if `UseInstalled` is defined. In Motif, Makefiles are generated using configuration files located in the Motif source tree unless `UseInstalled` and `UseInstalledMotif` are both defined, and using publicly installed files otherwise.. To see why this is, take a look at how `IMAKE_CMD` is initially defined in *Imake.tmpl* and subsequently redefined in *Motif.tmpl*.

Building Motif 2.0 According to the Motif Release Notes

I find it helpful to break down the Motif build process into three stages:

- Generating the Makefiles
- Compiling the software
- Installing the software

In this section, I'll describe how to perform these three stages of the build according to the instructions in the release notes. In later sections I'll describe alternate methods of generating the Makefiles. Compiling and installing the software is pretty much the same no matter how you generate the Makefiles, so even if you use an alternate method of generating them, you can still use the compilation and installation instructions in this section.

Note that since the standard build process for Motif uses X11R5-based configuration files, you may have difficulties using the instructions in this section for any system not supported by R5. For example, it's difficult to configure Motif under Solaris 2.x. In such cases, you'll probably need to use the instructions later on that describe how to use R6.1-based configuration files.

Generating the Makefiles

The instructions in the release notes assume that you'll use the Motif header files and libraries from within the Motif source tree, i.e., that `UseInstalledMotif` is undefined. On the other hand, they describe how to generate Makefiles for using X11 files either in the X11 source tree or installed. In each case, because the configuration files are R5-based, the source tree or installed files are also assumed to be from the X11R5 distribution.

To generate the initial top-level *Makefile* for a build that will use X11 header files and libraries located in the X11 source tree, the release notes specify the following command, where `XTop` is the pathname to the top of your X11 source tree:

```
% make Makefile "BOOTSTRAPCFLAGS=-IXTop/X11"
```

`XTop` should be an absolute pathname, not relative. If that command doesn't work, you might have better luck with this one:

```
% make -f Makefile.ini Makefile "BOOTSTRAPCFLAGS=-IXTop/X11"
```

After generating the initial *Makefile*, generate the Makefiles for the rest of the distribution:

```
% make Makefiles
```

Then you can proceed to "Compiling the Software" below.

If you want to use X11 header files and libraries that are installed publicly, the commands are slightly different. To build the initial top-level *Makefile*, the release notes specify a command similar to the one below, where `/usr/include/X11` is the path to your installed X11 header files (if the files are installed in a different directory on your system, make the appropriate substitution):

```
% make Makefile "BOOTSTRAPCFLAGS=-I/usr/include/X11" \
IMAKE_DEFINES=-DUseInstalled
```

If that command doesn't work, you might have better luck with this one:

```
% make -f Makefile.ini Makefile "BOOTSTRAPCFLAGS=-I/usr/include/X11" \
IMAKE_DEFINES=-DUseInstalled
```

On some systems you may need to include other information in the value of `BOOTSTRAPCFLAGS`, depending on how you normally build *imake*.

After generating the initial *Makefile*, generate the Makefiles for the rest of the distribution:

```
% make Makefiles
```

Then you can proceed to "Compiling the Software" below.

For using installed X11 files, the release notes specify that after you generate the initial *Makefile*, you should perform the procedure shown below. However, the procedure is unnecessary; it consists of a sequence of commands that involves becoming *root*, installing the Motif configuration files, then generating the remaining Makefiles in the Motif distribution:

```
% cd config
% ./imake -DUseInstalled -I.
% su
# make install
# exit
% cd ..
% make Makefiles
```

This procedure is unnecessary because a simple *make Makefiles* suffices. The procedure is perhaps even undesirable as well, since installing the Motif configuration files wipes out your X11R5 configuration files (see "Installing the Software" below).

I believe the procedure has probably been unnecessary since Motif 1.2.1. Prior to that release, `IMAKE_CMD` (the command used to build a *Makefile*) was defined to look for installed configuration files if `UseInstalled` was defined. That required that if you defined `UseInstalled` when you built the top-level *Makefile*, you had to install the configuration files before using that *Makefile* to run *make Makefiles*. However, beginning with release 1.2.1, `IMAKE_CMD` was redefined to look for installed configuration files only if both `UseInstalled` and `UseInstalledMotif` were defined. In that case, when you build the top-level *Makefile* with `UseInstalled` defined (but not `UseInstalledMotif`), *make Makefiles* still looks for configuration files in the Motif source tree and it's not necessary to install them. I suspect the release notes were simply never updated after 1.2 to take account of this change in behavior.

