Building and Installing X11R6.4

Kaleb S. KEITHLEY
The Open Group X Project Team

30 January, 1998

-- --

1. Easy Build Instructions

This quick summary is no substitute for reading the full build instructions later in this document.

Edit **xc/config/cf/site.def** for local preferences. If you want to install somewhere other than **/usr/X11R6.4**, change **ProjectRoot**. (Do *not* use **DESTDIR**.) If you want to build with *gcc* uncomment the **HasGcc2** line. If you have *gcc*, but not *cc*, please read the full build instructions.

If some time has elapsed since the initial release of R6.4, check to see if any public patches have been released. The source tar files may have been updated — check the patch-level line in the bug-report template. If the source in the tar files has not been updated then get all the patches and apply them, following the instructions at the top of each patch. Ignore the rebuild steps in the patch application instructions.

Check the appropriate vendor-specific **.cf** file in **xc/config/cf/** to make sure that *OSMajorVersion*, *OSMinorVersion*, and *OSTeenyVersion* are set correctly for your system. On most systems imake will figure these out automatically; but you may override them in your **xc/config/cf/site.def** if you want.

See if there is a *BootstrapCFlags* mentioned in the comments in the vendor-specific **.cf** file. (Most systems don't have or need one. The BootstrapCFlags in *sun.cf* is for SunOS 4.0.x, so if you're building on SunOS 4.1.x or SunOS 5/Solaris 2 then BootstrapCFlags doesn't apply.) If there isn't one, *cd* to the **xc** directory and type (in csh):

```
% make World >& world.log
```

If there is an applicable **BootstrapCFlags**, take its value and type:

% make World BOOTSTRAPCFLAGS="*value*" >& world.log

Do not call the output file "make.log" when doing "make World". After a successful build, you can install with:

```
% make install >& install.log
```

You can install manual pages with:

```
% make install.man >& man.log
```

While the system is building (or if things fail), read the rest of these installation instructions.

2. Building and Installing R6.4

Historically the MIT X Consortium and The X Consortium, Inc., sample implementation releases have always been source-code-only releases, and this release is no different.

2.1. Introduction

Every release of X has been progressively easier to configure, build, and install than the preceding releases — and we believe this release is the easiest release to build yet. That not withstanding, if things do go amiss during the build we assume that you have the basic skills necessary, and the willingness, to debug any errors that may occur in the build process. When you install, if you're going to use *xdm* or replace your system's old X, we assume you have a basic understanding of your system's initialization process. For Remote Execution (RX, embedding) we assume you that you understand the fundamentals of HTTP, CGI, and HTML. If these assumptions are not correct then you should consider finding someone who has proficiency in these areas to do the build and install for you.

After the release has been out for a while more up to date information about any newly-discovered problems may be found in the *Frequently Asked Questions* posting appearing monthly on the Usenet newsgroup comp.windows.x and xpert mailing list. The FAQ is also available via anonymous FTP from ftp://ftp.x.org/ in the file ftp://ftp.x.org/contrib/faqs/FAQ.Z, or possibly on one of X mirror sites.

2.2. Preparing Your Build System

The source is distributed in four gzip compressed UNIX **T**ape **AR**chive (tar) files. You will need about 200 Mb of disk space in order to unpack and build the release. Installing requires an additional 30-50 Mb assuming you have shared libraries (80-100 Mb without).

On non-UNIX systems you'll need a utility that can extract gzip compressed tar files to extract the sources. There are several to chose from, we do not make recommendations about which one you should use.

Release 6.4 sources are distributed among the tar files as follows:

tog-1.tar	contains everything in xc/ that isn't in the other tar files
tog-2.tar	contains xc/fonts
tog-3.tar	contains xc/doc/specs, xc/util
tog-4.tar	contains xc/doc/hardcopy

If you define *BuildFonts* to NO in your **site.def** file, then you only need to unpack tog-1.tar to build. If you build fonts, then you will also need tog-2.tar to build. If you already have the fonts from prior releases you can use those instead of downloading them again. We presume that you know how to copy or move them from your old source tree to the R6.4 source tree.

