An *Imake* tutorial

Mark Moraes, Computer Systems Research Institute,
University of Toronto
moraes@csri.toronto.edu

Revised September 25, 2002

`$(TOP)` *in the document refers to the top level directory of the X Windows source tree.*

1 Introduction

Imake is a tool for generating Makefiles [1] for large software distributions.

Imake uses the C preprocessor on macro-makefiles (called *Imakefiles*) to generate Makefiles. It uses a predefined template file for default values and macros. The *imake* program is part of, and used by the X Windows System software distribution. (The following description uses fragments from the X Windows distribution as examples — you might want to look at the files in `$(TOP)/config` for more detailed information. The source code for `imake` is in `$(TOP)/util/imake`, and includes a `sample.rules` file as well as a manual page)

This document attempts to provide a working introduction to *imake*. In particular, it provides an annotated tour of some of the relevant files, and through some example em Imakefiles. For a more precise description of *imake*'s workings, see the manual page.

2 Why *Imake*

Makefiles have some limitations for large software distributions, distributed over multiple directories, viz.

- A single makefile doesn't handle multiple directories well, which means that each directory needs a makefile. At that point, to change a default value, for example the directory where the binaries are to be installed, you have to change all the makefiles.
- Typically, makefiles are used to achieve a limited set of actions, like create a binary from a set of object files, or generate a library from some object files, etc. These can be encapsulated in rules that are slightly more complex that the ones *make* supports (which are only suffix based)
- A production makefile which supports all the actions necessary for a development environment, can get fairly long — typically, you want a *clean* target, an *install* target, a *depend* target, and an *all* target.
- For a system that is ported to many different systems, there are bound to be differences in the requirements of the systems, that cannot be dealt with solely at the C source level. For instance, some System V Unix [2] machines provide a BSD compatibility library which must be linked in with `-lbsd`, and BSD includes that must the C compiler must be told about with a `-I/usr/include/bsd option`. Vanilla *make* allows no facilities for changes like this, and you end up having multiple makefiles, with suffixes to indicate the type of system being used.

[1] Familiarity with the *make* program and C preprocessor syntax is assumed in this document. Those unfamiliar with these are advised to read their system manuals for *make* and *cpp* and experiment with them

[2] Unix is a trademark of AT&T

3 A simple example

For example, for a simple program, `xcalc`, the Imakefile looks something like:

```
LOCAL_LIBRARIES = $(XLIB)
  SYS_LIBRARIES = -lm
#ifdef MacIIArchitecture
        DEFINES = -DIEEE
#endif /* MacIIArchitecture */
           SRCS = xcalc.c sr.c
           OBJS = xcalc.o sr.o

ComplexProgramTarget(xcalc)
```

This will generate a Makefile which will contain an `all` target, and an `xcalc` target, both of which will result in the program `xcalc` being compiled and linked. The Makefile will also contain an `install` target, which will install the binaries in specified destination directories, an `install.man` target, which will install the manual pages for the program, a `clean` target, which will remove all compiler generated files, editor backup files, etc., and a `depend` target which will invoke the *makedepend* program to generate Makefile depenencies.

It will also contain a `Makefile` target, so you only need to type `make Makefile` to regenerate the Makefile should the Imakefile change.

The real work lies in `ComplexProgramTarget`, which is defined in the template as a `#define` macro, which expands out to the make rules needed to generate program *xcalc*, from the variables `SRCS` and `OBJS`. Let's take a look at that particular rule.

```
/*
 * This target is the general interface for building a single program
 */
#define ComplexProgramTarget(program)                                  @@\
        PROGRAM = program                                              @@\
                                                                       @@\
AllTarget(program)                                                     @@\
                                                                       @@\
program: $(OBJS) $(LOCAL_LIBRARIES)                                    @@\
        $(RM) $@                                                       @@\
        $(CC) -o $@ $(OBJS) $(LOCAL_LIBRARIES) $(LDFLAGS) $(SYSLAST_LIBRARIES)          @@\
                                                                       @@\
relink::                                                               @@\
        $(RM) $(PROGRAM)                                               @@\
        $(MAKE) $(MFLAGS) $(PROGRAM)                                   @@\
                                                                       @@\
InstallProgram(program,$(BINDIR))                                      @@\
InstallManPage(program,$(MANDIR))                                      @@\
DependTarget()                                                         @@\
clean::                                                                @@\
        $(RM) $(PROGRAM)
```