Compiling the Software

Once the Makefiles are generated, compiling the Motif software amounts to running the following commands in the Motif source root directory:

```
% make clean
% make includes
% make
```

The *make clean* isn't always strictly necessary, but if you're messing around experimenting with configuration parameters and regenerating Makefiles, it's a good idea to clean out the distribution tree for new compiles.

makedepend problems. The release notes indicate that you can run *make depend* after *make includes*, but it's common for *make depend* to die in the *lib/Xm* directory with an "out of space: increase MAXFILES" error. That happens because there are so many include files involved that *makedepend* runs out of table space. (This is one reason I don't recommend trying a *make World* operation. That involves a *make depend* step, so a *World* build typically will fail.)

If you really want tto generate dependencies, you have a couple of options. The first is to run *make depend* in individual directories other than *lib/Xm*. The second is to recompile the *makedepend* program with a larger value of `MAXFILES`. To do this, go to the source directory for *makedepend* and change the value of `MAXFILES` in *def.h* from 512 to a larger number. 1024 seems to be large enough, although you could make it larger to be safe. Be sure to modify *def.h* in the right source directory. If *make depend* first builds *makedepend* from source, it's probably doing so in your X11 source tree, so that's where you need to make the change.

I recommend the first option if your version of *makedepend* is from R5. Otherwise, even if you recompile it with a larger `MAXFILES`, it may take a very long time to run in the *lib/Xm* directory. I've observed times of nearly an hour on HP 715/64 and SPARC 2 systems. Even on a Sun Ultra, the R5 *makedepend* takes 10 minutes. The R6.1 *makedepend* is much more efficient, taking less than a minute on any of these machines.

mkdirhier problems. If the build fails because *mkdirhier* can't be found, one way around the problem is to copy the *mkdirhier.sh* script from your X11 distribution into a public directory that's in your search path, e.g., */usr/bin/X11*. (*mkdirhier.sh* is in *util/scripts* in the R5 distribution.) Copy the script, naming it *mkdirhier*, and make it executable. Then define the `MkDirHier` macro in *site.def* as shown below and rebuild your Makefiles:

```
    #define MkDirHier mkdirhier /* assume publicly installed */
```

Installing the Software

After Motif has been built, you can install it. However, you may not want to do this with a plain *make install* command. *make install* in the Motif source root does the following things, some of which are destructive:

- Installs Motif clients such as *mwm* and *xmbind*, and the UIL compiler
- Installs Motif header files and libraries
- Installs *imake* in */usr/bin/X11* (wiping out any X11R5 *imake* you may already have installed)
- Installs the Motif configuration files in */usr/lib/X11/config* (wiping out any X11R5 configuration files you may already have installed)

Installation of the programs, header files, and libraries does what you want: it puts them in locations where they'll be available publically. Installation of *imake* and the configuration files is more problematic. It installs those files so they're publicly available, but it also wipes out other files in the process.

I suppose it can be argued that it makes sense to put the Motif configuration files in the same place as your X11R5 files because the Motif files are based on the X11R5 files and are therefore quite similar to them. I suppose too that this approach is intended to allow *xmkmf* to be used for bootstrapping Motif-based projects.

However, I wasn't particularly interested in clobbering my X11R5 configuration files. What if I've already made site-specific modifications to my R5 files that I don't want to lose? Flopping the Motif files on top of them wipes out my changes. It's not clear that installing the Motif configuration files in */usr/lib/X11/config* allows *xmkmf* to use them anyway. If you've upgraded your X11 installation from R5 to R6 or to R6.1, *xmkmf* will look for configuration files in */usr/X11R6/lib/X11/config* or */usr/X11R6.1/lib/X11/config*, not in */usr/lib/X11/config*.

For these reasons, I wanted to be able to perform an installation that didn't include *imake* and the configuration files. If you're in the same boat, there are two ways you can install everything other than the configuration-related stuff. One way is to install from all the directories but *config*. You can do that in the Motif source root by overriding the value of `SUBDIRS` on the *make install* command line as follows:

```
% make install SUBDIRS="tools/makemsg lib tools/wml clients bitmaps bindings"
```

You may need to be *root* to run *make install*.

Another option, if you want to make sure you never install on top of the X11 configuration files and *imake*, is to remove *config* from the value of `SUBDIRS` in the top-level *Imakefile* and rebuild the *Makefile*. Then *make install* won't descend into the *config* directory at all.