2.3. Unpacking the Distribution

Create a directory to hold the sources and *cd* into it:

% mkdir *sourcedir*
% cd *sourcedir*

Then for each tar file **tog−*.tar.gz**, execute this:

% gunzip −c *ftp-dir*/tog−*N*.tar.gz | tar xf −

or if you have GNU's tar (FreeBSD, NetBSD, OpenBSD, or Linux too)

% tar xzf *ftp-dir*/tog−*N*.tar.gz

2.4. Applying Patches

If some time has elapsed since the initial release of R6.4, check to see if any public patches have been released. The source tar files may have been updated — check the patch-level line in the bug-report template. If the source in the tar files has not been updated then get all the patches and apply them, following the instructions at the top of each patch. Ignore the rebuild steps in the patch application instructions.

See the section "Public Patches" later in this document.

Then continue here.

2.5. Configuration Parameters (Imake Variables)

This release, like all the releases before it, uses *imake*, a utility for creating system-specific Makefiles from system-independent Imakefiles. Almost every directory in the release contains an **Imakefile**. System-specific configuration information is located in **xc/config/cf/**, which is used by the *imake* program every time a **Makefile** is generated in the source tree.

Most of the configuration work prior to building the release is to set parameters (imake variables) so that *imake* will generate correct Makefiles. If you're building on one of the supported systems almost no configuration work should be necessary.

You should define your configuration parameters in **xc/config/cf/site.def**. We provide an empty **site.def** file and a **site.sample** file. The **site.sample** file is a suggested **site.def** file — use it at your own risk.

Any public patches we release will never patch **site.def**, so you can be assured that applying a public-patch will not corrupt your site.def file. On rare occasion you may need to make the change in your vendor-specific **.cf** file; but you should avoid doing that if at all possible because any patch we might release could conceivably patch your vendor-specific **.cf** file and your change may be lost or garbled. You can override most of the things in your vendor-specific **.cf** file in your **site.def** file. (If you can't, it's a bug — please file a bug-report.)

On the systems we use here, imake will automatically determine the *OSMajorVersion*, *OSMinorVersion*, and *OSTeenyVersion* for your system. If your system isn't one of the systems we build on here, or you want to build for a different version of your operating system, then you can override them in the appropriate entry in your **site.def** file.

The **site.def** file has two parts, one protected with "#ifdef BeforeVendorCF" and one with "#ifdef AfterVendorCF". The file is actually processed twice, once before the **.cf** file and once after. About the only thing you need to set in the "before" section is **HasGcc2**; just about everything else can be set in the "after" section.

The **site.sample** also has commented out support to include another file, **host.def**. This scheme may be useful if you want to set most parameters site-wide, but some parameters vary from machine to machine. If you use a symbolic link tree, you can share **site.def** across all machines, and give each machine its own copy of **host.def**.

The config parameters are listed in **xc/config/cf/README**, but here are some of the new or more common parameters that you may wish to set in your **xc/config/cf/site.def**.

ProjectRoot
: The destination where X will be installed. This variable needs to be set before you build, as some programs that read files at run-time have the installation directory compiled in to them.

HasVarDirectory
: Set to **NO** if your system doesn't have /var or you don't want certain files to be installed in *VarDirectory*.

VarDirectory
: The location of site editable configuration and run-time files. Many sites prefer to install their X binaries on *read-only* media — either a disk slice (partition) that's mounted *read-only* for added security, an NFS volume mounted *read-only* for security and/or improved VM paging characteristics, or from a *live filesystem* on a CD-ROM. In order to simplify things like installing *app-default* files for locally built software, and allowing editing of miscellaneous configuration and policy files, and to allow xdm to create its master Xauthority file, some directories under *$ProjectRoot*/**lib/X11**

are actually installed in **/var/X11**, and *$ProjectRoot*/**lib/X11** contains symlinks to the directories in **/var/X11**.

HasGcc2
Set to **YES** to build with *gcc* version 2.x instead of your system's default compiler.

BuildXInputExt
Set to **YES** to build the X Input Extension. This extension requires device-dependent support in the X server, which exists only in *Xhp* and *XF86_** in the sample implementation.