Ignoring `AllTarget()` for now, we see that `program` is defined to depend on the `OBJS`, and `LOCAL_LIBRARIES`, and to make it, the old version is removed (a precaution to save disk space, among other things), and then, `program` is linked from `OBJS`, `LOCAL_LIBRARIES`, and `SYSLAST_LIBRARIES`. (`LDFLAGS` are link flags)

An additional target `relink::` is also generated, which can be used if libraries change. `InstallProgram` generates the the `install::` target, which puts the program in `BINDIR`, `InstallManPage` installs the manual page, `DependTarget` generates the target for *make depend* which invokes a *makedepend* program to follow all `#includes` in the source. These macros are listed below — they aren't very complex.

```
/*
 * Install a man page.
 */
#define InstallManPage(file,dest)                                   @@\
InstallManPageLong(file,dest,file)

/*
 * Install a program
 */
#define InstallProgram(program,dest)                                @@\
install:: program                                                   @@\
        $(INSTALL) -c $(INSTALLFLAGS) program dest


/*
 * This makes the depend target given OBJS.
 */
#define DependTarget()                                              @@\
depend:: $(DEPEND)                                                  @@\
                                                                    @@\
depend::                                                            @@\
        $(DEPEND) -s "# DO NOT DELETE" -- $(CFLAGS) -- $(SRCS)      @@\
                                                                    @@\
$(DEPEND):                                                          @@\
        @echo "making $@"; \                                        @@\
        cd $(DEPENDSRC); $(MAKE)                                    @@\


#define AllTarget(depends)                                          @@\
all:: depends
```

Note the use of make variables like `$(INSTALL)` to describe programs that are used - this makes it possible to define these in the templates, and thus enhances portability.

Also, note that the make targets use a conditional dependency branch with double colons (::) rather than the more common single colon (:). This allows multiple make targets.

Based on all this, the Imakefile expands to the following Makefile:

```
LOCAL_LIBRARIES = $(XLIB)
  SYS_LIBRARIES = -lm

           SRCS = xcalc.c sr.c
           OBJS = xcalc.o sr.o

 PROGRAM = xcalc

all:: xcalc

xcalc: $(OBJS) $(LOCAL_LIBRARIES)
        $(RM) $@
        $(CC) -o $@ $(OBJS) $(LOCAL_LIBRARIES) $(LDFLAGS) $(SYSLAST_LIBRARIES)

relink::
        $(RM) $(PROGRAM)
        $(MAKE) $(MFLAGS) $(PROGRAM)
```

```
install:: xcalc
        $(INSTALL) -c $(INSTALLFLAGS) xcalc $(BINDIR)

install.man:: xcalc.man
        $(INSTALL) -c $(INSTMANFLAGS) xcalc.man $(MANDIR)/xcalc.n

depend:: $(DEPEND)

depend::
        $(DEPEND) -s "# DO NOT DELETE" -- $(CFLAGS) -- $(SRCS)

$(DEPEND):
        @echo "making $@"; \
        cd $(DEPENDSRC); $(MAKE)

clean::
        $(RM) $(PROGRAM)
```

We've omitted a few standard definitions for `CFLAGS, CC, DEPEND, INSTALL, RM, LDFLAGS` etc. These are installation and system dependent, and form part of the Imake template. They will precede the above text.

4 Generating the Makefile for the first time

Invoking *Imake* for the first time is a little complex because you have to tell it where to find the template, as well as define the top level directory. The simplest way to do it is to use the `ximake` script that should have been installed along with the rest of `X` and `imake`. The invocation is simply

```
ximake
```

5 The *Imake* template

For a more portable setup, it is possible to split the template file into the following components:

- the rules
- the system-dependent macros, which vary depending on the specific architecture/operating-system combination, things like the compiler name, flags.
- the site dependent variables — where to install the binaries, libraries etc.