If you're interested in installing only the Motif libraries and header files (e.g., so that you can build other Motif-based projects), just do this:

```
% cd lib
% make install
```

In the next section, we'll discuss a method of installing the configuration files for public use that leaves any already-installed X11R5 files intact. If installation fails because *mkdirhier* can't be found, see "mkdirhier problems" earlier in this section.

Alternative Methods of Configuring Motif

This section describes ways of generating Makefiles for the Motif distribution that differ from those described in the release notes. I'll cover three cases:

- Configuring Motif with its own configuration files
- Configuring Motif with an alternate set of X11R5 configuration files
- Configuring Motif with an alternate set of X11R6.1 configuration files

These cases use distinct sets of configuration files. To make it easy to switch between sets at will, I'm going to use *imboot*, a general-purpose *Makefile* bootstrapper. (If you need *imboot*, see "Obtaining Software" at the end of this document.) *imboot* finds configuration files by looking in directories under a configuration root directory, which I'll assume here is */usr/local/lib/config*. (Make the appropriate substitution in the examples that follow if *imboot* uses a different directory on your system.) I'll use directories under the configuration root named *Motif-2.0*, *Motif-2.0R5*, and *Motif-2.0R6.1* to install configuration files for the cases to be covered. These names aren't especially euphonious, but they should be reasonably unambiguous.

There are some implications of using *imboot* and configuration files that are installed under the *imboot* configuration root:

- It's not necessary to overwrite any already-installed X11R5 configuration files you may have.
- When using publicly installed configuration files, *imboot* defines `UseInstalled`. This means the build process will assume that tools such as *imake* and *makedepend* are installed and won't try to compile them from source. The build will also look for X11 header files and libraries in public directories rather than in the X11 source tree. Obviously, this imposes the additional requirement that these tools, header files, and libraries be installed already. However, there are certain advantages. You almost certainly have *imake* and *makedepend* installed anyway (right?), so there's little point in building them again from source. Also, when you don't use header files and libraries that are located in the X11 source tree, you can do a *make clean* in that tree without causing subsequent builds of Motif to fail.

- Installing the configuration files makes it easier to use them for Motif-based projects other than Motif itself.

Here are the steps involved in using an alternate set of configuration files and *imboot* to configure Motif:

- Change the top-level Motif *Imakefile* by removing *config* from the value of `SUBDIRS`. This prevents the build from trying to compile *imake* from source in that directory.
- Copy the configuration files to a directory under the *imboot* configuration root. For the Motif configuration files, this means copying the files in the *config* directory of the Motif distribution. For R5 or R6.1 files, it means making a copy off the regular X11 configuration files, then adding to them the *Motif.tmpl* and *Motif.rules* files from the Motif distribution.
- Add Motif release-number definitions to the `BeforeVendorCF` section of *site.def* if they aren't there already.
- Define `ConfigDir` to specify the configuration file self-reference (the directory in which the files expect themselves to be located).
- Remove the `IMAKE_CMD` definition from *Motif.tmpl* so that *imake* doesn't look in the Motif source tree for configuration files.
- Make any other modifications that may be necessary.

The particulars of these steps are described below for each set of configuration files. When the instructions describe modifications to be made to configuration files or to Imakefiles, you can either make the changes yourself, or use the modified Imakefiles from the *motif-support* distribution in the *imake-stuff* archive. See the section "Obtaining Software" to get this distribution.

When you use a set of files other than the one that comes with Motif, an issue to which you should pay special attention is that OSF added some stuff to their configuration files that might not be in your X11 files. The additions tend to be mainly in the platform files and in *site.def*, so you should look in your set of configuration files and compare the appropriate platform file and *site.def* with the correponding files in the Motif distribution. For example, there may be a definition of `MotifDefines` that you'll need to add to your platform file. As a second example, the Motif version of *hp.cf* contains the following definition of `YaccFlags`:

```
#define YaccFlags -Nm15000
```

Without this line, *yacc* blows up on HP machines with a "table space too small" error in the *tools/wml* directory while building some UIL stuff. The X11 version of *hp.cf* doesn't have that definition, so you must add it if you use your own X11 files.

If you're using Solaris, your *imake* may be set up to use the OpenWindows configuration files located in */usr/openwin/lib/config* even if you tell it to use other files. For a discussion of how to fix this problem (and others), check Appendix J of the boa book (2nd. edition), or look for the "*imake* vs. Solaris" paper in the *imake-stuff* archive listed in the section "Obtaining Software."