DefaultUsrBin
This is a directory where programs will be found even if PATH is not set in the environment. It is independent of ProjectRoot and defaults to **/usr/bin**. It is used, for example, when connecting from a remote system via *rsh*. The *rstart* program installs its server in this directory.

InstallServerSetUID
Some systems require the X server to run as root to access the devices it needs. If you are on such a system and will not be using *xdm*, you may set this variable to **YES** to install the X server setuid to root; however the X Project Team strongly recommends that you not install your server suid-root, but that you use xdm instead. Talk to your system manager before setting this variable to **YES**.

InstallXdmConfig
By default set to NO, which suppresses installing xdm config files over existing ones. Leave it set to NO if your site has customized the files in *$ProjectRoot*/**lib/X11/xdm**, as many sites do. If you don't install the new files, merge any changes present in the new files.

MotifBC
Causes Xlib and Xt to work around some bugs in older versions of Motif. Set to **YES** only if you will be linking with Motif version 1.1.1, 1.1.2, or 1.1.3.

GetValuesBC
Setting this variable to **YES** allows illegal XtGetValues requests with NULL ArgVal to usually succeed, as R5 did. Some applications erroneously rely on this behavior. Support for this will be removed in a future release.

The following vendor-specific **.cf** files are in the release but have not been tested recently and hence probably need changes to work: **apollo.cf**, **bsd.cf**, **convex.cf**, **DGUX.cf**, **luna.cf**, **macII.cf**, **Mips.cf**, **moto.cf**, **Oki.cf**, **pegasus.cf**, **x386.cf**. **Amoeba.cf** is known to require additional patches.

The file **xc/lib/Xdmcp/Wraphelp.c**, for XDM-AUTHORIZATION-1, is not included in this release. See ftp://ftp.x.org/pub/R6.4/xdm-auth/README.

2.6. System Build Notes

This section contains hints on building X with specific compilers and operating systems.

If the build isn't finding things right, make sure you are using a compiler for your operating system. For example, a pre-compiled *gcc* for a different OS (e.g. as a cross-compiler) will not have right symbols defined, so *imake* will not work correctly.

2.6.1. gcc

X will not compile on some systems with *gcc* version 2.5, 2.5.1, or 2.5.2 because of an incorrect declaration of memmove() in a gcc fixed include file.

If you are using a *gcc* version prior to 2.7 on Solaris x86, you need to specify **BOOTSTRAPCFLAGS="−Dsun"** in the "make World" command.

If you're building on a system that has an unbundled compiler, e.g. Solaris 2.x, and you do not have the *cc* compiler, you need to contrive to have *cc* in your path in order to bootstrap imake. One way to do this is to create a symlink cc that points to *gcc*.

% cd /usr/local/bin; ln −s *path-to-gcc* cc

Once *imake* has been built all the Makefiles created with it will explicitly use *gcc* and you can remove the symlink. Another way around this is to edit **xc/config/imake/Makefile.ini** to specify *gcc* instead of *cc*.

2.6.2. Other GNU tools

Use of the GNU BinUtils assembler, *as*, and linker, *ld*, is not supported — period! If you have them installed on your system you must rename or remove them for the duration of the R6.4 build. (You can restore them afterwards.)

The system-supplied *make* works just fine for building R6.4 and that's what we suggest you use. If you've replaced your system's *make* with GNU *make* then we recommend that you restore the system *make* for the duration of your R6.4 build. After R6.4 is done building you can revert to GNU make. GNU make on most systems (except Linux, where it is the default make) is not a supported build configuration. GNU make may work for you, and if it does, great; but if it doesn't we do not consider it a bug in R6.4. If, after this admonition, you still use GNU make and your build fails, reread the above, and retry the build with the system's *make* before you file a bug-report.

2.6.3. IBM AIX 4.x

On AIX 4.x, the file **lib/font/Type1/objects.c** must be compiled without optimization (**−O**) or the X server and fontserver will exit when Type 1 fonts are used.

2.6.4. SunOS 4.0.x

SunOS 4.0 and earlier need BOOTSTRAPCFLAGS=-DNOSTDHDRS because it does not have unistd.h and stdlib.h. Do *not* supply a BOOTSTRAPCFLAGS when building any SunOS 4.1 or 5.x (Solaris 2) version.