For example the X distribution has an `Imake.rules` file, which defines various rules like `ComplexProgramTarget()`, system dependent files of the form *machinetype*`.cf`, eg. `sun.cf, ultrix.cf`, ..., and a site dependent variables file called `site.def`. The template looks something like the following: (This is abbreviated to keep the flavour but omit much detail, deemed unnecessary for the purposes of this tutorial — annotations in C style comments)

```
#ifdef sun
#define MacroIncludeFile "sun.cf"
#define MacroFile sun.cf
#undef sun
#define SunArchitecture
#endif /* sun */
```

```
/*
 *  The macrofile sun.cf contains definitions specific to Sun
 *  machines --- examples from it are after this file
 */
/**/###########################################################################
/**/# platform-specific configuration parameters - edit MacroFile to change
#include MacroIncludeFile

/**/###########################################################################
/**/# site-specific configuration parameters - edit site.def to change
#include "site.def"

/*****************************************************************************
 *                                                                           *
 *                          DEFAULT DEFINITONS                               *
 *                                                                           *
 * The following section contains defaults for things that can be overridden *
 * in the various .macros files and site.def.  DO NOT EDIT!                  *
 *                                                                           *
 *****************************************************************************/

#ifndef SystemV
#define SystemV                 NO      /* need system 5 style */
#endif
#ifndef BuildServer
#define BuildServer             YES     /* go ahead and build server */
#endif
#ifndef UnalignedReferencesAllowed
#define UnalignedReferencesAllowed NO   /* if arbitrary deref is okay */
#endif
#ifndef BourneShell                     /* to force shell in makefile */
#define BourneShell             /bin/sh
#endif

/*
 *  Lots and lots more variables that can be overridden in the site.def
 *  including:
 */

#ifndef DestDir
#define DestDir /* as nothing */
#endif
#ifndef UsrLibDir
#define UsrLibDir $(DESTDIR)/usr/lib
#endif
#ifndef BinDir
#define BinDir $(DESTDIR)/usr/bin/X11
#endif
#ifndef IncDir
#define IncDir $(DESTDIR)/usr/include/X11
#endif
#ifndef IncRoot
#define IncRoot $(DESTDIR)/usr/include
```

```
#endif
#ifndef AdmDir
#define AdmDir $(DESTDIR)/usr/adm
#endif
#ifndef LibDir
#define LibDir $(USRLIBDIR)/X11
#endif
#ifndef LintlibDir
#define LintlibDir $(USRLIBDIR)/lint
#endif
#ifndef FontDir
#define FontDir $(LIBDIR)/fonts
#endif

/*
 *  This is followed by the actual definitions that will appear in the
 *  Makefile, again, the list is partial.
 */
/*
 * This list must be reflected in the DIRS_TO_BUILD list in the top-level
 * Makefile.
 */
        DESTDIR = DestDir               /* root of install */
      USRLIBDIR = UsrLibDir             /* libraries */
         BINDIR = BinDir                /* programs */
         INCDIR = IncDir                /* header files */
        INCROOT = IncRoot               /* base for header files */
         ADMDIR = AdmDir                /* server log files */
         LIBDIR = LibDir                /* rgb, XErrorDB, etc. */
     LINTLIBDIR = LintlibDir            /* lint libraries */
        FONTDIR = FontDir               /* font directories */

         CFLAGS = $(CDEBUGFLAGS) $(INCLUDES) $(STD_DEFINES) $(DEFINES)
      LINTFLAGS = $(LINTOPTS) $(INCLUDES) $(STD_DEFINES) $(DEFINES) -DLINT
        LDFLAGS = $(CDEBUGFLAGS) $(SYS_LIBRARIES) $(SYSAUX_LIBRARIES)
            TOP = TOPDIR
#ifdef CompileWithInstalled
   EXTENSIONLIB = $(USRLIBDIR)/lib/libXext.a
           XLIB = $(USRLIBDIR)/libX11.a
         XMULIB = $(USRLIBDIR)/libXmu.a
        OLDXLIB = $(USRLIBDIR)/liboldX.a
       XTOOLLIB = $(USRLIBDIR)/libXt.a
         XAWLIB = $(USRLIBDIR)/libXaw.a
       INCLUDES = -I$(INCDIR) -I$(INCROOT)
#else /* ! CompileWithInstalled */
   EXTENSIONLIB = $(EXTENSIONSRC)/lib/libXext.a
           XLIB = $(XLIBSRC)/libX11.a
         XMULIB = $(XMUSRC)/libXmu.a
        OLDXLIB = $(OLDXLIBSRC)/liboldX.a
       XTOOLLIB = $(TOOLKITSRC)/libXt.a
         XAWLIB = $(AWIDGETSRC)/libXaw.a
       INCLUDES = -I$(TOP)
#endif /* CompileWithInstalled */
```

```
/*
 *  The above fragment illustrates how different makefiles can be
 *  generated for the distribution, presumably, once with
 *  CompileWithInstalled not defined in site.def, so that the
 *  distribution can be bootstrapped and built for the first time, then
 *  with CompileWithInstalled defined in site.def so that it uses the
 *  already installed libraries instead of requiring the compiled
 *  objects remain forever in src directories
 */

      MACROFILE = MacroFile
   ICONFIGFILES = $(IRULESRC)/Imake.tmpl \
                  $(IRULESRC)/$(MACROFILE) $(IRULESRC)/site.def
  IMAKE_DEFINES = /* leave blank, for command line use only */
      IMAKE_CMD = $(NEWTOP)$(IMAKE) -TImake.tmpl -I$(NEWTOP)$(IRULESRC) \
                  -s Makefile $(IMAKE_DEFINES)

/*
 *  Now include the actual rules file --- this contains the definitions
 *  of macros like ComplexProgramTarget, as described earlier
 */

#include "Imake.rules"

/*
 *  INCLUDE_IMAKEFILE is defined by the imake program to be the name of
 *  the Imakefile you are processing, usually just Imakefile. It
 *  gets included here.
 */
/**/###########################################################################
/**/# start of Imakefile
#include INCLUDE_IMAKEFILE

/*
 *  Followed by some basic rules for all Makefiles, especially useful
 *  for Imakefiles where all you want to do is make the subdirectories.
 */
/**/###########################################################################
/**/# Imake.tmpl common rules for all Makefiles - do not edit
/*
 * These need to be here so that rules in Imakefile occur first;  the blank
 * all is to make sure that an empty Imakefile doesn't default to make clean.
 */
emptyrule::

CleanTarget()
MakefileTarget()
TagsTarget()

#ifdef IHaveSubdirs

/**/###########################################################################
```

```
/**/# rules for building in SUBDIRS - do not edit

InstallSubdirs($(SUBDIRS))
InstallManSubdirs($(SUBDIRS))
CleanSubdirs($(SUBDIRS))
TagSubdirs($(SUBDIRS))
MakefileSubdirs($(SUBDIRS))

#else

/**/###########################################################################
/**/# empty rules for directories that do not have SUBDIRS - do not edit

install::
	@echo "install done"

install.man::
	@echo "install.man done"

Makefiles::

#endif /* if subdirectory rules are needed */

/**/###########################################################################
/**/# dependencies generated by makedepend
```