Configuring Motif Using Motif Configuration Files

To install the Motif configuration files where *imboot* can get at them, copy them to a directory under the *imboot* configuration root. You can do that from the Motif source root directory like this:

```
% mkdirhier /usr/local/lib/config/Motif-2.0
% cp config/* /usr/local/lib/config/Motif-2.0
```

Then change directory to */usr/local/lib/config/Motif-2.0*.

Override the value of the self-reference by defining it in *site.def* like this:

```
#ifndef ConfigDir
#define ConfigDir /usr/local/lib/config/Motif-2.0
#endif
```

Locate the following construct in *Motif.tmpl*:

```
#if defined(UseInstalled) && !defined(UseInstalledMotif)
     IRULESRC = $(CONFIGSRC)
    IMAKE_CMD = $(IMAKE) -DUseInstalled -I$(NEWTOP)$(IRULESRC) $(IMAKE_DEFINES)
#endif
```

These lines redefine `IMAKE_CMD` so that *imake* sometimes looks in the Motif source tree for configuration files. This isn't what you want for using files that are installed, so either remove these lines or comment them out.

Run through the items listed in the "Getting Started" section earlier in this document to make sure you've addressed the issues listed there.

After making the changes above, change directory to the Motif source tree root. Remove `config` from the value of the `SUBDIRS` variable in the top-level *Imakefile*. Then you should be able to build the Makefiles like this:

```
% imboot -c Motif-2.0
% make Makefiles
```

Generating Makefiles Using Alternate X11R5 Configuration Files

To use a set of X11R5 files as a base and add the Motif-specific files to them, create a directory */usr/local/lib/config/Motif-2.0R5*, copy your R5 files into that directory, and add to them the *Motif.tmpl* and *Motif.rules* files from the Motif distribution. Then change directory to */usr/local/lib/config/Motif-2.0R5*.

Define the Motif release level in the `BeforeVendorCF` section of *site.def*:

```
#define MotifMajorVersion 2
#define MotifMinorVersion 0
#define MotifUpdateLevel 0
```

Override the value of the self-reference by defining it in the `AfterVendorCF` section of *site.def* like this:

```
#ifndef ConfigDir
#define ConfigDir /usr/local/lib/config/Motif-2.0R5
#endif
```

You also need to integrate the Motif files into the X11 architecture. *Imake.tmpl* will have a section that looks like this:

```
#include <Project.tmpl>

#include <Imake.rules>
```

Modify that section so it looks like this:

```
#include <Project.tmpl>
#include <Motif.tmpl>

#include <Imake.rules>
#include <Motif.rules>
```

Locate the following construct in *Motif.tmpl*:

```
#if defined(UseInstalled) && !defined(UseInstalledMotif)
     IRULESRC = $(CONFIGSRC)
```

```
        IMAKE_CMD = $(IMAKE) -DUseInstalled -I$(NEWTOP)$(IRULESRC) $(IMAKE_DEFINES)
    #endif
```

These lines redefine `IMAKE_CMD` so that *imake* sometimes looks in the Motif source tree for configuration files. This isn't what you want for using files that are installed, so either remove these lines or comment them out.

Run through the items listed in the "Getting Started" section earlier in this document to make sure you've addressed the issues listed there.

After making the changes above, change directory to the Motif source tree root. Remove `config` from the value of the `SUBDIRS` variable in the top-level *Imakefile*. Then you should be able to build the Makefiles like this:

```
    % imboot -c Motif-2.0R5
    % make Makefiles
```

Generating Makefiles Using X11R6.1 Configuration Files

First, remove `config` from the value of the `SUBDIRS` variable in the top-level *Imakefile.*

Motif is not officially supported for use with X11R6.1, but it's possible to build it using a set of R6.1 configuration files. If you want to do this, you must have an R6 or later version of *imake* installed, due to changes in `XCOMM` handling between R5 and R6. Otherwise you'll end up with Makefiles that are full of `XCOMM` lines and *make* will complain when you try to use them. (If your *imake* is not recent enough, see "Obtaining Software" at the end of this document.)

To use a set of X11R6.1 files as a base and add the Motif-specific files to them, create a directory */usr/local/lib/config/Motif-2.0R6.1*, copy your R6.1 files into that directory, and add to them the *Motif.tmpl* and *Motif.rules* files from the Motif distribution. Then change directory to */usr/local/lib/config/Motif-2.0R6.1*.