2.6.5. Linux

On Linux systems imake has preliminary support to automatically determine which Linux distribution you're using. At this time it only automatically detects S.u.S.E. Linux. On other Linux systems you should set the LinuxDistribution parameter in your **xc/config/cf/site.def** — see the **xc/config/cf/linux.cf** file for the list of valid values. On Linux systems imake will also automatically determine which version of libc and binutils your system has. You may override these in your **xc/config/cf/site.def** file.

Many distributions of Linux have poor or no support for ANSI/POSIX/ISO C locale support. If your Linux distribution is one of these you should make certain that the imake variable *LinuxLocaleDefines* is set to **-DX_LOCALE** so that compose processing and other internationalization features will work correctly. To help decide if you should use -DX_LOCALE, look in /usr/share/locale — if it's empty, you should probably use the -DX_LOCALE define.

2.6.6. Microsoft Windows NT

All of the base libraries are supported, including multi-threading in Xlib and Xt, but some of the more complicated applications, specifically *xterm* and *xdm*, are not supported.

There are also some other rough edges in the implementation, such as lack of support for non-socket file descriptors as Xt alternate inputs and not using the registry for configurable parameters like the system filenames and search paths.

The *Xnest* server has been made to run on NT; although it still requires a real X server for output still. A real X server can not be built from these sources — in order to display X applications on a MS-Windows host you will have to acquire a real X Server.

You have several choices for imake's *RmTreeCmd*. Look at the possible definitions in the **xc/config/cf/Win32.cf** file, choose one that's right for you, and add it to your **xc/config/cf/site.def** file.

2.7. The Build

For all the supported UNIX and UNIX-like systems you can simply type (in csh):

```
% make World >& world.log
```

You can call the output file something other than "world.log"; but don't call it "make.log" because files with this name are automatically deleted during the initial "cleaning" stage of the build.

The build can take several hours on older systems, and may take as little as an hour on the faster systems that are available today. On UNIX and UNIX-like systems you may want to run it in the background and keep a watch on the output. For example:

```
% make World >& world.log &
% tail −f world.log
```

If something goes wrong, the easiest thing is to correct the problem and start over again, i.e. typing "make World".

2.7.1. UNIX and UNIX-like systems

Check your vendor-specific **.cf** file; if it doesn't have BootstrapCFlags that apply to your version of the operating system then type (in csh):

```
% make World >& world.log
```

Otherwise type (in csh):

```
% make World BOOTSTRAPCFLAGS="value" >& world.log
```

None of the *supported* operating systems need to use BOOTSTRAPCFLAGS.

2.7.2. Microsoft Windows NT

On NT, make certain your Path, Include, and Lib environment variables are set accordingly. For example here we use the command line compiler in VC++ 4.0 Standard Edition, which is installed in C:MSDE-VSTD. To setup the environment type:

```
> set Path=old-path;C:\MSDEVSTD\bin;C:\path-to-RmTreeCmd
> set Include=C:\MSDEVSTD\include
> set Lib=C:\MSDEVSTD\lib
```

Then to build, at the prompt, type:

```
C:\> nmake World.Win32 > world.log
```

2.8. Installing X

After the build has successfully completed you can install the software by typing the following as root:

```
% make install >& install.log
```

or on Microsoft Windows NT

```
C:\> nmake install > install.log
```

Again, you might want to run this in the background and use *tail* to watch the progress.

You can install the manual pages by typing the following as root:

```
% make install.man >& man.log
```

2.9. Shared Libraries

The version number of some of the the shared libraries has been changed. On SunOS 4, which supports minor version numbers for shared libraries, programs linked with the R6.4 libraries will use the new libraries with no special action required.

On most other modern operating systems the version portion of the library name, i.e. "6.1" portion of "libX11.so.6.1" is a string. Even if it's only one character long, e.g. "1" (as in libX11.so.1) it's still a string. This string uniquely identifies and distinguishes one version of the library from another. Even though all the libraries in this release are compatible with the libraries from previous releases, and there's otherwise no reason to change the version string, we do it to identify which source release the libraries were built from.