The Sun.macros file looks something

```
#define SunOSPlatform     YES           /* set to NO if not running SunOS */
#define OSName            SunOS 3.5
#define OSMajorVersion    3
#define OSMinorVersion    5

/**/# platform:  $XConsortium: Sun.macros,v 1.52 88/10/23 11:00:55 jim Exp $
/**/# operating system:  OSName

#if SunOSPlatform && OSMajorVersion == 3 && OSMinorVersion <= 2
#define OptimizedCDebugFlags /* as nothing */
#endif

BOOTSTRAPCFLAGS =
             AS = as
             CC = cc
            CPP = /lib/cpp
             LD = ld
           LINT = lint
        INSTALL = install
           TAGS = ctags
             RM = rm -f
             MV = mv
             LN = ln -s
         RANLIB = ranlib
RANLIBINSTFLAGS = -t
             AR = ar clq
```

```
             LS = ls
       LINTOPTS = -axz
    LINTLIBFLAG = -C
           MAKE = make
STD_CPP_DEFINES =
    STD_DEFINES =

/* This defines the server you want */
#define XsunServer Xsun
```

There's more in the macros file but the rest of the definitions are X dependent.

Most of the definitions shown here can be overridden in the `site.def` file, so a system administrator can completely reconfigure the setup, if necessary.

6 Simple Imakefiles

We end with a few simple examples to illustrate the rules in the X Imake.rules.