Define the Motif release level in the `BeforeVendorCF` section of *site.def*:

```
    #define MotifMajorVersion 2
    #define MotifMinorVersion 0
    #define MotifUpdateLevel 0
```

Override the value of the self-reference by defining it in the `AfterVendorCF` section of *site.def* like this:

```
    #ifndef ConfigDir
    #define ConfigDir /usr/local/lib/config/Motif-2.0R6.1
    #endif
```

You also need to integrate the Motif files into the X11 architecture. *Imake.tmpl* has a section that looks like this:

```
    #ifndef LocalRulesFile
    #define LocalRulesFile <noop.rules>
    #endif
    #include LocalRulesFile
    #include <Project.tmpl>
    #ifndef LocalTmplFile
    #define LocalTmplFile <noop.rules>
    #endif
    #include LocalTmplFile
```

Define `LocalTmplFile` and `LocalRulesFile` as the names of the Motif-specific files by adding the following to *site.def*:

```
    #define LocalTmplFile <Motif.tmpl>
```

```
#define LocalRulesFile <Motif.rules>
```

Locate the following construct in *Motif.tmpl*:

```
#if defined(UseInstalled) && !defined(UseInstalledMotif)
     IRULESRC = $(CONFIGSRC)
    IMAKE_CMD = $(IMAKE) -DUseInstalled -I$(NEWTOP)$(IRULESRC) $(IMAKE_DEFINES)
#endif
```

These lines redefine `IMAKE_CMD` so that *imake* sometimes looks in the Motif source tree for configuration files. This isn't what you want for using files that are installed, so either remove these lines or comment them out.

Motif.rules refers to a rule called `SaberProgramTarget()`. The rule is defined in the R5 *Imake.rules* file. In R6.1, the rule is still present, but it's now called `CenterProgramTarget()` because the Saber C product is now known as CodeCenter. You could go through *Motif.rules* and change each instance of `SaberProgramTarget()` to `CenterProgramTarget()`, but it's easier to put the following line at the top of the file:

```
#define SaberProgramTarget CenterProgramTarget
```

Run through the items listed in the "Getting Started" section earlier in this document to make sure you've addressed the issues listed there. If you define `XTop`, it should point to the top of your R6.1 source tree, not to the top of your R5 source tree.

Now you've done the easy part. The hard part is getting the configuration machinery for libraries to work correctly, because there are significant differences between R5 and R6 in the way you build libraries. (For more information, see the boa book, 2nd. edition, pp. 106-110.) These differences necessitate some fairly extensive changes to the way Motif libraries are built:

- Generator macros `SharedLibReferences()` and `UnsharedLibReferences()` are defined in *Motif.tmpl* and used to generate Motif library variable names. However, these versions differ from and are incompatible with the versions defined in the R6.1 configuration files. The incompatibility must be dealt with.

- The template file *Library.tmpl* does more in R6.1 than it does in R5. This affects the way you write library-building Imakefiles.

These differences make it necessary to modify *Motif.tmpl* in the configuration files, as well as all library-building Imakefiles in the Motif source distribution. I'll give an outline of what's involved here. To see the full extent of the changes, get the *motif-support* distribution (see the section "Obtaining Software").

Changes to Motif.tmpl. The Motif macros `SharedLibReferences()` and `UnsharedLib-References()` generate variable assignments using their first argument to determine the variable name. For instance, the macros are invoked this way for the Mrm library:

```
#if SharedLibMrm
SharedLibReferences(MRESOURCELIB,Mrm,$(MRESOURCESRC),$(SOMRMREV))
#else
UnsharedLibReferences(MRESOURCELIB,Mrm,$(MRESOURCESRC))
#endif
```

The macros use `MRESOURCELIB` to generate variable assignments for `MRESOURCELIB` and `DEP-MRESOURCELIB` in Makefiles:

```
   MRESOURCELIB = $(DEPMRESOURCELIB)
DEPMRESOURCELIB = $(MRESOURCESRC)/libMrm.a
```

The fourth argument to `SharedLibReferences()` is the shared library revision number, typically specified by referring to the variable whose value is that number. In the example above, `$(SOMRMREV)` specifies the revision number. `SOMRMREV` must be assigned elsewhere; in *Motif.tmpl*, its value is assigned like this:

```
SOMRMREV = SharedMrmRev
```