An old program that was linked with libXext.so.6.3 won't run if you delete libXext.so.6.3 and install libXext.so.6.4 in its place. In general on these systems you have the following choices:

1. Keep the old versions of the libraries around.
2. Relink all applications with the new libraries.
3. Create a symlink using the old name which points to the new name.

   For example, to have programs that were linked against libXext.so.6.3 use libXext.so.6.4, make this symlink:

   ```
   % cd $ProjectRoot/lib
   % ln −s libXext.so.6.4 libXext.so.6.3
   ```

On some distributions of Linux the run-time loader is broken — requiring that the library's internal SONAME match the *filename* — and the symlink solution won't work. We recommend that you get a new run-time loader which is not broken or recompile your run-time loader to not require that the SONAME match.

2.10. Setting Up xterm

If your **/etc/termcap** and **/usr/lib/terminfo** databases do not have correct entries for *xterm*, use the sample entries provided in the directory **xc/programs/xterm/**. System V users may need to compile and install the **terminfo** entry with the *tic* utility.

Since each *xterm* will need a separate pseudoterminal, you need a reasonable number of them for normal execution. You probably will want at least 32 on a small, multiuser system. On most systems, each pty has two devices, a master and a slave, which are usually named /dev/tty[pqrstu][0-f] and /dev/pty[pqrstu][0-f]. If you don't have at least the ‘‘p’’ and ‘‘q’’ sets configured (try typing ‘‘ls /dev/?ty??’’), you should have your system administrator add them. This is commonly done by running the *MAKEDEV* script in the **/dev** directory with appropriate arguments.

2.11. Starting Servers Automatically at System Boot

The *xfs* and *xdm* programs are designed to be run automatically at system startup. Please read the manual pages for details on setting up configuration files; reasonable sample files are in **xc/programs/xdm/config/** and **xc/programs/xfs/**.

Since *xfs* can serve fonts over the network, you do not need to run a font server on every machine with an X display. You should start *xfs* before *xdm*, since *xdm* may start an X server which is a client of (dependent on) the font server.

2.11.1. On BSD-based systems using /etc/rc or /etc/rc.local

If your system uses an **/etc/rc** or **/etc/rc.local** file at boot time, you can usually enable these programs by placing the following at or near the end of the file:

```
if [ −f $ProjectRoot/bin/xfs ]; then
      $ProjectRoot/bin/xfs & echo −n ’ xfs’
fi

if [ −f $ProjectRoot/bin/xdm ]; then
      $ProjectRoot/bin/xdm; echo −n ’ xdm’
fi
```

On later versions of FreeBSD the preferred way of doing this is to create the directory *$ProjectRoot*/etc/rc.d. Add this directory to the *local_startup* variable defined in /etc/rc.conf, and then create short scripts in this directory to start xfs and xdm.

If you are unsure about how system boot works, or if your system does not use **/etc/rc**, consult your system administrator for help.

2.11.2. On Linux systems

Most Linux distributions have an /etc/inittab entry specifically for xdm. Depending on your distribution this may be *run-level* three, four, or five. To use xdm, edit **/etc/inittab** and find the line which contains *initdefault* and change it from 2 to the appropriate run-level

You Linux distribution may already have a script to start xdm at a particular run-level. For example on S.u.S.E. Linux 5.0 there is the file /sbin/init.d/xdm, and the symlink /sbin/init.d/rc3.d/S30xdm which points to /sbin/init.d/xdm. Change /sbin/init.d/xdm to use *$ProjectRoot/bin/xdm*. You can use the xdm script as a model write an xfs script. Depending on your Linux distribution you may find these files in /etc/init.d instead of /sbin/init.d.