6.1 A single program, one source file

The first is for a simple program that has only one source file `xstring.c` and a corrsponding manual page `xstring.man` It requires the libraries `XAWLIB, XTOOLLIB, and XLIB` (the X Athena widgets, the X Toolkit, and the C library interface to X).

```
LOCAL_LIBRARIES = $(XAWLIB) $(XTOOLLIB) $(XLIB)

SimpleProgramTarget(xstring)
```

6.2 A single program, multiple source files

A slightly more complex Imakefile is the one for `xcalc`, the example described earlier.

```
LOCAL_LIBRARIES = $(XLIB)
  SYS_LIBRARIES = -lm
#ifdef MacIIArchitecture
        DEFINES = -DIEEE
#endif /* MacIIArchitecture */
           SRCS = xcalc.c sr.c
           OBJS = xcalc.o sr.o

ComplexProgramTarget(xcalc)
```

6.3 Multiple programs

An Imakefile for a directory with multiple programs is a little more complicated. `ComplexProgramTarget_1` is similar to `ComplexProgramTarget` but it uses `OBJS1` to link the program instead of `OBJS`. The first argument is the program name, the second is the list of `LOCAL_LIBRARIES`, and the third is the list of `SYSLAST_LIBRARIES`. It is possible to have more complex programs defined (the X Imake rules allow for 3 such targets) called `ComplexProgramTarget_1, ComplexProgramTarget_2,` and `ComplexProgramTarget_3` which link `OBJS1, OBJS2` and `OBJS3` respectively.

`SingleProgramTarget` is a variant that allows multiple programs to be built - it differs from a `ComplexProgramTarget_n` in that it does not generate targets to install the program or the manual page, so it can be used for internal programs as well — i.e. programs that are compiled to generate files or preprocess data for the installable targets.

`CFLAGS`, `LINTFLAGS` and `LDFLAGS` is defined as follows:

```
      CFLAGS = $(CDEBUGFLAGS) $(INCLUDES) $(STD_DEFINES) $(DEFINES)
   LINTFLAGS = $(LINTOPTS) $(INCLUDES) $(STD_DEFINES) $(DEFINES) -DLINT
     LDFLAGS = $(CDEBUGFLAGS) $(SYS_LIBRARIES) $(SYSAUX_LIBRARIES)
```