The X11R6.1 versions of `SharedLibReferences()` and `UnsharedLibReferences()` are similar, but use the first argument as the "base" name for the variables to be generated. They assume that "LIB" will not be passed as part of the argument and they automatically add it to the end of the variable names they generate. The fourth argument to `SharedLibReferences()` is the name (not the value) of the library revision-number variable. The fifth argument is the revision number, typically specified by referring to the macro whose value is that number. Thus, the generator macros are invoked like this:

```
#if SharedLibMrm
SharedLibReferences(MRESOURCE,Mrm,$(MRESOURCESRC),SOMRMREV,SharedMrmRev)
#else
UnsharedLibReferences(MRESOURCE,Mrm,$(MRESOURCESRC))
#endif
```

Note the differences:

- The first argument does not have "LIB" at the end.
- The fourth argument to `SharedLibReferences()` is `SOMRMREV` rather than `$(SOMRMREV)`.
- `SharedLibReferences()` has a fifth argument, `SharedMrmRev`.

The macros generate assignments for the library variables by concatenating `LIB` to the end of the first argument. In addition, they construct an assignment for the revision-number variable from the fourth and fifth arguments:

```
       SOMRMREV = SharedMrmRev
   MRESOURCELIB = $(DEPMRESOURCELIB)
DEPMRESOURCELIB = $(MRESOURCESRC)/libMrm.a
```

The preceding example illustrates the general method for changing *Motif.tmpl* generator macro invocations to conform to the X11R6.1 conventions: remove `LIB` from end of the first argument of `SharedLibReferences()` and `UnsharedLibReferences()`, and modify the final argument of `SharedLibReferences()`.

This method works for all but the Xm library, which for some reason is associated with the variable name `XMLIBONLY`, i.e., a name with "LIB" is in the middle of the name and not at the end. Because of this, you can't convert the macro invocations by removing "LIB" from the end of the first arguments. To handle this case, just leave the first arguments alone and modify the final arguments of `SharedLibReferences()` in the usual way. For the Xm library, the relevant section of *Motif.tmpl* looks like this:

```
#if SharedLibXm
SharedLibReferences(XMLIBONLY,Xm,$(MWIDGETSRC),$(SOXMREV))
#else
UnsharedLibReferences(XMLIBONLY,Xm,$(MWIDGETSRC))
#endif
```

Change it to this:

```
#if SharedLibXm
SharedLibReferences(XMLIBONLY,Xm,$(MWIDGETSRC),SOXMREV,SharedXmRev)
#else
UnsharedLibReferences(XMLIBONLY,Xm,$(MWIDGETSRC))
#endif
```

This will generate variable names of `XMLIBONLYLIB` and `DEPXMLIBONLYLIB` (not `XMLIBONLY` and `DEPXMLIBONLY` as we want). To work around the problem, add lines that "alias" the correct variable names to the incorrect names:

```
   XMLIBONLY = $(XMLIBONLYLIB)
DEPXMLIBONLY = $(DEPXMLIBONLYLIB)
```

The Motif definitions of `SharedLibReferences()` and `UnsharedLibReferences()` in *Motif.tmpl* are no longer needed; remove them.

Finally, since the revision-number variables will have their values assigned to them by `SharedLib-References()` now, you can remove the following section from *Motif.tmpl*:

```
#if HasSharedLibraries
          SOXMREV = SharedXmRev
         SOMRMREV = SharedMrmRev
         SOUILREV = SharedUilRev
     SOACOMMONREV = SharedACommonRev
      SOSCRIPTREV = SharedScriptRev
        SOUTILREV = SharedUtilRev
      SOCREATEREV = SharedCreateRev
      SOVISUALREV = SharedVisualRev
       SOSYNTHREV = SharedSynthRev
     SOMCOMMONREV = SharedMCommonRev
#endif
```

Making the changes just described gets you to the point where you can generate the Makefiles. Change directory to the root of the Motif source tree. Remove `config` from the value of the `SUBDIRS` variable in the top-level *Imakefile*. Then you should be able to build the Makefiles like this:

```
% imboot -c Motif-2.0R6.1
% make Makefiles
```

Unfortunately, the changes made so far aren't sufficient to generate *correct* Makefiles in the directories where libraries are built.

Changes to Imakefiles. To make the Motif libraries build correctly, you must change the Imakefiles in the subdirectories of the *lib* directory. In general, the changes to make are:

- Include the *Library.tmpl* file at a different point for R6.1 than for R5.
- For R6.1, don't invoke most of library construction and installation rules; *Library.tmpl* invokes most of them for you. (There are a few exceptions, such as rules that compile object files with special flags.)