2.11.3. On Digital Unix, HPUX 10, and SVR4 systems

Most systems run xdm by default at some particular run-level of the system. There is a master *init.d* file and a run-level symlink *rc?.d* that points to the master *init.d* file:

Operating System	rc?.d symlink	init.d file
Digital Unix 4.0	/sbin/rc3.d/S95xlogin	/sbin/init.d/xlogin
HPUX 10.20	/sbin/rc3.d/S800xdm	/sbin/init.d/xdm
Solaris 2.[0-4]		
Solaris 2.5	/etc/rc3.d/S99xdm	/etc/init.d/xdm.rc
Solaris 2.6	/etc/rc2.d/S99dtlogin	/etc/init.d/dtlogin
IRIX 6.2	/etc/rc2.d/S98xdm	/etc/init.d/xdm
Unixware	/etc/rc2.d/S69xdm	/etc/init.d/xdm

In general you can edit the *init.d* file to use *$ProjectRoot***/bin/xdm**. You can use the xdm file as a model to write an /etc/rc?.d/S??xfs file to start xfs. Some systems may already have files to start xfs. Starting in Solaris 2.5 Sun uses inetd to start xfs — you should remove the xfs entries from /etc/inetd.conf and /etc/services before adding xfs to the run-level files.

2.11.4. On SystemV-based systems

On systems with a **/etc/inittab** file, you can edit this file to add the lines

xfs:3:once:*$ProjectRoot*/bin/xfs
xdm:3:once:*$ProjectRoot*/bin/xdm

2.12. Using OPEN LOOK applications

You can use the X11R6.x Xsun server with OPEN LOOK applications; but you must pass the **−swapLkeys** flag to the server on startup, or the OPEN LOOK Undo, Copy, Paste, Find, and Cut keys may not work correctly. For example, to run Sun's OpenWindows 3.3 desktop environment with an X11R6 server, use the command:

% openwin −server *$ProjectRoot/bin/Xsun −swapLkeys*

The keysyms reported by keys on the numeric keypad have also changed since X11R5; if you find that OpenWindows applications do not respond to keypad keys and cursor control keys when using an R6 server, you can remap the keypad to generate R5 style keysyms using the following *xmodmap* commands:

keysym Pause = F21
keysym Print = F22
keysym Break = F23

```
keysym KP_Equal = F24
keysym KP_Divide = F25
keysym KP_Multiply = F26
keysym KP_Home = F27
keysym KP_Up = Up
keysym KP_Prior = F29
keysym KP_Left = Left
keycode 100 = F31
keysym KP_Right = Right
keysym KP_End = F33
keysym KP_Down = Down
keysym KP_Next = F35
keysym KP_Insert = Insert
keysym KP_Delete = Delete
```

2.13. Rebuilding after Patches

Eventually you are going to make changes to the sources, for example by applying any public patches that may be released or to fix any bugs you may have found.

If only source files are changed, rebuild by going to the base of your source tree **xc** and typing:

```
% make >& make.log
```

If there are imake configuration file changes, the best thing to do is type:

```
% make Everything >& every.log
```

"Everything" is similar to "World" in that it rebuilds every **Makefile**, but unlike "World" it does not delete the existing objects, libraries, and executables, and only rebuilds what is out of date.

2.14. Formatting the Documentation

The PostScript files in **xc/doc/hardcopy** can be generated from the sources in **xc/doc/specs**. Most of the documentation is in troff using the −ms macros. The easiest way to format it is to use the Imakefiles provided.

Set the name of your local troff program by setting the variable **TroffCmd** in **xc/config/cf/site.def**. Then build the Makefiles:

```
cd xc/doc
make SUBDIRS=specs Makefiles
```

Finally, go to the directory you are interested in and type "make" there. This command will generate **.PS** files. You can also generate text files by specifying the document name with a **.txt** extension as a *make* target, e.g., "make icccm.txt".

3. Public Patches

The Open Group X Project Team may from time to time issue public patches for this release to fix any serious problems that are discovered. Such fixes are a subset of fixes available to X Project Team members. Public patches are available via anonymous FTP from ftp://ftp.x.org/pub/R6.4/fixes, or from your local X mirror site. Check the site closest to you first.

You can determine which public patches have already been applied to your source tree by examining the "VERSION" line of **xc/bug-report**. The source in the tar files you have may already have some patches applied; you only need to apply later patches. If you try to apply patches out of order or apply patches that are already in your tree, *patch* will tell you that you have the wrong version and not apply the patch.

Source for the *patch* program is in **xc/util/patch/**. The *patch* program included on some systems may not support all the options this version has. If you have problems applying patches, or if you're otherwise in doubt, use this version.

Table of Contents