An Imakefile can redefine some of the right hand side variables - typically,

- `CDEBUGFLAGS` can be `-g` or `-O`, or if you're using the GNU C compiler, `-g -O`,
- `INCLUDES` is usually `-I${TOP}` by default,
- `DEFINES` is left for user defined `-D` flags. (`STD_DEFINES` is defined by the system macros file and should be left alone)
- `SYS_LIBRARIES` or `SYSAUX_LIBRARIES` can be used for `-l` options to link in system libraries. (`SYSLAST_LIBRARIES` is defined by the system macros, typically for portability or emulation.

```
XWSRC=$(CONTRIBSRC)/widgets/Xhp
XPICLIBDIR=$(LIBDIR)/xpic
# The documents go in $(DOCDIR)/xpic - see the install target.
DOCDIR=/local/doc/X11
##############################################################
# -DXPIC is used by the filename completion code.
# -DMAGIC will include code that puts a '#! $(BINDIR)/xpic'
# header on the saved xpic files, and make them executable, so
# you can just execute the saved xpic file and it starts up
# xpic.
# -DDEBUG turns on debugging output - do you really want that?!
# -DDRAWBBOX is also for debugging - it draws the bounding box
# of all gels.
# -DGRAB should be defined if XtAddGrab and XtRemoveGrab can be
# made to work in Minibuf.c and input.c - I can't seem to get them
# to work.
# -DTYPEOUT includes the typeout code - this is still not used by
# xpic - not much point defining it.
DEFINES = -DXPIC -DMAGIC
##############################################################
CDEBUGFLAGS = -g
INCLUDES =  -I$(XWSRC) -Ibitmaps
##############################################################
# Note - You need the HP widgets (Xw) for the X11 Toolkit
# These should be defined in an Imake.tmpl - except that the
# normal Imake.tmpl insists on these being compiled in the
# X directory tree.
# For X11R3, the HP widgets use the R2 Intrinsics, which are
# provided in with the HP src.
XWLIB=$(XWSRC)/lib/libXw.a
XTOOLLIB=$(XWSRC)/lib/libXt.a
# Sigh - we use sin(), cos(), atan2() for arrow.
MATHLIB=-lm

OBJS1 = main.o windows.o xpic.o handlers.o input.o  \
                event.o grid.o error.o spline.o arrow.o newfonts.o \
                util.o gels.o null.o obj_line.o obj_spline.o obj_text.o \
                obj_box.o obj_circ.o obj_ell.o obj_block.o obj_elem.o \
                updown.o text.o isqrt.o ask.o xtypeout.o Minibuf.o Window.o
SRCS1 = main.c windows.c xpic.c handlers.c input.c  \
```

```
                event.c grid.c error.c spline.c arrow.c newfonts.c \
                util.c gels.c null.c obj_line.c obj_spline.c obj_text.c \
                obj_box.c obj_circ.c obj_ell.c obj_block.c obj_elem.c \
                updown.c text.c isqrt.c ask.c xtypeout.c Minibuf.c Window.c
##############################################################
OBJS2 = x2pic.o hash.o
SRCS2 = x2pic.c hash.c
##############################################################
OBJS3 = x2ps.o hash.o
SRCS3 = x2ps.c hash.c

PROGRAMS = xpic x2ps x2pic

/*
 * ComplexProgramTarget(program, local libraries, system libraries)
 *
 * The difference between local libraries and system libraries is that
 * the program will depend on the local libraries, so they must be
 * filenames, while the system libraries are not dependencies, so
 * you can use -l
 */
ComplexProgramTarget_1(xpic,$(XWLIB) $(XTOOLLIB) $(XLIB),$(MATHLIB))

/*
 * SingleProgramTarget(program, objects, local libraries, system libraries)
 *
 * Note that this does not generate install: targets so we do that
 * explicitly
 */
SingleProgramTarget(x2pic,$(OBJS2) $(MALLOC),,)
InstallProgram(x2ps, $(BINDIR))
InstallManPage(x2ps, $(MANDIR))

SingleProgramTarget(x2ps,$(OBJS3) $(MALLOC),,$(MATHLIB))
InstallProgram(x2pic, $(BINDIR))
InstallManPage(x2pic, $(MANDIR))

/*
 * This will install the script x2tex.script as $(BINDIR)/x2tex
 */
InstallScript(x2tex, $(BINDIR))
InstallManPage(x2tex, $(MANDIR))

/*
 * Some additional files we want to install. This is ordinary Make
 * stuff, but we use Imake defined variables like $(INSTALL).
 * $(INSTALLAPPFLAGS) is meant for normal text files.
 */

install::
        -mkdir $(XPICLIBDIR)
        $(INSTALL) $(INSTAPPFLAGS) x2ps.pro $(XPICLIBDIR)
        $(INSTALL) $(INSTAPPFLAGS) x2ps.tra $(XPICLIBDIR)
```

```
        -mkdir $(XPICLIBDIR)/fontdesc
        $(INSTALL) $(INSTAPPFLAGS) fontdesc/xpic $(XPICLIBDIR)/fontdesc
        $(INSTALL) $(INSTAPPFLAGS) fontdesc/x2pic $(XPICLIBDIR)/fontdesc
        $(INSTALL) $(INSTAPPFLAGS) fontdesc/x2tpic $(XPICLIBDIR)/fontdesc
        $(INSTALL) $(INSTAPPFLAGS) fontdesc/x2ps $(XPICLIBDIR)/fontdesc

/*
 * This is a trick that generates a .h file from the Makefile
 * variables. We compare the newly generated file tune.h.new with
 * the old one tune.h, and replace the old one with the new one only
 * if it is different. This will prevent unnecessary compiles, and
 * avoids the need to have .c files depend on the Makefile because
 * you want them recompiled if a -D define changes
 */
tune.h: Makefile
        echo \#define LIBDIR \"$(XPICLIBDIR)\" > tune.h.new
        echo \#define PROGRAMNAME \"$(BINDIR)/xpic\" >> tune.h.new
        echo \#define DUMPDIR \"$(DUMPDIR)\" >> tune.h.new
        -cmp -s tune.h.new tune.h || cp tune.h.new tune.h
```