By testing the `ProjectX` macro, it's possible to write the Imakefiles so they'll work whether you're using R5-based or R6.1-based configuration files. `ProjectX` has a value of 5 or 6, which allows release-dependent decisions to be made in Imakefiles.

Modified Imakefiles can be found in the *motif-support* distribution. I'll describe the changes for the Xm library below.

To convert the Xm *Imakefile*, change directory into *lib/Xm* and find this line in the *Imakefile*:

```
#include <Library.tmpl>
```

Library.tmpl should be included at this point only for R5-based files, so change the line to this:

```
#if ProjectX < 6
#include <Library.tmpl>
#endif
```

Then find the section that looks like this:

```
LibraryObjectRule()

SpecialLibObjectRule(Xmos.o,$(ICONFIGFILES),$(SRCH_DEFINES))
SpecialLibObjectRule(VirtKeys.o,$(ICONFIGFILES),$(BINDINGS_DEF))
SpecialLibObjectRule(ImageCache.o,$(ICONFIGFILES), $(XPM_INCLUDES) $(XPM_DEFINES) )

#if DoSharedLib && SharedDataSeparation
SpecialObjectRule(sharedlib.o,,$(SHLIBDEF))
#endif

#if DoSharedLib
#if DoNormalLib
SharedLibraryTarget(Xm,$(SOXMREV),$(OBJS),shared,..)
#else
SharedLibraryTarget(Xm,$(SOXMREV),$(OBJS),.,.)
#endif
InstallSharedLibrary(Xm,$(SOXMREV),$(USRLIBDIR))
#if SharedDataSeparation
SharedLibraryDataTarget(Xm,$(SOXMREV),$(UNSHAREDOBJS))
InstallSharedLibraryData(Xm,$(SOXMREV),$(USRLIBDIR))
#endif
#endif
#if DoNormalLib
NormalLibraryTarget(Xm,$(OBJS))
InstallLibrary(Xm,$(USRLIBDIR))
#endif
#if DoProfileLib
ProfiledLibraryTarget(Xm,$(OBJS))
InstallLibrary(Xm_p,$(USRLIBDIR))
#endif
#if DoDebugLib
DebuggedLibraryTarget(Xm,$(OBJS))
InstallLibrary(Xm_d,$(USRLIBDIR))
#endif

LintLibraryTarget(Xm,$(SRCS))
InstallLintLibrary(Xm,$(LINTLIBDIR))

BuildIncludes($(HEADERS),Xm,..)

InstallMultiple($(HEADERS),$(USRINCDIR)/Xm)

DependTarget()

NormalLintTarget($(SRCS))
```

All of this must be retained for R5-based configuration files, but most of these rules are invoked automatically for R6.1-based files. What we want to do here is convert the original section shown above to a construct that looks like this:

```
#if ProjectX < 6
...original section (except DependTarget())...
#else
...modified section for R6.1...
#endif
DependTarget()
```

The modified section that goes in the else-clause looks like this:

```
#define LibName Xm
#define SoRev SOXMREV
#define HasSharedData SharedDataSeparation
#define LibHeaders NO

#include <Library.tmpl>

SpecialLibObjectRule(Xmos.o,$(ICONFIGFILES),$(SRCH_DEFINES))
SpecialLibObjectRule(VirtKeys.o,$(ICONFIGFILES),$(BINDINGS_DEF))
SpecialLibObjectRule(ImageCache.o,$(ICONFIGFILES), $(XPM_INCLUDES) $(XPM_DEFINES) )
```

```
#if DoSharedLib && SharedDataSeparation
SpecialObjectRule(sharedlib.o,,$(SHLIBDEF))
#endif

BuildIncludes($(HEADERS),Xm,..)

InstallMultiple($(HEADERS),$(USRINCDIR)/Xm)
```

The `#define` lines that precede the inclusion of *Library.tmpl* determine what *Library.tmpl* does. The `LibName` and `SoRev` lines define the library name and revision number. The `HasSharedData` line specifies whether or not special rules are needed for a shared-data target. The `LibHeaders` line prevents *Library.tmpl* from generating an `includes` target. (This is necesssary because the target that would be generated is incorrect. Instead, rules to generate the `includes` target are invoked explicitly.) Given the initial `#define`s, *Library.tmpl* invokes most of the library-building rules automatically. The lines that follow the inclusion of *Library.tmpl* are for those rules that are not taken care of and that must be repeated from the R5 section.