6.4 A library archive of object files

An Imakefile for a library looks something like:

```
#
# This library contains miscellaneous utility routines and is not part
# of the Xlib standard.
#

    STD_DEFINES = LibraryDefines
    CDEBUGFLAGS = LibraryCDebugFlags
       INCLUDES = -I. -I$(TOP) -I$(TOP)/X11
   INSTALLFLAGS = $(INSTINCFLAGS)
       LINTLIBS = $(LINTXLIB)

#ifdef OsNameDefines
OS_NAME_DEFINES = OsNameDefines
#endif

HEADERS = \
        Xmu.h

SRCS = Atoms.c CrPixFBit.c CvtStdSel.c DefErrMsg.c DrRndRect.c FToCback.c \
        Lookup.c Lower.c RdBitF.c StrToBS.c StrToBmap.c StrToCurs.c \
        StrToJust.c StrToOrnt.c StrToWidg.c

OBJS = Atoms.o CrPixFBit.o CvtStdSel.o DefErrMsg.o DrRndRect.o FToCback.o \
        Lookup.o Lower.o RdBitF.o StrToBS.o StrToBmap.o StrToCurs.o \
        StrToJust.o StrToOrnt.o StrToWidg.o

all::

NormalLibraryTarget(Xmu,$(OBJS))
LintLibraryTarget(Xmu,$(SRCS))
```

```
InstallLibrary(Xmu,$(USRLIBDIR))
InstallLintLibrary(Xmu,$(LINTLIBDIR))

InstallMultiple($(HEADERS),$(INCDIR))

DependTarget()

NormalLintTarget($(SRCS))
```

Sometimes, it is nice to be able to generate libraries compiled with debugging on, and libraries compiled with profiling on, so that these can be used by developers. Something like the following, added before the `NormalLibraryTarget`

```
#if DebugLibXmu && ProfileLibXmu
DebuggedAndProfiledLibraryObjectRule()
#else
# if DebugLibXmu
DebuggedLibraryObjectRule()
# else
#  if ProfileLibXmu
ProfiledLibraryObjectRule()
#  else
NormalLibraryObjectRule()
#  endif
# endif
#endif
```

and something like the following added before the `InstallMultiple` would do the trick quite nicely.

```
#if ProfileLibXmu
ProfiledLibraryTarget(Xmu,$(OBJS))
InstallLibrary(Xmu_p,$(USRLIBDIR))
#endif

#if DebugLibXmu
DebuggedLibraryTarget(Xmu,$(OBJS))
#endif
```

If we wanted a debugging library, we'd define the symbol `DebugLibXmu` in the `site.def` file, and the DebuggedLibraryRule would ensure it generated a library target for `libname_d.a`, while if we defined the `ProfileLibXmu` symbol, we would get a profiled library target for `libname_p.a`, in addition to the usual `libname.a`. This sort of configurability is very hard to get from a vanilla makefile.

6.5 Subdirectories

The following Imakefile is for a directory that only contains subdirectories, and we want to make each of the subdirectories:

```
#define IHaveSubdirs
#define PassCDebugFlags 'CDEBUGFLAGS=$(CDEBUGFLAGS)'

SUBDIRS = X \
        oldX \
        Xt \
```

```
        Xmu \
        Xaw

MakeSubdirs($(SUBDIRS))
DependSubdirs($(SUBDIRS))
MakeLintLibSubdirs($(SUBDIRS))
MakeLintSubdirs($(SUBDIRS),install.ln,install.ln)
```

This will have the `Makefiles` target defined as well as `Makefile` and using the `make Makefiles` command will cause it to regenerate the Makefiles in the subdirectories. It does this by prepending a "../" in front of the `TOP` variable, which means that `TOP` should be a relative path. (This is the only reason for `TOP` to be a relative path - if there are no subdirectories, `TOP` can easily be an absolute path.

7 Miscellaneous

Imake puts null comments of the form in front of normal *make* style comments i.e. the `#` to protect them from the C preprocessor. They survive unscathed in the resulting Makefile, while C style comments vanish. This provides a good way to distinguish betwen Imakefile comments and comments that you want to appear in the Makefile.

However, it does not protect comments in the files that are included `(site.def, Imake.rules, machine.macros)` so if you want to use `#` style comments in those files, you must protect them yourself with in front of them.

8 History

Imake was originally written by Todd Brunhoff (toddb@tekcrl.crl) for X Windows Version 11 Release 1. The configurations for X Windows Version 11 Releases 2, 3 and 4 were reworked considerably by Jim Fulton (jim@expo.lcs.mit.edu) to isolate the rules, machine dependent information, and site dependent information into separate files.