Additional Note for X11R6.1

The Motif release notes say little about using the configuration files in the Motif distribution to build Motif for use with X11R6.1. They do mention that such use is unsupported and suggest that if you want to try it, you should change the definition of `XTop` in *config/site.def* from this:

```
#define XTop    top_of_X11r5_source_tree
```

To this:

```
#define UsingR6Source YES
#if defined(UsingR6Source)
#define XTop    top_of_X11r6_source_tree/xc
#else
#define XTop    top_of_X11r5_source_tree
#endif
```

There are two problems here. First, the definition of `UsingR6Source` as `YES` implies that the macro is a Boolean, and thus that defining it as `NO` would have the opposite effect of defining it as `YES`. But that's not the case. `UsingR6Source` is tested only to see whether or not it's defined, not what its value is.

Second, the value of `XTop` for the R6 case isn't quite correct. The *xc* directory isn't UNDER the top of the R6.1 source tree, it IS the top of the R6.1 source tree. The release notes might be clearer were they to say that you should set `XTop` like this:

```
#define UsingR6Source /* as nothing */
#if defined(UsingR6Source)
#define XTop    top_of_X11r6_source_tree
#else
#define XTop    top_of_X11r5_source_tree
#endif
```

where `top_of_X11r6_source_tree` and `top_of_X11r5_source_tree` are pathnames ending in `xc` and `mit`, respectively. For example, I might define `XTop` this way:

```
#define UsingR6Source /* as nothing */
#if defined(UsingR6Source)
#define XTop    /src/extern/X-stuff/X11R6.1/xc
#else
#define XTop    /src/extern/X-stuff/mit
#endif
```

Generating Makefiles for Motif-Based Projects

If you've installed configuration files under the *imboot* configuration root as described in the section "Alternative Methods of Configuring Motif," you should be able to use them not only to configure Motif itself, but also to configure Motif-based projects. The command for doing this depends on which set of configuration files you want to use:

```
% imboot -c Motif-2.0          To use files from the Motif distribution
% imboot -c Motif-2.0R5        To use alternate X11R5 files
% imboot -c Motif-2.0R6.1      To use alternate X11R6.1 files
```

Follow the *imboot* command with *make Makefile* if the project has subdirectories.

Note that you can configure a project using one set of files, then reconfigure it using a different set of files by running *imboot* multiple times with different arguments following the −*c* flag. This allows you to experiment with the portability of a project to both R5 and R6.1:

```
% imboot -c Motif-2.0R5
% make clean ; make
% imboot -c Motif-2.0R6.1
% make clean ; make
```

To Define UseInstalledMotif, or Not?

One issue that remains unaddressed is whether or not to define `UseInstalledMotif`. You don't need to define this macro as long as `MTop` is set correctly. However, to build Motif-based projects with `UseInstalledMotif` undefined, you must have the header files and libraries in place in the Motif source tree. That means if you run *make clean* in the Motif source tree, you can no longer build Motif-based projects. (This is the same problem that occurs with building X11-based projects using X11 header files and libraries from the X11 source tree.) To avoid this, you can build Motif once and install it, then define `UseInstalledMotif` in your installed configuration files. That way you can use the files for Motif-based projects.

To define `UseInstalledMotif`, put this in *site.def*:

```
#ifndef UseInstalledMotif
#define UseInstalledMotif /* force on if not already on */
#endif
```

You might notice that when `UseInstalledMotif` is defined, header files and libraries are still built if you recompile the Motif distribution. This is normal.

Obtaining Software

Some parts of this document assume *imake* and *imboot* are installed publicly. The most recent version of *imake* is available as part of the current X11 distribution, which can be obtained at the following locations or their mirror sites:

```
http://www.x.org
ftp://ftp.x.org
```

Alternatively, you can obtain the *itools* distribution, a distribution containing just *imake* and related configuration programs. *itools* is available in the *imake-book* archive:

```
http://www.primate.wisc.edu/software/imake-book
ftp://ftp.primate.wisc.edu/software/imake-book
```

imboot is available as part of the *itools* distribution, or in standalone form from the *imake-stuff* archive:

```
http://www.primate.wisc.edu/software/imake-stuff
ftp://ftp.primate.wisc.edu/software/imake-stuff
```

The *imake-stuff* archive is also the location of the *motif-support* distribution, which contains instructions and patches for modifying configuration files. In particular, these patches implement the modifications described in the section "Alternative Methods of Configuring Motif."